



API Manager

Documentation

# WSO2 API Manager

## Documentation

### Version 1.9.0

# Table of Contents

1. WSO2 API Manager Documentation .....	5
1.1 About API Manager .....	5
1.1.1 Introducing the API Manager .....	6
1.1.2 About this Release .....	6
1.2 Getting Started .....	7
1.2.1 Quick Start Guide .....	8
1.2.2 Downloading the Product .....	39
1.2.3 Installation Prerequisites .....	39
1.2.4 Installing the Product .....	41
1.2.4.1 Installing on Linux or OS X .....	42
1.2.4.2 Installing on Solaris .....	43
1.2.4.3 Installing on Windows .....	44
1.2.4.4 Installing as a Linux Service .....	47
1.2.4.5 Installing as a Windows Service .....	49
1.2.5 Building from Source .....	54
1.2.6 Running the Product .....	55
1.2.7 Upgrading from the Previous Release .....	59
1.2.8 Get Involved .....	63
1.2.8.1 WSO2 GitHub Repositories .....	65
1.3 User Guide .....	68
1.3.1 Key Concepts .....	69
1.3.2 Tutorials .....	91
1.3.2.1 Developer Portal .....	91
1.3.2.1.1 Create and Publish an API .....	91
1.3.2.1.2 Create an API with an Inline Script .....	96
1.3.2.1.3 Edit an API Using the Swagger UI .....	100
1.3.2.1.4 Add API Documentation .....	110
1.3.2.1.5 Manage the API Lifecycle .....	119
1.3.2.1.6 Publish to Multiple External API Stores .....	129
1.3.2.1.7 Engage a new Throttling Policy .....	132
1.3.2.1.8 Change the Default Mediation Flow of API Requests .....	134
1.3.2.1.9 Map the Parameters of your Backend URLs with the API Publisher URLs .....	146
1.3.2.1.10 Convert a JSON Message to SOAP and SOAP to JSON .....	152
1.3.2.1.11 Publish through Multiple API Gateways .....	168
1.3.2.1.12 Enforce Throttling and Resource Access Policies .....	171
1.3.2.1.13 Include Additional Headers in the API Console .....	176
1.3.2.2 API Consumer .....	180
1.3.2.2.1 Invoke an API using a SOAP Client .....	180
1.3.2.2.2 Invoke an API using the Integrated API Console .....	184
1.3.2.2.3 Subscribe to an API .....	185
1.3.2.2.4 Use the Community Features .....	190
1.3.2.3 Security .....	195
1.3.2.3.1 Block Subscription to an API .....	195
1.3.2.3.2 Pass a Custom Authorization Token to the Backend .....	200
1.3.3 Configuring the API Manager .....	213

1.3.3.1 Customizing the API Store .....	214
1.3.3.2 Configuring Multiple Tenants .....	219
1.3.3.2.1 Multi Tenant Architecture .....	219
1.3.3.2.2 Managing Tenants .....	222
1.3.3.2.3 Tenant-Aware Load Balancing using WSO2 ELB .....	223
1.3.3.3 Adding Internationalization and Localization .....	225
1.3.3.4 Configuring Single Sign-on with SAML2 .....	226
1.3.3.5 Changing the Default Transport .....	232
1.3.3.6 Configuring Caching .....	234
1.3.3.7 Working with Databases .....	239
1.3.3.7.1 Setting up the Physical Database .....	239
1.3.3.7.2 Managing Datasources .....	282
1.3.3.8 Managing Users and Roles .....	292
1.3.3.8.1 Adding User Roles .....	293
1.3.3.8.2 Adding Users .....	295
1.3.3.9 Configuring User Stores .....	297
1.3.3.9.1 Realm Configuration .....	298
1.3.3.9.2 Changing the RDBMS .....	302
1.3.3.9.3 Configuring Primary User Stores .....	302
1.3.3.9.4 Configuring Secondary User Stores .....	318
1.3.3.10 Directing the Root Context to the API Store .....	320
1.3.3.11 Adding Links to Navigate Between the Store and Publisher .....	321
1.3.3.12 Maintaining Separate Production and Sandbox Gateways .....	322
1.3.3.13 Configuring Transports .....	325
1.3.3.14 Transforming API Message Payload .....	325
1.3.3.15 Sharing Applications and Subscriptions .....	334
1.3.4 Extending the API Manager .....	336
1.3.4.1 Writing Custom Handlers .....	337
1.3.4.2 Adding Mediation Extensions .....	341
1.3.4.3 Integrating with WSO2 Governance Registry .....	341
1.3.4.4 Adding Workflow Extensions .....	342
1.3.4.4.1 Adding an Application Creation Workflow .....	343
1.3.4.4.2 Adding an Application Registration Workflow .....	346
1.3.4.4.3 Adding an API Subscription Workflow .....	349
1.3.4.4.4 Adding a User Signup Workflow .....	352
1.3.4.4.5 Invoking the API Manager from the BPEL Engine .....	355
1.3.4.4.6 Customizing a Workflow Extension .....	356
1.3.4.4.7 Configuring Workflows for Tenants .....	360
1.3.4.4.8 Configuring Workflows in a Cluster .....	367
1.3.4.4.9 Changing the Default User Role in Workflows .....	370
1.3.4.5 Adding new Throttling Tiers .....	370
1.3.4.6 Adding a Reverse Proxy Server .....	372
1.3.4.7 Adding a new API Store Theme .....	372
1.3.5 Working with Security .....	376
1.3.5.1 Passing Enduser Attributes to the Backend Using JWT .....	377
1.3.5.2 Encrypting Secure Endpoint Passwords .....	381
1.3.5.3 Maintaining Logins and passwords .....	382
1.3.5.4 Saving Access Tokens in Separate Tables .....	384

1.3.5.5 Configuring WSO2 Identity Server as the Key Manager .....	385
1.3.5.6 Configuring a Third-Party Key Manager .....	386
1.3.6 Working with Statistics .....	391
1.3.6.1 Publishing API Runtime Statistics .....	392
1.3.6.2 Integrating with Google Analytics .....	397
1.3.6.3 Viewing API Statistics .....	400
1.4 Admin Guide .....	407
1.4.1 Deploying and Clustering the API Manager .....	408
1.4.2 Tuning Performance .....	408
1.4.3 Removing Unused Tokens from the Database .....	413
1.4.4 Migrating the APIs to a Different Environment .....	416
1.5 Published APIs .....	419
1.5.1 Publisher APIs .....	420
1.5.2 Store APIs .....	426
1.5.3 Token API .....	433
1.5.3.1 Exchanging SAML2 Bearer Tokens with OAuth2 (SAML Extension Grant Type) .....	434
1.5.3.2 Generating Access Tokens with Authorization Code (Authorization Code Grant Type) ..	441
1.5.3.3 Generating Access Tokens with NT LAN Manager (NTLM Grant Type) .....	442
1.5.3.4 Generating Access Tokens with User Credentials (Password Grant Type) .....	443
1.5.4 WSO2 Admin Services .....	444
1.6 Reference Guide .....	448
1.6.1 Product Profiles .....	449
1.6.2 Default Product Ports .....	450
1.6.3 Changing the Default Ports with Offset .....	453
1.6.4 Error Handling .....	454
1.6.5 WSO2 Patch Application Process .....	455
1.7 FAQ .....	457
1.8 Getting Support .....	465
1.9 Site Map .....	466

# WSO2 API Manager Documentation

Welcome to WSO2 API Manager Documentation! [WSO2 API Manager \(APIM\)](#) is a fully open source, complete solution for creating, publishing and managing all aspects of an API and its lifecycle, and is ready for massively scalable deployments.

Use the descriptions below to find the section you need, and then browse the topics in the left navigation panel. You can also use the **Search** box on the left to find a term in this documentation, or use the search box in the top right-hand corner to search in all WSO2 product documentation.

To download a PDF of this document or a selected part of it, click [here](#) (generate only one PDF at a time). Use the same link to export to HTML or XML.

<b>About API Manager</b>  Introduces WSO2 API Manager, including the business cases it solves, its features, architecture and how to get help.	<b>Getting Started</b>  Instructions to download, install, run and get started quickly with WSO2 API Manager.	<b>Tutorials</b>  A step-by-step guide through the most common use cases.
<b>User Guide</b>  Introduces the features and functionality of the API Manager, solution development and extension points.	<b>Admin Guide</b>  Introduces product deployment and other system administration tasks.	<b>Published APIs</b>  APIs to be used in your applications.

# About API Manager

The topics in this section introduce WSO2 API Manager, including the business cases it solves, its features, and architecture.

- [Introducing the API Manager](#)
- [About this Release](#)

## Introducing the API Manager

As an organization implements SOA, it can benefit by exposing core processes, data and services as APIs to the public. External parties can mash up these APIs in innovative ways to build new solutions. A business can increase its growth potential and partnership advancements by facilitating developments that are powered by its APIs in a simple, decentralized manner.

However, leveraging APIs in a collaborative way introduces new challenges in exercising control, establishing trust, security and regulation. As a result, proper API management is crucial.

WSO2 API Manager overcomes these challenges with a set of features for API creation, publication, lifecycle management, versioning, monetization, governance, security etc. using proven WSO2 products such as [WSO2 Enterprise Service Bus](#), [WSO2 Identity Server](#), and [WSO2 Governance Registry](#). In addition, as it is also powered by the [WSO2 Business Activity Monitor](#) and is immediately ready for massively scalable deployments.

WSO2 API Manager is fully open source and is released under [Apache Software License Version 2.0](#), one of the most business-friendly licenses available today. It provides Web interfaces for development teams to deploy and monitor APIs, and for consumers to subscribe to, discover and consume APIs through a user-friendly storefront. The API Manager also provides complete API governance and shares the same metadata repository as WSO2 Governance Registry. If your setup requires to govern more than APIs, we recommend you to use WSO2 API manager for API governance and WSO2 Governance Registry for the other artefacts. That the default communication protocol of the Key Manager is Thrift.

The WSO2 API Manager is an on-going project with continuous improvements and enhancements introduced with each new release to address new business challenges and customer expectations. WSO2 invites users, developers and enthusiasts to [get involved](#) or get the assistance of our development teams at many different levels through online forums, mailing lists and support options.

## About this Release

### *What is new in this release*

The WSO2 API Manager version **1.9.0** is the successor of version **1.8.0**. It contains the following new features, enhancements and changes:

- Upgraded the Swagger version to 2.0. See [Edit an API Using the Swagger UI](#).
- Ability to expose the API Store, Publisher and Gateway on your own custom URLs.
- Ability to define the API version in the context so that the version appears in the API URL before the API's name. See [Create and Publish an API](#).
- Ability to connect to third-party key management and authentication systems. See [Configuring a Third-Party Key Manager](#).
- Changed the API visibility model when in single vs multi tenant modes. See [API visibility and subscription](#).
- Demo API already created and published in the API Store to be used out of the box. See the [APIM Quick Start Guide](#).
- Improved functionality when publishing an API to multiple Gateways. Instead of an API being published to all available Gateway environments, you can now select the Gateways that you want to publish to through the UI. See [Publish through Multiple API Gateways](#).
- Ability to generate access tokens for APIs protected by scopes via the subscriptions page in the API Store. Also had ability to whitelist a scope. See [OAuth scopes](#).
- Java docs provided with this version.
- Ability to configure statistics with WSO2 BAM at runtime using the Admin Dashboard Web application. The UI provides capability to configure the event receiver and analyser nodes and the statistics datasource. See [Pub](#)

lishing API Runtime Statistics.

- Support for NTLM grant type. See [Generating Access Tokens with NT LAN Manager \(NTLM Grant Type\)](#).
- Multiple new Published APIs.

#### ***Compatible WSO2 product versions***

These are the products that were tested for compatibility with WSO2 APIM 1.9.0:

- WSO2 Governance Registry 4.6.0
- WSO2 Identity Server 5.0.0 SP01
- WSO2 Business Activity Monitor 2.5.0

WSO2 APIM 1.9.0 is based on WSO2 Carbon 4.2.0 and is expected to be compatible with any other WSO2 product that is based on the same Carbon version. If you get any compatibility issues, please [contact team WSO2](#). For information on the third-party software required with APIM 1.9.0, see [Installation Prerequisites](#).

#### ***Deprecated features***

- The integrated WSO2 REST client in the API Store is deprecated in 1.8.0 and retired in 1.9.0. We now encourage users to [use the Swagger Console to invoke your APIs](#).
- WSDL endpoints are deprecated in this version. For any existing APIs with WSDL endpoints, you can see the WSDL option activated in the edit mode.

#### ***Fixed issues***

For a list of fixed issues in this release, see [WSO2 API Manager 1.9.0- Fixed Issues](#).

#### ***Known issues***

For a list of known issues, see [WSO2 API Manager 1.9.0- Known Issues](#).

# Getting Started

The following topics show how to download, install, run and get started quickly with WSO2 API Manager.

- Quick Start Guide
- Downloading the Product
- Installation Prerequisites
- Installing the Product
- Building from Source
- Running the Product
- Upgrading from the Previous Release
- Get Involved

## Quick Start Guide

WSO2 API Manager is a complete solution for publishing APIs, creating and managing a developer community and for routing API traffic in a scalable manner. It leverages the integration, security and governance components from the WSO2 Enterprise Service Bus, WSO2 Identity Server, and WSO2 Governance Registry. In addition, as it is powered by the WSO2 Business Activity Monitor (BAM), the WSO2 API Manager is ready for massively scalable deployments immediately.

### Before you begin,

1. Install [Oracle Java SE Development Kit \(JDK\)](#) version 1.6.24 or later or 1.7.\* and set the `JAVA_HOME` environment variable.
2. [Download WSO2 API Manager](#).
3. Start the API Manager by going to `<APIM_HOME>/bin` using the command-line and executing `wso2server.bat` (for Windows) or `wso2server.sh` (for Linux.)

Let's go through the use cases of the API Manager:

- [Invoking your first API](#)
- [Understanding the API Manager concepts](#)
- Deep diving into the API Manager
  - Creating users and roles
  - Creating an API from scratch
  - Adding API documentation
  - Adding interactive documentation
  - Versioning the API
  - Publishing the API
  - Subscribing to the API
  - Invoking the API
  - Monitoring APIs and viewing statistics

### Invoking your first API

Follow the steps in this section to quickly deploy a sample API, publish it, subscribe to it, and invoking it.

1. Open the API Publisher (<https://<hostname>:9443/publisher>) and log in with **admin/admin** credentials.
2. Click the **Deploy Sample API** button. It deploys a sample API called `WeatherAPI` into the API Manager.

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following menu items: APIs (selected), Browse, Add, All Statistics, My APIs (selected), Subscriptions, and Statistics. The main content area is titled "All APIs" and displays the message "No APIs created yet. Click the button below to get started." Below this message are two buttons: "New API..." and "Deploy Sample API". The "Deploy Sample API" button is highlighted with a red border.

3. Click Weather API to open it.

The screenshot shows the WSO2 API Publisher interface. The sidebar is identical to the previous one. The main content area is titled "All APIs" and includes a search bar with "Filter APIs" and "Search" buttons. A single API card is displayed, featuring a weather icon (sun partially obscured by clouds with rain drops). The card details are: WeatherAPI, 1.0.0, apicreator, 0 Users, and CREATED. The entire API card is highlighted with a red border.

Let's publish this API.

4. Go to the **Lifecycle** tab and note that the **State** is PUBLISHED. The API is already published to the API Store.

APIs / All / WeatherAPI-1.0.0

## WeatherAPI - 1.0.0 [Manage](#)

Lifecycle PUBLISHED

Propagate Changes to API Gateway

Update Reset

5. Log in to the API Store (<https://<hostname>:9443/store>) with **admin/admin** credentials and note that WeatherAPI is visible under the **APIs** menu. Click it to open the API.

APIs Recently Added

WeatherAPI-1.0.0  
apicreator ★★★★★

APIs WeatherAPI  
1.0.0 apicreator

6. The subscription options are on the right-hand side of the page. Select the default **application** and an available tier, and click **Subscribe**.

The screenshot shows the WSO2 API Manager API Store interface. At the top, there's a navigation bar with links for 'API STORE', 'APIs', 'Prototyped APIs', 'My Applications', 'My Subscriptions', 'Forum', 'Statistics', 'Themes', and 'TestUser'. Below the navigation bar is a search bar with the placeholder 'Search API' and a magnifying glass icon. On the left, there's a sidebar with the text 'More APIs from 'apicreator''. The main content area is titled 'WeatherAPI - 1.0.0'. It features a profile picture for 'apicreator', a sun and cloud icon, and a rating of 'N/A'. Below that, it shows the 'Version: 1.0.0', 'Status: PUBLISHED', and 'Updated: 15/May/2015 15:26:35 PM IST'. To the right, there's a section titled 'Applications' with a dropdown set to 'DefaultApplication', and a 'Tiers' dropdown set to 'Unlimited'. A note below says 'Allows unlimited requests'. At the bottom of this section is a blue 'Subscribe' button. Below the main content area are tabs for 'Overview', 'Documentation', 'API Console', and 'Throttling Info'.

- When the subscription is successful, choose to go to the **My Subscriptions** page and click the **Generate keys** button to generate an access token to invoke the API.

The screenshot shows the 'My Subscriptions' page. The top navigation bar is identical to the one in the API Store. The main content area is titled 'Subscriptions'. It has a sub-section titled 'Applications With Subscriptions' with a dropdown set to 'DefaultApplication' and a 'Show Keys' checkbox. Below that is a section titled 'Keys - Production' with a dropdown set to 'DefaultApplication'. It contains a message 'Production keys are not yet generated for this application.' and a blue 'Generate keys' button. To the right is a 'Allowed Domains' section with a dropdown set to 'ALL'. A note below says 'The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.' and a 'Token Validity' field set to '3600 Seconds'. Below this is another section titled 'Keys - Sandbox'.

- You are now successfully subscribed to an API. Let's invoke the API using the integrated Swagger-based API Console.
- Click the **APIs** menu in the API Store again and then click the API to open it. When the API opens, click its **A C o n s o l e** tab.

## WeatherAPI - 1.0.0

admin



<b>Rating:</b>	Your rating: N/A ★★★★★
<b>Version:</b>	1.0.0
<b>Status:</b>	PUBLISHED
<b>Updated:</b>	15/May/2015 16:38:57 PM IST

**Applications**

Select Application...

**Tiers**

Unlimited

Allows unlimited requests

**Subscribe**
[Overview](#) [Documentation](#) [API Console](#) [Throttling Info](#)

Try  On  Environment.

Set Request Header

## WeatherAPI

[Swagger \( /swagger.json \)](#)

default

[Show/Hide](#) [List Operations](#) [Expand Operations](#)
 /

 /

9. Expand the GET method, give the parameter value as "London," and click **Try it out**.

/

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
q	London	Name of the City	query	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			

10. Note the response for the API invocation. It returns the weather in London.

**Response Body**

```
{
  "coord": {
    "lon": -81.23,
    "lat": 42.98
  },
  "sys": {
    "message": 0.013,
    "country": "CA",
    "sunrise": 1431684065,
    "sunset": 1431736888
  },
  "weather": [
    {
      "id": 804,
      "main": "Clouds",
      "description": "overcast clouds",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 28.0
  }
}
```

You have deployed a sample API, published it to the API Store, subscribed to it, and invoked the API using our integrated API Console.

#### Understanding the API Manager concepts

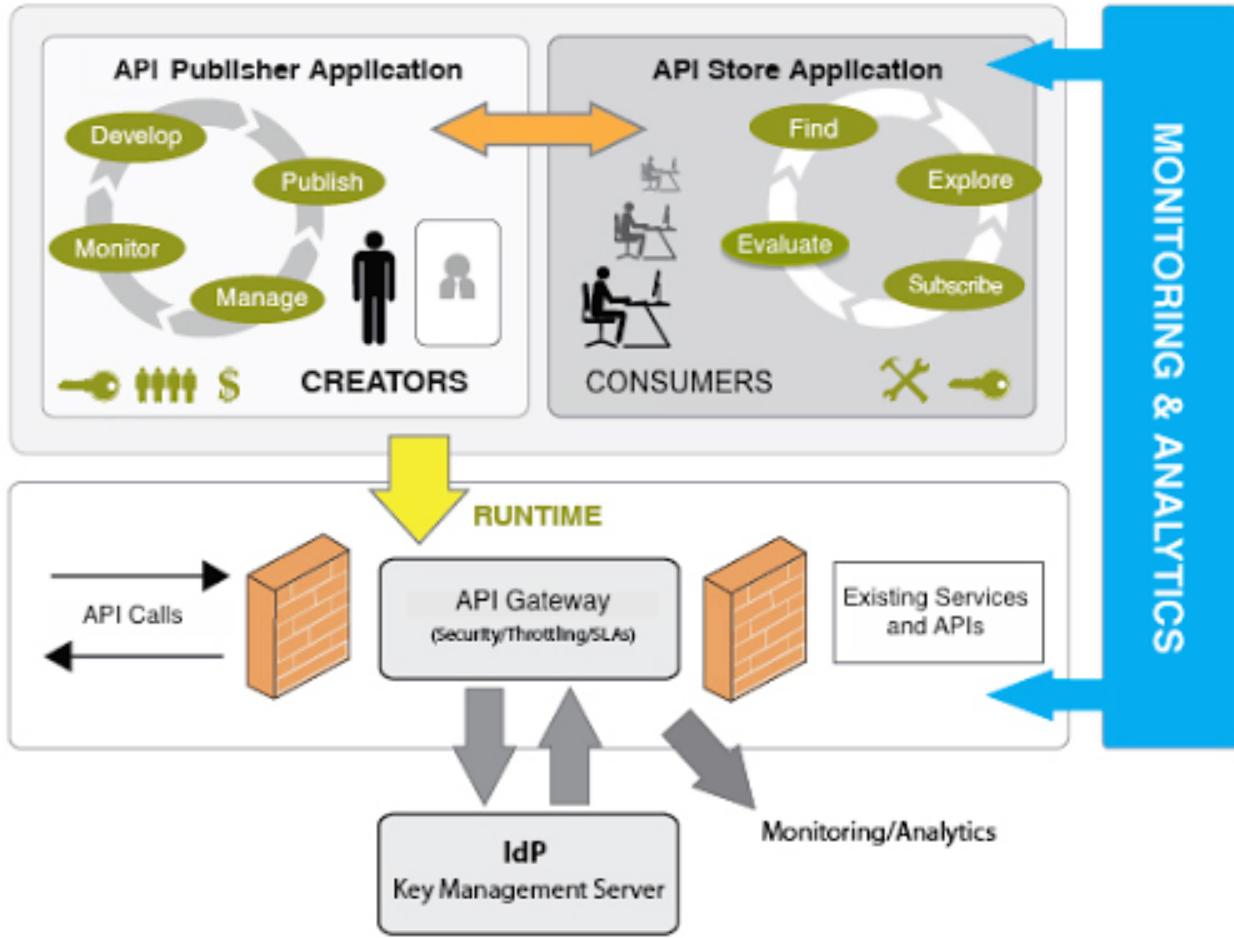
Before we look into the API management activities in detail, let's take a look at the basic API management concepts.

[ [Components](#) ] [ [Users and roles](#) ] [ [API lifecycle](#) ] [ [Applications](#) ] [ [Throttling tiers](#) ] [ [API keys](#) ] [ [API resources](#) ]

#### Components

The API Manager comprises the following components:

- **API Gateway:** Secures, protects, manages, and scales API calls. It is a simple API proxy that intercepts API requests and applies policies such as throttling and security checks. It is also instrumental in gathering API usage statistics. The Web interface can be accessed via `https://<Server Host>:9443/carbon`.
- **Key Manager:** Handles all security and key-related operations. The API Gateway connects with the Key Manager to check the validity of subscriptions, OAuth tokens, and API invocations. The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway.
- **API Publisher:** Enables API providers to publish APIs, share documentation, provision API keys, and gather feedback on features, quality and usage. You access the Web interface via `https://<Server Host>:9443/publisher`.
- **API Store:** Enables API consumers to self register, discover and subscribe to APIs, evaluate them, and interact with API Publishers. You access the Web interface via `https://<Server Host>:9443/store`.
- Additionally, statistics are provided by the monitoring component, which integrates with WSO2 BAM.



## Users and roles

The API manager offers three distinct community roles that are applicable to most enterprises:

- **Creator:** A creator is a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions, how it is exposed by the Gateway, etc.) and uses the API publisher to provision APIs into the API Store. The creator uses the API Store to consult ratings and feedback provided by API users. Creators can add APIs to the store but cannot manage their life cycle (e.g., make them visible to the outside world.)
- **Publisher:** A publisher manages a set of APIs across the enterprise or business unit and controls the API life cycle and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **Consumer:** A consumer uses the API Store to discover APIs, see the documentation and forums, and rate/comment on the APIs. Consumers subscribe to APIs to obtain API keys.

## API lifecycle

An API is the published interface, while the service is the implementation running in the backend. APIs have their own life cycles that are independent of the backend services they rely on. This life cycle is exposed in the API Publisher Web interface and is managed by the publisher role.

The following stages are available in the default API life cycle:

- **CREATED:** API metadata is added to the API Store, but it is not visible to subscribers yet, nor deployed to the API Gateway.
- **PROTOTYPED:** The API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. Users can try out a prototyped API without subscribing to it.

- **PUBLISHED:** The API is visible in the API Store and available for subscription.
- **DEPRECATED:** The API is still deployed in the API Gateway (i.e., available at runtime to existing users) but not visible to subscribers. You can deprecate an API automatically when a new version of it is published.
- **RETIRED:** The API is unpublished from the API Gateway and deleted from the Store.
- **BLOCKED:** Access to the API is temporarily blocked. Runtime calls are blocked, and the API is not shown in the API Store anymore.

You can manage the API and service life cycles in the same governance registry/repository and automatically link them. This feature is available in WSO2 Governance Registry (version 4.5 onwards).

## Applications

An application is primarily used to decouple the consumer from the APIs. It allows you to do the following:

- Generate and use a single key for multiple APIs.
- Subscribe multiple times to a single API with different SLA levels.

You create an application to subscribe to an API. The API Manager comes with a default application, and you can also create as many applications as you like.

## Throttling tiers

Throttling tiers are associated with an API at subscription time. They define the throttling limits enforced by the API Gateway, e.g., 10 TPS (transactions per second). You define the list of tiers that are available for a given API at the publisher level. The API Manager comes with three predefined tiers (Gold/Silver/Bronze) and a special tier called `Unlimited`, which you can disable by editing the `<TierManagement>` element of the `<APIM_HOME>/repository/conf/api-manager.xml` file.

## API keys

The API Manager supports two scenarios for authentication:

- An access token is used to identify and authenticate a whole application.
- An access token is used to identify the final user of an application (for example, the final user of a mobile application deployed on many different devices).

**Application access token:** Application access tokens are generated by the API consumer and must be passed in the incoming API requests. The API Manager uses the OAuth2 standard to provide key management. An API key is a simple string that you pass with an HTTP header (e.g., "Authorization: Bearer NtBQkXoKElu0H1a1fQ0DWfo6IX4a") and it works equally well for SOAP and REST calls.

Application access tokens are generated at the application level and valid for all APIs that you associate to the application. These tokens have a fixed expiration time, which is set to 60 minutes by default. You can change this to a longer time, even for several weeks. Consumers can regenerate the access token directly from the API Store. To change the default expiration time, you open the `<APIM_HOME>/repository/conf/identity.xml` file and change the value of the element `<ApplicationAccessTokenDefaultValidityPeriod>`. If you set a negative value, the token never expires.

**Application user access token:** You generate access tokens on demand using the Token API. In case a token expires, you use the Token API to refresh it.

Application user access tokens have a fixed expiration time, which is 60 minutes by default. You can update it to a longer time by editing the `<ApplicationAccessTokenDefaultValidityPeriod>` element in the `<APIM_HOME>/repository/conf/identity.xml` file.

The Token API takes the following parameters to generate the access token:

- Grant Type
- Username
- Password
- Scope

To generate a new access token, you issue a Token API call with the above parameters where `grant_type=pass word`. The Token API then returns two tokens: an access token and a refresh token. The access token is saved in a session on the client side (the application itself does not need to manage users and passwords). On the API Gateway side, the access token is validated for each API call. When the token expires, you refresh the token by issuing a token API call with the above parameters where `grant_type=refresh_token` and passing the refresh token as a parameter.

## API resources

An API is made up of one or more resources. Each resource handles a particular type of request and is analogous to a method (function) in a larger API. API resources accept the following optional attributes:

- **verbs**: Specifies the HTTP verbs a particular resource accepts. Allowed values are GET, POST, PUT, OPTIONS, DELETE. You can give multiple values at once.
- **uri-template**: A URI template as defined in <http://tools.ietf.org/html/rfc6570>. E.g., /phoneregister/<phoneNumber>.
- **url-mapping**: A URL mapping defined as per the servlet specification (extension mappings, path mappings, and exact mappings).
- **Throttling tiers**: Limits the number of hits to a resource during a given period of time.
- **Auth-Type**: Specifies the Resource level authentication along the HTTP verbs. Auth-type can be None, Application, or Application User.
  - None: Can access the particular API resource without any access tokens.
  - Application: An application access token is required to access the API resource.
  - Application User: A user access token is required to access the API resource.

## Deep diving into the API Manager

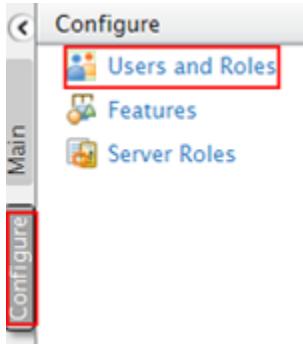
Let's take a look at the typical API management activities in detail:

- Creating users and roles
- Creating an API from scratch
- Adding API documentation
- Adding interactive documentation
- Versioning the API
- Publishing the API
- Subscribing to the API
- Invoking the API
- Monitoring APIs and viewing statistics

### ***Creating users and roles***

In [Users and roles](#), we introduced a set of users who are commonly found in many enterprises. Let's see how you can log in to the Management Console as an admin and create these roles.

1. Log in to the Management Console (<https://<hostname>:9443/carbon>) of the API Manager using admin/admin credentials.
2. Select the **Users and Roles** menu under the **Configure** menu.



3. Click the **Roles** link and then click **Add New Role**.

## Roles

Search

Select Domain

Enter role name pattern (\* for all)

Name	Actions
admin	<input type="button" value="Assign Users"/> <input type="button" value="View Users"/>
Internal/everyone	<input type="button" value="Permissions"/>
Internal/identity	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>
Internal/subscriber	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>
<input type="button" value="Add New Role"/>	
<input type="button" value="Add New Internal Role"/>	

4. Give the role name as `creator` and click **Next**.

## Add Role

### Step 1 : Enter role details

Enter role details

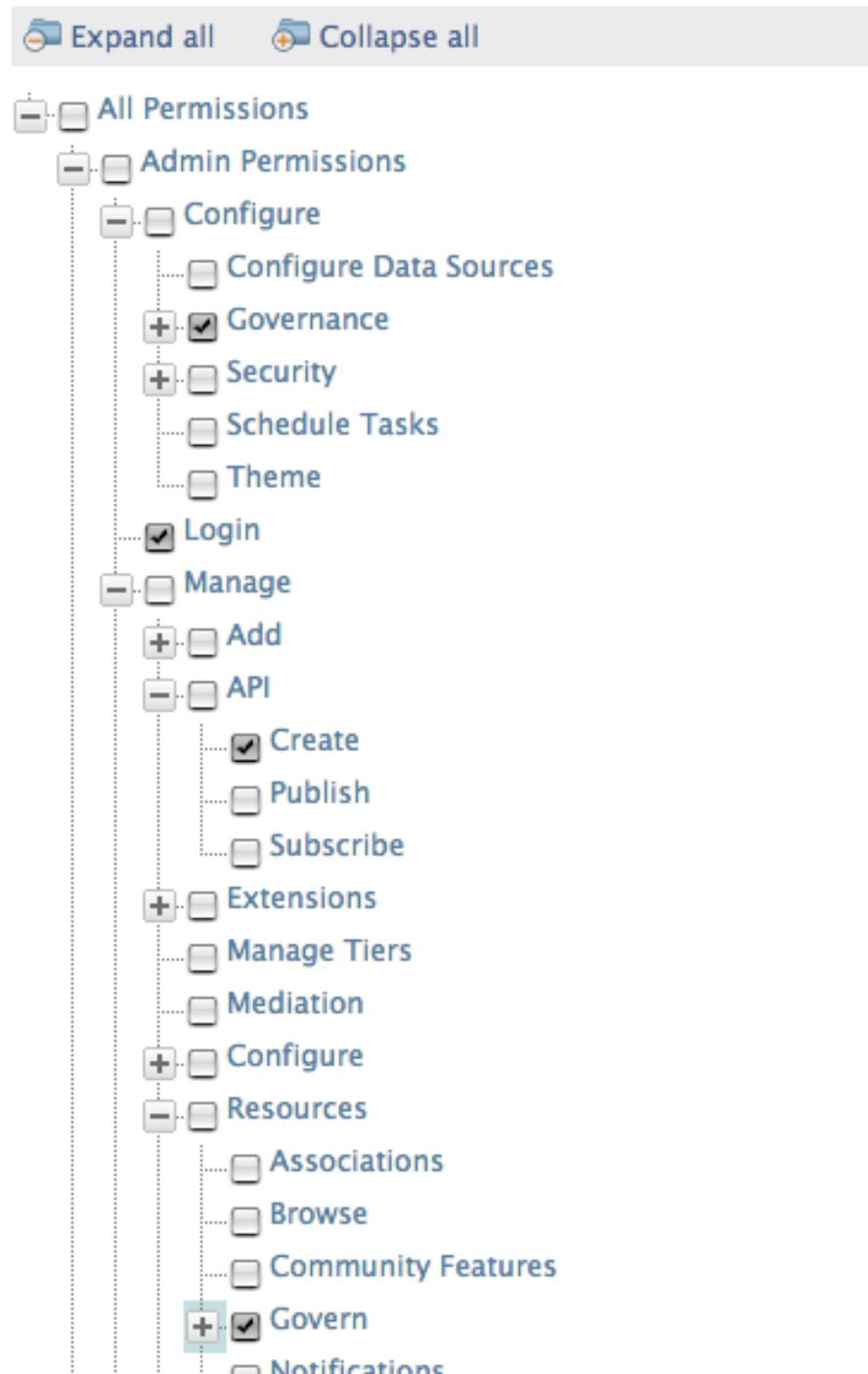
Domain

Role Name\*

5. A list of permissions opens. Select the following and click **Finish**.

- Configure > Governance and all underlying permissions
- Login
- Manage > API > Create
- Manage > Resources > Govern and all underlying permissions

## Step 2 : Select permissions to add to Role



6. Similarly, create the publisher role with the following permissions.
  - Login
  - Manage > API > Publish
7. Note that the API Manager comes with the `subscriber` role available by default. It has the following permissions:
  - Login
  - Manage > API > Subscribe
8. Note that the roles you added (creator, internal/subscriber, and publisher) are now displayed under **Roles**.

## Roles

Search

Select Domain: ALL-USER-STORE-DOMAINS

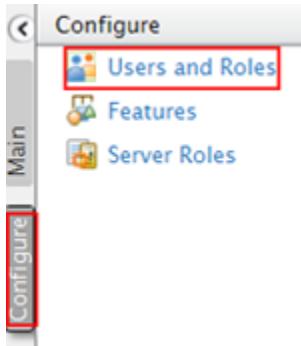
Enter role name pattern (\* for all): \*

Name	Actions
admin	Assign Users  View Users
creator	Rename  Permissions  Assign Users  View Users  Delete
Internal/everyone	Permissions
Internal/identity	Rename  Permissions  Assign Users  View Users  Delete
Internal/subscriber	Rename  Permissions  Assign Users  View Users  Delete
publisher	Rename  Permissions  Assign Users  View Users  Delete

+ Add New Role  
 + Add New Internal Role

Let's create users for each of the roles.

- Click the **Users and Roles** menu under the **Configure** menu again.



- Click the **Users** link and then click **Add New User**.

## Users

Search

Select Domain: ALL-USER-STORE-DOMAINS

Enter user name pattern (\* for all): \*

Name	Actions
admin	Change Password  Assign Roles  View Roles  User Profile
User1	Change Password  Assign Roles  View Roles  Delete  User Profile

+ Add New User  
 Bulk Import Users

- Give the username/password and click **Next**. For example, let's create a new user by the name apipublisher.

**Add User**

**Step 1 : Enter user name**

Enter user name

Domain	PRIMARY
User Name*	apipublisher
Password*	*****
Password Repeat*	*****

**Next >** **Finish** **Cancel**

12. Select the role you want to assign to the user (e.g., publisher) and click **Finish**.

**Add User**

**Step 2 : Select roles of the user**

Enter role name pattern (\* for all) \*

**Users of Role**

Select all on this page | Unselect all on this page

<input type="checkbox"/> admin
<input type="checkbox"/> creator
<input checked="" type="checkbox"/> publisher
<input checked="" type="checkbox"/> Internal/everyone
<input type="checkbox"/> Internal/subscriber
<input type="checkbox"/> Internal/identity

**Finish** **Cancel**

13. Similarly, create a new user by the name `apicreator` and assign the creator role.

#### ***Creating an API from scratch***

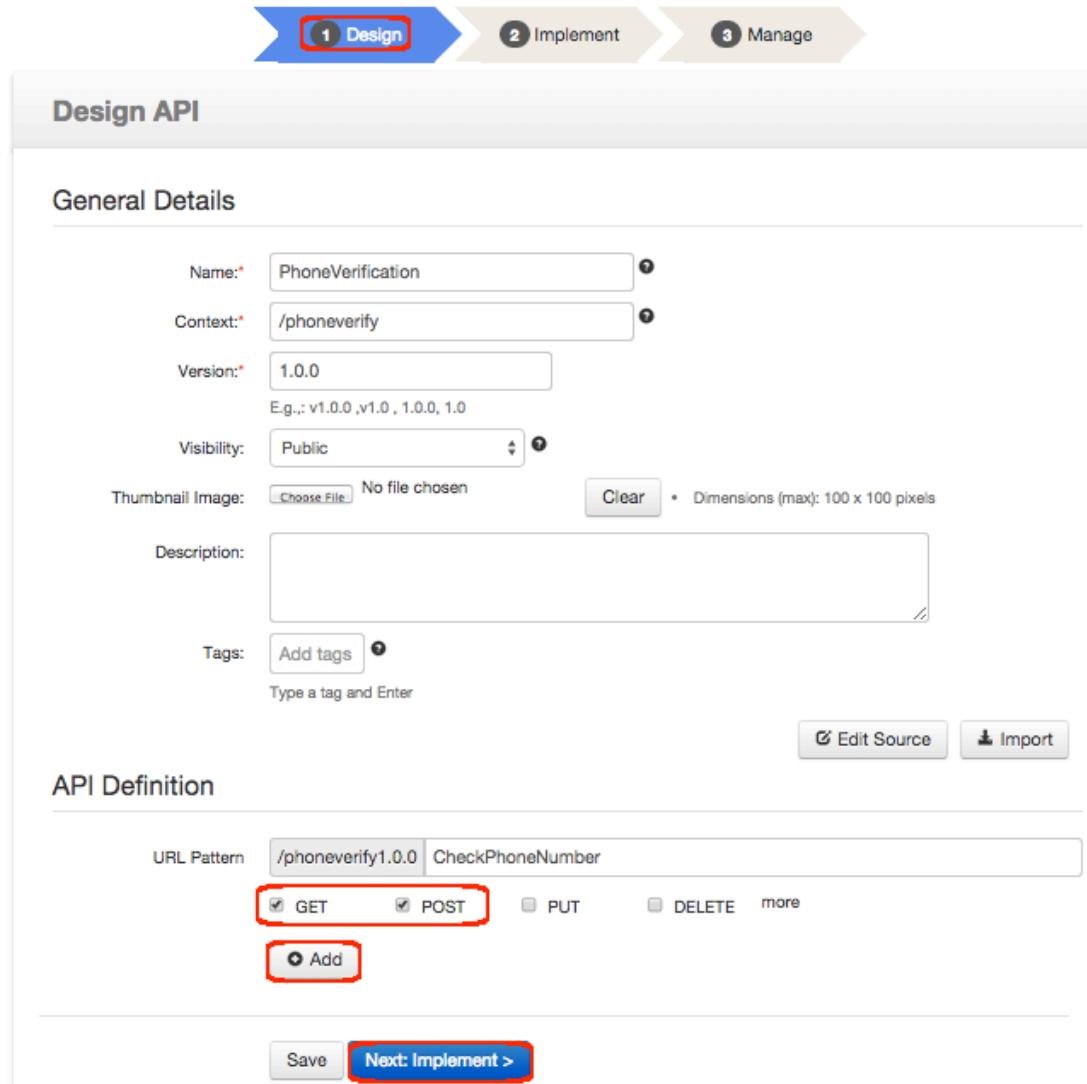
Let's create an API from scratch.

1. Log in to the API Publisher (<https://<hostname>:9443/publisher>) as `apicreator`.
2. Select the option to design a new API and click **Start Creating**.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with a dark header 'APIs' and several menu items: 'Browse', 'Add' (which is highlighted with a red box), 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions' (which is also highlighted with a red box), and 'Tier Permissions'. The main area has a title 'APIs / Add New API' and a heading 'Lets get started!'. It lists three options: 'I have an Existing API', 'I have a SOAP Endpoint', and 'Design new API' (which is selected and highlighted with a blue box). Below each option is a brief description. At the bottom right of the main area is a blue button labeled 'Start Creating'.

3. Give the information in the table below and click **Implement** to move on to the next page.

Field	Sample value
Name	PhoneVerification
Context	/phoneverify
Version	1.0.0
Visibility	Public
API Definition	<ul style="list-style-type: none"> <li>• URL pattern: CheckPhoneNumber</li> <li>• Request types: GET, POST</li> </ul>



The screenshot shows the 'Design API' step in the WSO2 API Manager. The 'General Details' section is active, indicated by a red box around the '1 Design' button in the top navigation bar. The form fields include:

- Name: PhoneVerification
- Context: /phoneverify
- Version: 1.0.0
- Visibility: Public
- Thumbnail Image: No file chosen (with 'Choose File' and 'Clear' buttons)
- Description: (empty text area)
- Tags: Add tags (button) - Type a tag and Enter

Buttons at the bottom right include 'Edit Source' and 'Import'.

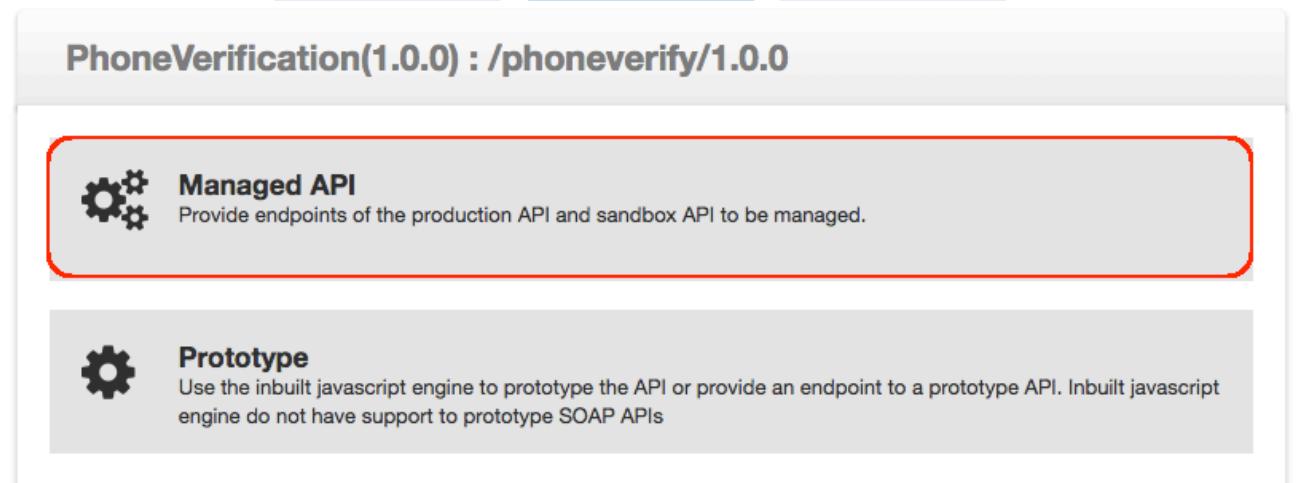
4. Select the Managed API option.



The 'API Definition' section is shown with the following configuration:

- URL Pattern: /phoneverify1.0.0 CheckPhoneNumber
- Method Selection: GET (checked), POST (checked), PUT (unchecked), DELETE (unchecked), more (link)
- Add (button)

Buttons at the bottom left include 'Save' and 'Next: Implement >' (highlighted with a red box).



The screenshot shows the 'PhoneVerification(1.0.0) : /phoneverify/1.0.0' page. It contains two main sections:

- Managed API**: Provides endpoints of the production API and sandbox API to be managed. This section is highlighted with a red border.
- Prototype**: Use the inbuilt javascript engine to prototype the API or provide an endpoint to a prototype API. Inbuilt javascript engine do not have support to prototype SOAP APIs.

5. Give the following information in the **Implement** tab that opens and click **Manage** once you are done.

Field	Sample value

Endpoint type	HTTP
Production endpoint	<p>In this guide, we work with a service exposed by the Cdyne services provider. We use their phone validation service, which has SOAP and REST interfaces. Endpoint is <a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>.</p> <p>This sample service has two operations: CheckPhoneNumber and CheckPhoneNumbers. Let's use CheckPhoneNumber here.</p>

- Click **Manage** to go to the Manage tab and provide the following information. Leave default values for the rest of the parameters in the UI.

Field	Value	Description
Tier Availability	<Select all available tiers>	The API can be available at different levels of service. They allow you to limit the number of successful hits to an API during a given period of time.



## Manage API: PhoneVerification : /phoneverify/1.0.0/1.0.0

### Configurations

Make this default version  ⓘ No default version defined for the current API

Tier Availability: \* **4 selected** ⓘ

Transports: \*  HTTPS  HTTP ⓘ

Sequences:  Check to select a custom sequence to be executed in the message flow

Response Caching: **Disabled** ⓘ

### Gateway Environments ›

### Business Information ›

### Resources

- Once you are done, click **Save**.

#### **Adding API documentation**

- After saving the API, click its thumbnail in the API Publisher to open it.
- Click on the API's **Docs** tab and click the **Add New Document** link.

## PhoneVerification - 1.0.0 ⓘ Edit

**Overview** **Versions** **Docs** **Users**

**Add New Document**

Name	Type	Modified On	Actions
No documentation associated with the API			

- The document options appear. Note that you can create documentation inline, via a URL, or as a file. For inline documentation, you can edit the content directly from the API publisher interface. You get several documents types:
  - How To
  - Samples and SDK
  - Public forum / Support forum (external link only)
  - API message formats
  - Other
- Create a 'How To' named **PhoneVerification**, specifying in-line content as the source and optionally

entering a summary. When you have finished, click **Add Document**.

## PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[+ Add New Document](#)

Name\*  
PhoneVerification

Summary  
Check the validity of a phone number

Type  
 How To  
 Samples & SDK  
 Public Forum  
 Support Forum  
 Other (specify)

Source  
 In-line [?](#)  
 URL [?](#)  
 File [?](#)

[Add Document](#) [Cancel](#)

5. Once the document is added, click **Edit Content** associated with it to open an embedded editor.

## PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[+ Add New Document](#)

Name	Type	Modified On	Actions
PhoneVerification	How To	5/19/2015, 2:27:47 PM	<a href="#">Edit Content</a> <a href="#">Update</a> <a href="#">Delete</a>

6. Enter your API's documentation.

## PhoneVerification

Determine whether a phone number is wireless or a landline - in real time. Even if the phone number has been ported between service providers, Phone Verification will return the latest information.

Phone Verification validates the ten digits of a U.S. telephone number and returns carrier information as well as the Location Routing Number (LRN) for telephone numbers administered by the North American Numbering Plan Administration. LRN is a unique 10-digit number that represents a telephone switch through which multiple phone numbers are routed. The LRN enables Local Number Portability (LNP), which allows phone numbers to be ported to different carriers.

The benefits of using the Phone Verification API include:

- Determine if a phone number is valid
- Remove dashes, parenthesis and spaces
- Receive latest SMTP email string data
- Define time zone to restrict calling times
- Validate wireless vs landline data for compliance

## ***Adding interactive documentation***

WSO2 API Manager has an integrated [Swagger UI](#), which is part of the [Swagger project](#).

Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe a services in the same way that interfaces describe lower-level programming code.

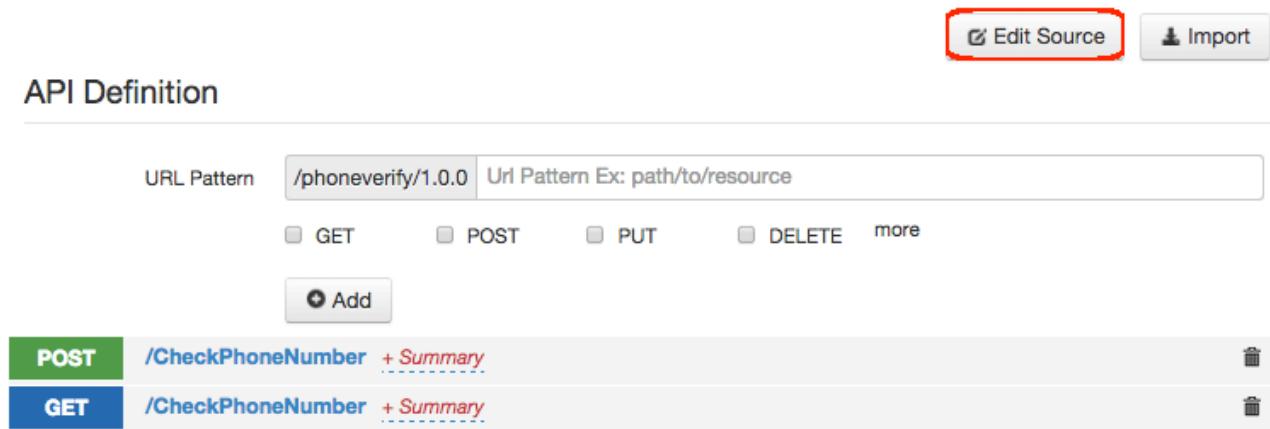
The [Swagger UI](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generates documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation, and more discoverability. The Swagger UI has JSON code, and its UI facilitates easier code indentation, provides keyword highlighting, and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI.

Also, see the [Swagger 2.0 specification](#).

1. Open the API Publisher (<https://<hostname>:9443/publisher>) and log in as **apicreator**.
  2. Click the PhoneVerification API to open it, and then click the **Edit** link right next to the API's name. This opens the API in its edit mode.

## PhoneVerification - 1.0.0

3. Click the **Edit Source** button near the **Resources** section.



The screenshot shows the 'API Definition' interface. At the top right, there are two buttons: 'Edit Source' (highlighted with a red box) and 'Import'. Below these are fields for 'URL Pattern' (set to '/phoneverify/1.0.0') and 'Method' (checkboxes for GET, POST, PUT, DELETE, more). An 'Add' button is present. A table lists API endpoints: a POST method for '/CheckPhoneNumber' and a GET method for the same endpoint. Both rows have '+ Summary' links and trash can icons.

4. The JSON code of the API opens in a separate page. Expand its GET method, add the following parameters and click **Save**.

```
parameters:
  - name: PhoneNumber
    paramType: query
    required: true
    type: string
    description: Give the phone number to be validated
    in: query
  - name: LicenseKey
    paramType: query
    required: true
    type: string
    description: Give the license key as 0 for testing purpose
    in: query
```

Save Close

File - Preferences - Help -

```

1 - paths:
2 -   /CheckPhoneNumber:
3 -     post:
4 -       x-auth-type: "Application & Application User"
5 -       x-throttling-tier: Unlimited
6 -       responses:
7 -         "200": {}
8 -     get:
9 -       x-auth-type: "Application & Application User"
10 -      x-throttling-tier: Unlimited
11 -      responses:
12 -        "200": {}
13 -    parameters:
14 -      - name: PhoneNumber
15 -        paramType: query
16 -        required: true
17 -        type: string
18 -        description: Give the phone number to be validated
19 -        in: query
20 -      - name: LicenseKey
21 -        paramType: query
22 -        required: true
23 -        type: string
24 -        description: Give the license key as 0 for testing purpose
25 -        in: query
26 -    swagger: "2.0"
27 -  info:
28 -    title: PhoneVerification
29 -    version: 1.0.0

```

5. Back in the API Publisher, note that the changes you did appear in the API Console's UI. You can add more parameters and edit the summary/descriptions using the API Publisher UI as well. Once done, click **Save**.

### API Definition

URL Pattern  Url Pattern Ex: path/to/resource

GET     POST     PUT     DELETE    more

**Add**

POST	/CheckPhoneNumber	+ Summary	trash
<b>GET</b>	/CheckPhoneNumber	+ Summary	trash

Description :  
+ Add Implementation Notes

Response Content Type : application/json

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required
PhoneNumber	Give the phone number to be validated	query	string	True
LicenseKey	Give the license key as 0 for testing purpose	query	string	True

Parameter Name  **Add Parameter**

**Save** **Next: Implement >**

### Versioning the API

Let's create a new version of this API.

1. Log in to the API Publisher as **apicreator** if you are not logged in already.
2. Click the PhoneVerification API, and then the **Create New Version** button that appears in its **Overview** tab.

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with options like 'APIs', 'Browse', 'Add', 'All Statistics', and 'My APIs'. Under 'My APIs', there are sections for 'Subscriptions' and 'Statistics'. The main area displays the 'PhoneVerification - 1.0.0' API details. The 'Overview' tab is selected, highlighted with a red box. Below it, there are tabs for 'Versions', 'Docs', and 'Users'. To the right of the tabs, there is a summary card with icons for users (0), created (1.0.0), and docs. A large table provides detailed information about the API, including visibility, context, production URL, date last updated, tier availability, default API version, and published environments. At the bottom of the API details section, there is a 'Create New Version' button, also highlighted with a red box. Below the API details, there is a modal dialog titled 'To Version' with an input field containing '2.0.0' and a note 'E.g., v1.0.1'. It also includes a 'Default Version' checkbox and 'Done' and 'Cancel' buttons.

Visibility	Public
Context	/phoneverify/1.0.0
Production URL	http://ws.cdyne.com/phoneverify/phon...
Date Last Updated	5/19/2015, 3:19:29 PM
Tier Availability	Bronze,Unlimited,Gold,Silver
Default API Version	None
Published Environments	Production and Sandbox

3. Give a new version number (e.g., 2.0.0) and click **Done**.

This is a modal dialog titled 'To Version'. It contains an input field with the value '2.0.0' and a note 'E.g., v1.0.1'. Below the input field is a 'Default Version' checkbox followed by the text 'No default version defined for the current API'. At the bottom of the dialog are two buttons: 'Done' (highlighted with a red box) and 'Cancel'.

4. Note that the new version of the API is created in the API Publisher.

#### **Publishing the API**

1. Log in to the API Publisher as the **apipublisher** user that you created earlier in this guide, and click the PhoneVerification API's version 2.0.0.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', and 'Statistics'. The main area is titled 'All APIs' and contains a search bar with 'Filter APIs' and a 'Search' button. Below the search bar are two API entries, each with a gear icon. The first entry is 'PhoneVerification 1.0.0' with '0 Users' and 'CREATED'. The second entry is 'PhoneVerification 2.0.0', which is circled in red, with '0 Users' and 'CREATED'.

2. The API opens. Go to its **Lifecycle** tab, select the state as PUBLISHED from the drop-down list, and click **Update**.

The screenshot shows the 'PhoneVerification - 2.0.0' API details page. At the top, there are tabs for 'Overview', 'Lifecycle' (which is highlighted with a red box), 'Versions', 'Docs', and 'Users'. Below the tabs, there's a form with a 'State:' dropdown set to 'PUBLISHED'. Underneath the dropdown are three checkboxes: 'Propagate Changes to API Gateway' (checked), 'Deprecate Old Versions' (unchecked), and 'Require Re-Subscription' (checked). At the bottom of the form are 'Update' and 'Reset' buttons, with 'Update' also highlighted with a red box.

mean the following:

- **Propagate Changes to API Gateway:** Used to define an API proxy in the API Gateway runtime component, allowing the API to be exposed to the consumers via the API Gateway. If this option is left unselected, the API metadata will not change, and you will have to manually configure the API Gateway according to the information published in the API Store.
- **Deprecate Old Versions:** If selected, any prior versions of the API that are published will be set to the DEPRECATED state automatically.
- **Require Re-Subscription:** Invalidates current user subscriptions, forcing users to subscribe again.

3. Go to the API Store (<https://<hostname>:9443/store>) using your browser and note that the PhoneVe

rification 2.0.0 API is visible under the APIs menu.

The screenshot shows the WSO2 API Manager dashboard. At the top, there's a navigation bar with links for 'API STORE', 'Prototyped APIs', 'My Applications', 'My Subscriptions', 'Forum', and 'Statistics'. The 'APIs' link is highlighted with a red box. Below the navigation bar is a search bar labeled 'Search API' with a magnifying glass icon. On the left, there's a 'Recently Added' section showing a card for 'PhoneVerification-2.0.0' with a gear icon, a five-star rating, and the version '2.0.0'. To the right, there's a larger 'APIs' section with a gear icon, the name 'PhoneVerification', and the version '2.0.0'.

### **Subscribing to the API**

1. Go to the API Store (<https://<hostname>:9443/store>) and create an account using the **Sign-up** link.

The screenshot shows the WSO2 API Manager dashboard after signing up. The 'Sign-up' link in the top navigation bar is highlighted with a red box. The rest of the interface is similar to the previous screenshot, showing the 'APIs' menu item highlighted and the 'PhoneVerification' API listed in the 'APIs' section.

2. After signing up, log in to the API Store and click the PhoneVerification 2.0.0 API that you published earlier.
3. Note that you can now see the subscription options on the right-hand side of the UI. Select the default application, select the Bronze tier and click **Subscribe**.

The screenshot shows the WSO2 API Manager interface. At the top, there's a navigation bar with tabs for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Themes, and TestUser. Below the navigation bar is a search bar labeled "Search API". The main content area displays the "PhoneVerification - 2.0.0" API. On the left, there's a sidebar with a link to "More APIs from 'admin'" and a thumbnail for "WeatherAPI-1.0.0". The main panel shows the API's rating (5 stars), version (2.0.0), status (PUBLISHED), and last updated (19/May/2015 15:37:07 PM IST). To the right, there's a section for "Applications" with a dropdown set to "DefaultApplication" and a "Tiers" dropdown set to "Bronze". A red box highlights the "Subscribe" button at the bottom right of this section.

4. Once the subscription is successful, choose to go to the **My Subscriptions** page.
5. In the **My Subscriptions** page, click the **Generate Keys** button to generate an access token that you need to invoke the API.

The screenshot shows the "Subscriptions" page. At the top, there's a navigation bar with tabs for APIs, Prototyped APIs, My Applications, My Subscriptions (which is highlighted with a red box), Forum, Statistics, Themes, and TestUser. Below the navigation bar is a search bar labeled "Search API". The main content area is titled "Applications With Subscriptions" and shows a dropdown menu set to "DefaultApplication". A checkbox labeled "Show Keys" is checked. Below this, there are two sections: "Keys - Production" and "Keys - Sandbox". The "Keys - Production" section contains a message stating "Production keys are not yet generated for this application." and a "Generate keys" button (which is highlighted with a red box). To the right of this, there's a "Allowed Domains" section with a dropdown set to "ALL" and a note about domain restrictions. The "Keys - Sandbox" section contains a tip about setting token validity periods.

**Tip :** You can set a token validity period in the given text box. By default, it is set to one hour. If you set a minus value (e.g., -1), the token will never expire.

You are now successfully subscribed to an API. Let's invoke it.

#### Invoking the API

1. Click the **APIs** menu in the API Store and then click on the API that you want to invoke. When the API opens, go to its API Console tab.

**WSO2 API Manager**

The screenshot shows the WSO2 API Manager interface. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Themes, and admin. A search bar is also present. Below the navigation bar, the title "PhoneVerification - 1.0.0" is displayed. On the left, there is a profile icon for "admin". The main content area displays the "PhoneVerification" API details, including its icon (three gears), rating (N/A), version (1.0.0), status (PUBLISHED), and last updated (08/Dec/2014 16:50:11 PM IST). To the right, there are dropdown menus for "Applications" (Select Application...) and "Tiers" (Unlimited), both with "Allows unlimited requests" noted below them. A "Subscribe" button is also visible. Below the details, there are tabs for Overview, Documentation, API Console (which is selected and highlighted with a red box), and Throttling Info. Under the API Console tab, there are dropdowns for "Try" (DefaultApplication) and "On" (Production) environments, and a "Envioromant." dropdown. A "Set Request Header" field contains "Authorization : Bearer b690ca26e6375b44b5bde78a44fbef2". The "PhoneVerification" resource is listed with two operations: "GET /CheckPhoneNumber" and "POST /CheckPhoneNumber". The "GET" method is selected. The "Parameters" section for the "POST" method shows two required parameters: "PhoneNumber" and "LicenseKey", each with a description and a red box around it. The "PhoneNumber" parameter is described as "Give the phone number to be validated" and the "LicenseKey" parameter is described as "Give the license key as 0 for testing purpose".

PhoneVerification - 1.0.0

admin

**Rating:** Your rating: N/A

**Version:** 1.0.0

**Status:** PUBLISHED

**Updated:** 08/Dec/2014 16:50:11 PM IST

**Applications:** Select Application...

**Tiers:** Unlimited

Allows unlimited requests

**Subscribe**

**Overview** **Documentation** **API Console** **Throttling Info**

Try DefaultApplication On Production Envioromant.

Set Request Header Authorization : Bearer b690ca26e6375b44b5bde78a44fbef2

**PhoneVerification**

Swagger Resource Listing ( /api-docs )

checkphonenumbers :

Show/Hide List Operations Expand Operations Raw

**GET** /CheckPhoneNumber

**POST** /CheckPhoneNumber

**GET** /CheckPhoneNumber

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	(required)	Give the phone number to be validated	query	string
LicenseKey	(required)	Give the license key as 0 for testing purpose	query	string

2. Expand the GET method of the resource CheckPhoneNumber. Note the parameters that you added when creating the interactive documentation now appear with their descriptions so that as a subscriber, you know how to invoke this API.

3. Give sample values for the PhoneNumber and LicenseKey and click Try it out to invoke the API.

**GET** /CheckPhoneNumber

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			

**Try it out!**

**POST** /CheckPhoneNumber

4. Note the response for the API invocation. Because we used a valid phone number in this example, the response is valid.

### Response Body

```

<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query"> <Company>Toll Free</Company>
  <Valid>true</Valid>
    <Use>Assigned to a code holder for normal use.</Use>
    <State>TF</State>
    <RC />
    <OCN />
    <OriginalNumber>18006785432</OriginalNumber>
    <CleanNumber>8006785432</CleanNumber>
    <SwitchName />
    <SwitchType />
    <Country>United States</Country>
    <CLLI />
    <PrefixType>Landline</PrefixType>
    <LATA />
    <sms>Landline</sms>
    <Email />
    <AssignDate />
    <TelecomCity />
    <TelecomCounty />

```

### Response Code

200

### Response Headers

```

Pragma: no-cache
Content-Type: text/xml; charset=utf-8
Cache-Control: no-cache
Expires: -1

```

You have invoked an API using the API Console.

#### **Monitoring APIs and viewing statistics**

Both the API publisher and store provide several statistical dashboards. Some of them are as follows:

- Number of subscriptions per API (across all versions of an API)
- Number of API calls being made per API (across all versions of an API)
- The subscribers who did the last 10 API invocations and the APIs/versions they invoked
- Usage of an API and from which resource path (per API version)
- Number of times a user has accessed an API
- The number of API invocations that failed to reach the endpoint per API per user
- API usage per application
- Users who make the most API invocations, per application
- API usage from resource path, per application

The steps below explain how to configure [WSO2 Business Activity Monitor \(BAM\) 2.5.0](#) with the API Manager. The statistics in these dashboards are based on data from the BAM. The steps below explain how to configure WSO2 BAM 2.5.0 with the API Manager.

 If you are on **Windows**, note the following:

- If you installed the JDK in Program Files in the Windows environment, avoid the space by using PROGRA~1 when specifying environment variables for JAVA\_HOME and PATH. Otherwise, the server throws an exception.
- Install Cygwin (<http://www.cygwin.com>). WSO2 BAM depends on Apache Hadoop, which requires Cygwin in order to run on Windows. Install at least the basic net (OpenSSH,tcp\_wrapper packages) and security-related Cygwin packages. After Cygwin installation, update the PATH variable with C:/cygwin/bin. If you already have WSO2 BAM running, you must restart it now.

Let's do the configurations first.

1. Apply an offset of 3 to the default BAM port by editing the <BAM\_HOME>/repository/conf/carbon.xml file. This makes the BAM server run on port 9446 instead of the default port 9443, and avoids port conflicts when multiple WSO2 products run on the same host.

```
<Offset>3</Offset>
```

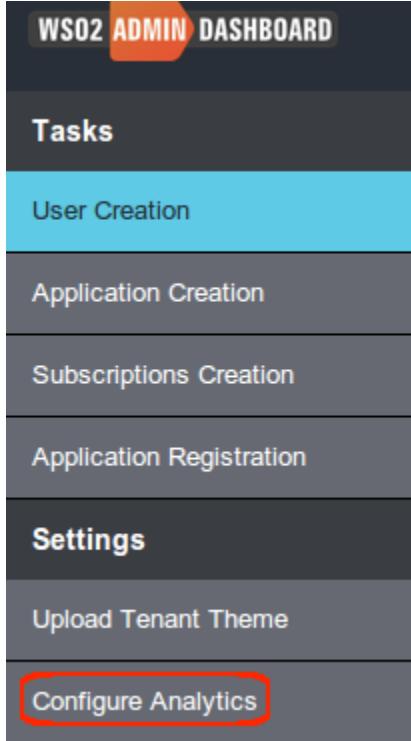
2. Download MySQL from <https://www.mysql.com/> and install it in your server.
3. Go to the command-line and issue the following commands to connect to the MySQL server and create a database (e.g., TestStatsDB). This database is used to save the statistical data collected by the BAM. You do not need to create any tables in it.

```
mysql -u <username> -p <password> -h <host_name or IP>;
CREATE DATABASE TestStatsDB;
```

4. Save the MySQL connector JAR inside both the <APIM\_HOME>/repository/components/lib and <BAM\_HOME>/repository/components/lib folders.
5. Give the datasource definition under the <datasource> element in the <BAM\_HOME>/repository/conf/datasources/master-datasources.xml file. For example,

```
<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for getting statistics to API Manager</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/TestStatsDB</url>
            <username>db_username</username>
            <password>db_password</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

6. Start the BAM server by running either of the following commands in the command line:
  - On Windows: <PRODUCT\_HOME>\bin\wso2server.bat --run
  - On Linux/Solaris/Mac OS: sh <PRODUCT\_HOME>/bin/wso2server.sh
7. Start the API Manager and log in to its Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) with **admin/admin** credentials.
8. Click the **Configure Analytics** menu.



9. Select the **Enable** check box to enable statistical data publishing and add the following:
  - Add a URL group as `tcp://<BAM server IP>:7614` and click **Add URL Group**.
  - Fill the details under **Statistics Summary Database** according to the information you added to the `master-datasources.xml` file in step 4.

## Configure Analytics

Enable API usage publishing and Statistics aggregation

Enable

### Event Receiver Configurations

URL Group: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/>	<input type="text"/> admin	<input type="text"/> .....
<input type="button"/> Add URL Group		

Event Receiver Group	Username	Actions
<input type="text"/> (tcp://localhost:7614)	admin	<input type="button"/>

### Data Analyzer Configurations

URL: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/> https://localhost:9446	<input type="text"/> admin	<input type="text"/> .....
<input type="button"/>		

### Statistics Summary Datasource

URL: <sup>*</sup>	JDBC Driver Class: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/> jdbc:mysql://localhost:3306/TestStatsDB	<input type="text"/> com.mysql.jdbc.Driver	<input type="text"/> root	<input type="text"/> .....
<input type="button"/> Show More Options			

Save

10. Click **Save**. The BAM deploys the Analytics toolbox, which describes the information collected, how to analyze the data, and the location of the database where the analyzed data is stored.
11. Invoke several APIs to generate some statistical data and wait a few seconds.
12. Connect to the API Publisher as a creator or publisher and click the statistical dashboards available under the **All Statistics** and **Statistics** menus. For example,

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following navigation paths:

- APIs
- Browse
- Add
- All Statistics (highlighted with a red box)
- API Subscriptions
- API Usage
- API Response Times
- API Last Access Times (highlighted with a red box)
- API Usage by Resource Path
- API Usage by Destination
- API Usage Comparison
- Faulty Invocations
- My APIs
- Subscriptions
- Statistics (highlighted with a red box)
- Tier Permissions
- Tier Permissions

The main content area displays a dashboard titled "API Last Access Times (Across All Versions)". It includes a search bar and a table with the following data:

API	VERSION	SUBSCRIBER	ACCESS TIME (GMT+05:30)
[redacted]	1.0.0	[redacted]	5/18/2015, 2:12:00 PM
[redacted]	1.0.0	[redacted]	5/8/2015, 3:37:00 PM

Showing 1 to 2 of 2 entries

The **All Statistics** menu is available for both API creators and publishers. It shows statistics of all APIs. The **Statistics** menu is available for API creators to see statistics of only the APIs created by them.

This concludes the API Manager quick start. You have set up the API Manager and gone through the basic use

cases of the product. For more advanced use cases, please see the [User Guide](#) and the [Admin Guide](#) of the API Manager documentation.

## Downloading the Product

Follow the instructions below to download the binary distribution of the API Manager.

The binary distribution contains the binary files for both MS Windows, and Linux-based operating systems. It is recommended for most users. You can also download, and [build the source code](#).

1. In your Web browser, go to <http://wso2.com/products/api-manager>.
2. Click the **Download** button in the upper right-hand corner of the page to download the **latest** version. To download an older version, click the **Previous Releases** link and then select the version that you want.
3. Enter the required details in the form, and click **Download**.

Next, go to [Installation Prerequisites](#) for instructions on installing the necessary supporting applications.

## Installation Prerequisites

Prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. Verify that the computer has the supported operating system and development platforms before starting the installation.

### System requirements

<b>Memory</b>	<ul style="list-style-type: none"> <li>• ~ 2 GB minimum</li> <li>• ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently.</li> </ul>
<b>Disk</b>	<ul style="list-style-type: none"> <li>• ~ 500 MB for a fresh installation pack, excluding space allocated for log files and databases.</li> </ul>

### Environment compatibility

- All WSO2 Carbon-based products are Java applications that can be run on **any platform that is Oracle JDK 1.6.\*/1.7.\* compliant. JDK 1.8 is not supported yet**. Also, we **do not recommend or support OpenJDK**.
- All WSO2 Carbon-based products are generally compatible with most common DBMSs. The embedded H2 database is suitable for development, testing, and some production environments. For most enterprise production environments, however, we recommend you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. For more information, see [Working with Databases](#). Additionally, we do not recommend the H2 database as a user store.
- It is **not recommended to use Apache DS** in a production environment due to scalability issues. Instead, use an LDAP like OpenLDAP for user management.
- For environments that WSO2 products are tested with, see [Compatibility of WSO2 Products](#).
- If you have difficulty in setting up any WSO2 product in a specific platform or database, please [contact us](#).

### Required applications

The following applications are required for running the API Manager and its samples or for building from the source code. Mandatory installs are marked with \*.

Application	Purpose	Version	Download Links
-------------	---------	---------	----------------

<b>Oracle Java S E Development Kit (JDK)*</b>	<ul style="list-style-type: none"> <li>• To launch the product as each product is a Java application.</li> <li>• To <a href="#">build the product from the source distribution</a> (both JDK and Apache Maven are required).</li> <li>• To run Apache Ant.</li> </ul>	<p>1.6.27 or later / 1.7.*</p> <ul style="list-style-type: none"> <li>• If you are using <b>JDK 1.6</b>, you might need to replace the Java Cryptography Extension (JCE) policy files in your JDK with the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files. This will avoid "illegal key size" errors when you try to invoke a secured Web service.</li> <li>• <b>To build the product</b> from the source distribution, you must use JDK 1.6 instead of JDK 1.7.</li> <li>• <b>Oracle and IBM JRE 1.7 are also supported</b> when running (not building) WSO2 products.</li> <li>• We <b>do not recommend OpenJDK</b>.</li> </ul>	<a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a>
---	---	---	---

<b>Apache ActiveMQ JMS Provider</b>	<ul style="list-style-type: none"> <li>To enable the product's JMS transport and try out JMS samples. The ActiveMQ client libraries must be installed in the product's classpath before you can enable the JMS transport.</li> </ul>	5.5.0 or later  If you use any other JMS provider (e.g., Apache Qpid), install any necessary libraries and/or components.	<a href="http://activemq.apache.org">http://activemq.apache.org</a>
<b>Apache Ant</b>	<ul style="list-style-type: none"> <li>To compile and run the product samples in &lt;APIM_HOME&gt;/samples .</li> </ul>	1.7.0 or later	<a href="http://ant.apache.org">http://ant.apache.org</a>
<b>SVN Client</b>	<ul style="list-style-type: none"> <li>To check out the code to <a href="#">build the product from the source distribution</a>. If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do <b>not</b> need to install SVN.</li> </ul>		<ul style="list-style-type: none"> <li>Linux - <a href="http://subversion.apache.org/packages.html">http://subversion.apache.org/packages.html</a></li> <li>Windows - <a href="http://tortoisesvn.net/downloads.html">http://tortoisesvn.net/downloads.html</a></li> </ul>
<b>Apache Maven</b>	<ul style="list-style-type: none"> <li>To <a href="#">build the product from the source distribution</a> (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do <b>not</b> need to install Maven.</li> </ul>	3.0.*	<a href="http://maven.apache.org">http://maven.apache.org</a>
<b>Web Browser</b>	<ul style="list-style-type: none"> <li>To access the <a href="#">Management Console</a>. The Web browser must be JavaScript enabled to take full advantage of the Management console.</li> </ul> <p> On Windows Server 2003, you must not go below the medium security level in Internet Explorer 6.x.</p>		

You are now ready to install. Click one of the following links for instructions:

- [Installing on Linux or OS X](#)
- [Installing on Solaris](#)
- [Installing on Windows](#)
- [Installing as a Linux Service](#)

## Installing the Product

Installing WSO2 is very fast and easy. Before you begin, be sure you have met the installation prerequisites, and then follow the installation instructions for your platform. WSO2 also provides pre-configured packages for automated installation based on Puppet or similar solutions. For information, [contact team WSO2](#).

- [Installing on Linux or OS X](#)
- [Installing on Solaris](#)
- [Installing on Windows](#)
- [Installing as a Linux Service](#)
- [Installing as a Windows Service](#)

### Installing on Linux or OS X

 **Before you begin,** please see our compatibility matrix to find out if this version of the product is fully tested on Linux or OS X.

Follow the instructions below to install API Manager on Linux or Mac OS X.

#### *Installing the required applications*

1. Log in to the command line (Terminal on Mac) either as root or obtain root permissions after logging in via `su` or `sudo` command.
2. [Installation Prerequisites](#) Ensure that your system meets the . Java Development Kit (JDK) is essential to run the product.

#### *Installing the API Manager*

1. [Downloading the Product](#) Download the latest version of the API Manager as described in.
2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_M_HOME>`.

#### *Setting up JAVA\_HOME*

You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the operating system.

1. In your home directory, open the `BASHRC` file (`.bash_profile` file on Mac) using editors such as `vi`, `emacs`, `pico`, or `mcedit`.
2. Assuming you have JDK 1.6.0\_25 in your system, add the following two lines at the bottom of the file, replacing `/usr/java/jdk1.6.0_25` with the actual directory where the JDK is installed.

```
On Linux:  
export JAVA_HOME=/usr/java/jdk1.6.0_25  
export PATH=${JAVA_HOME}/bin:${PATH}
```

```
On OS X:  
export JAVA_HOME=/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
```

3. Save the file.

 If you do not know how to work with text editors in a Linux SSH session, run the following command: `cat >> .bashrc`. Paste the string from the clipboard and press "Ctrl+D."

- To verify that the `JAVA_HOME` variable is set correctly, execute the following command:

```
On Linux:  
echo $JAVA_HOME
```

```
On OS X:  
which java
```

If the above command gives you a path like `/usr/bin/java`, then it is a symbolic link to the real location. To get the real location, run the following:  
`ls -l `which java``

- The system returns the JDK installation path.

#### **Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

- Set the properties from a script:** Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- Set the properties from an external registry:** If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the [secure vault](#).

**i** When using SUSE Linux, it ignores `/etc/resolv.conf` and only looks at the `/etc/hosts` file. This means that the server will throw an exception on startup if you have not specified anything besides `localhost`. To avoid this error, add the following line above `127.0.0.1 localhost` in the `/etc/hosts` file:  
`:<ip_address> <machine_name> localhost`

You are now ready to [run the product](#).

#### **Installing on Solaris**

**⚠ Before you begin**, please see our [compatibility matrix](#) to find out if this version of the product is fully tested on Solaris.

Follow the instructions below to install API Manager on Solaris.

#### **Installing the required applications**

- Establish a SSH connection to the Solaris machine or log in on the text console. You should either log in as root or obtain root permissions after login via `su` or `sudo` command.  
**Installation Prerequisites** Be sure your system meets the . Java Development Kit (JDK) is essential to run the product.

#### **Installing the API Manager**

- Downloading the Product** Download the latest version of the API Manager as described in.
- Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_HOME>`.

#### **Setting up JAVA\_HOME**

You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK)

is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the operating system.

1. In your home directory, open the BASHRC file in your favorite text editor, such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.6.0\_25 in your system, add the following two lines at the bottom of the file, replacing /usr/java/jdk1.6.0\_25 with the actual directory where the JDK is installed.

```
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
```

The file should now look like this:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
export M2_HOME=/opt/apache-maven-3.0.3
export PATH=${M2_HOME}/bin:${PATH}
```

3. Save the file.

If you do not know how to work with text editors in an SSH session, run the following command: cat >> .bashrc

Paste the string from the clipboard and press "Ctrl+D."

4. To verify that the JAVA\_HOME variable is set correctly, execute the following command:

```
echo $JAVA_HOME
```

```
[suncoma@wso2 ~]$ echo $JAVA_HOME
/usr/java/jdk1.6.0_25
[suncoma@wso2 ~]$
```

5. The system returns the JDK installation path.

### **Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script:** Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- **Set the properties from an external registry:** If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the [secure vault](#).

You are now ready to [run the product](#).

### **Installing on Windows**



**Before you begin**, please see our compatibility matrix to find out if this version of the product is fully tested on Windows.

Follow the instructions below to install API Manager on Windows.

#### ***Installing the required applications***

**Installation Prerequisites** Be sure your system meets the. Java Development Kit (JDK) is essential to run the product.

2. Be sure that the `PATH` environment variable is set to "`C:\Windows\System32`", because the `findstr` window's exe is stored in this path.

#### ***Installing the API Manager***

**Downloading the Product** Download the latest version of the API Manager as described in.

2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_HOME>`.

#### ***Setting up JAVA\_HOME***

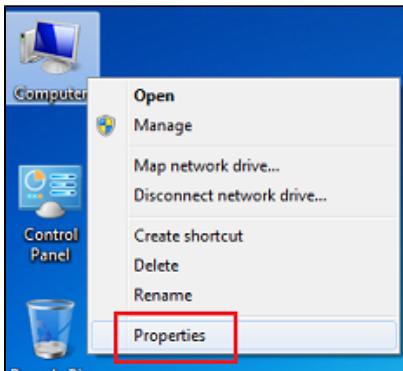
You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:/Program Files/Java`, such as `C:/Program Files/Java/jdk1.6.0_27`. If you have multiple versions installed, choose the latest one, which you can find by sorting by date.

Environment variables are global system variables accessible by all the processes running under the operating system. You can define an environment variable as a system variable, which applies to all users, or as a user variable, which applies only to the user who is currently logged in.

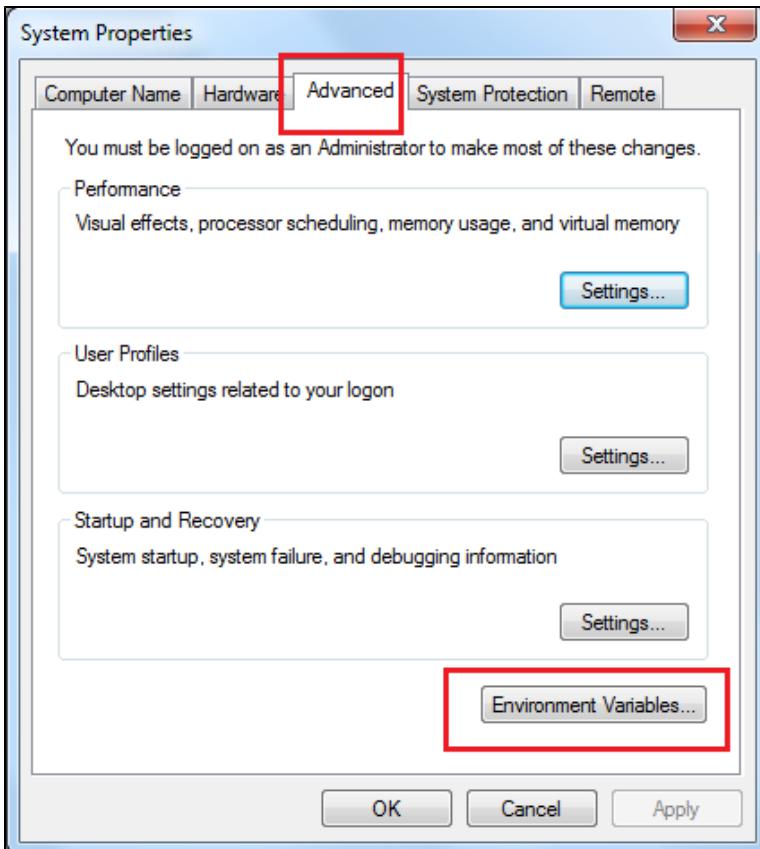
You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, [set it at the command prompt](#).

#### ***Setting up JAVA\_HOME using the system properties***

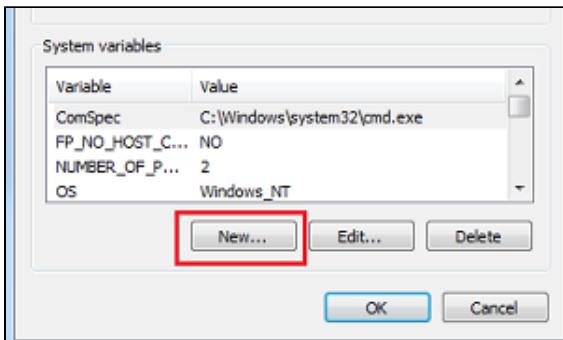
1. Right-click the **My Computer** icon on the desktop and choose **Properties**.



2. In the System Properties window, click the **Advanced** tab, and then click the **Environment Variables** button.



3. Click the New button under **System variables** (for all users) or under **User variables** (just for the user who is currently logged in).



4. Enter the following information:

- In the **Variable name** field, enter: JAVA\_HOME
- In the **Variable value** field, enter the installation path of the Java Development Kit, such as: c:/Program Files/Java/jdk1.6.0\_27

The JAVA\_HOME variable is now set and will apply to any subsequent command prompt windows you open. If you have existing command prompt windows running, you must close and reopen them for the JAVA\_HOME variable to take effect, or manually set the JAVA\_HOME variable in those command prompt windows as described in the next section. To verify that the JAVA\_HOME variable is set correctly, open a command window (from the **Start** menu, click **Run**, and then type **CMD** and click **Enter**) and execute the following command:

```
set JAVA_HOME
```

The system returns the JDK installation path. You are now ready to [run the product](#).

#### **Setting JAVA\_HOME temporarily using the Windows command prompt (CMD)**

You can temporarily set the JAVA\_HOME environment variable within a Windows command prompt window (CMD). This is useful when you have an existing command prompt window running and you do not want to restart it.

1. In the command prompt window, enter the following command where <JDK\_INSTALLATION\_PATH> is the JDK installation directory and press **Enter**.

```
set JAVA_HOME=<JDK_INSTALLATION_PATH>
```

For example: set JAVA\_HOME=c:/Program Files/java/jdk1.6.0\_27

The JAVA\_HOME variable is now set for the current CMD session only.

2. To verify that the JAVA\_HOME variable is set correctly, execute the following command:

```
set JAVA_HOME
```

3. The system returns the JDK installation path.

### **Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script:** Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- **Set the properties from an external registry:** If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the [secure vault](#).

You are now ready to [run the product](#).

## **Installing as a Linux Service**

Follow the sections below to run a WSO2 product as a Linux service:

- [Prerequisites](#)
- [Setting up CARBON\\_HOME](#)
- [Running the product as a Linux service](#)

### **Prerequisites**

Install JDK 1.6.24 or later or 1.7.\* and set up the JAVA\_HOME environment variable.

### **Setting up CARBON\_HOME**

Extract the WSO2 product to a preferred directory in your machine and set the environment variable CARBON\_HOME to the extracted directory location.

### **Running the product as a Linux service**

1. To run the product as a service, create a startup script and add it to the boot sequence. The basic structure of the startup script has three parts (i.e., start, stop and restart) as follows:

```
#!/bin/bash

case "$1" in
start)
    echo "Starting the Service"
;;
stop)
    echo "Stopping the Service"
;;
restart)
    echo "Restarting the Service"
;;
*)
    echo $"Usage: $0 {start|stop|restart}"
exit 1
esac
```

Given below is a sample startup script. <PRODUCT\_HOME> can vary depending on the WSO2 product's directory.

```
#!/bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='<PRODUCT_HOME>/bin/wso2server.sh start > /dev/null &'
restartcmd='<PRODUCT_HOME>/bin/wso2server.sh restart > /dev/null &'
stopcmd='<PRODUCT_HOME>/bin/wso2server.sh stop > /dev/null &'

case "$1" in
start)
    echo "Starting the WSO2 Server ..."
    su -c "${startcmd}" user1
;;
restart)
    echo "Re-starting the WSO2 Server ..."
    su -c "${restartcmd}" user1
;;
stop)
    echo "Stopping the WSO2 Server ..."
    su -c "${stopcmd}" user1
;;
*)
    echo $"Usage: $0 {start|stop|restart}"
exit 1
esac
```

In the above script, the server is started as a user by the name user1 rather than the root user. For example,  
`su -c "${startcmd}" user1`

2. Add the script to /etc/init.d/ directory.

**i** If you want to keep the scripts in a location other than /etc/init.d/ folder, you can add a symbolic link to the script in /etc/init.d/ and keep the actual script in a separate location. Say your script name is prodserver and it is in /opt/WSO2/ folder, then the commands for adding a link to /etc/init.d/ is as follows:

- Make executable: `sudo chmod a+x /opt/WSO2/prodserver`
  - Add a link to `/etc/init.d/`: `sudo ln -snf /opt/WSO2/prodserver /etc/init.d/prodserver`
3. Install the startup script to respective runlevels using the command `update-rc.d`. For example, give the following command for the sample script shown in step1:

```
sudo update-rc.d prodserver defaults
```

The `defaults` option in the above command makes the service to start in runlevels 2,3,4 and 5 and to stop in runlevels 0,1 and 6.

A **runlevel** is a mode of operation in Linux (or any Unix-style operating system). There are several runlevels in a Linux server and each of these runlevels is represented by a single digit integer. Each runlevel designates a different system configuration and allows access to a different combination of processes.

4. You can now start, stop and restart the server using `service <service name> {start|stop|restart}` command. You will be prompted for the password of the user (or root) who was used to start the service.

## Installing as a Windows Service

WSO2 Carbon and any Carbon-based product can be run as a Windows service as described in the following sections:

- Prerequisites
- Setting up the YAJSW wrapper configuration file
- Setting up CARBON\_HOME
- Running the product in console mode
- Working with the WSO2CARBON service

### **Prerequisites**

- Install JDK 1.6.24 or later or 1.7.\* and set up the `JAVA_HOME` environment variable.
- Download and install a service wrapper library to use for running your WSO2 product as a Windows service. WSO2 recommends Yet Another Java Service Wrapper ([YAJSW](#)) version 11.03, and several WSO2 products provide a default `wrapper.conf` file in their `<PRODUCT_HOME>/bin/yajsw/` directory. The instructions below describe how to set up this file.

### **Setting up the YAJSW wrapper configuration file**

The configuration file used for wrapping Java Applications by YAJSW is `wrapper.conf`, which is located in the `<YAJSW_HOME>/conf/` directory and in the `<PRODUCT_HOME>/bin/yajsw/` directory of many WSO2 products. Following is the minimal `wrapper.conf` configuration for running a WSO2 product as a Windows service. Open your `wrapper.conf` file, set its properties as follows, and save it in `<YAJSW_HOME>/conf/` directory.

**i** If you want to set additional properties from an external registry at runtime, store sensitive information like usernames and passwords for connecting to the registry in a properties file and secure it with [secure vault](#).

### **Minimal wrapper.conf configuration**

```
#####
# working directory
#####
wrapper.working.dir=${carbon_home}\
# Java Main class.
```

```

# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
*****wrapper.java.mainclass=*****
# tmp folder
# yajsw creates temporary files named in_.. out_.. err_.. jna..
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
*****wrapper.java.tmp.path = ${jna_tmpdir}*****
# Application main class or native executable
# One of the following properties MUST be defined
*****wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap*****
# Log Level for console output. (See docs for log levels)
wrapper.console.level=INFO
# Log file to use for wrapper output logging.
wrapper.logfile=${wrapper.home}\log\wrapper.log
# Format of output for the log file. (See docs for formats)
#wrapper.logfile.format=LPTM
# Log Level for log file output. (See docs for log levels)
#wrapper.logfile.level=INFO
# Maximum size that the log file will be allowed to grow to before
# the log is rolled. Size is specified in bytes. The default value
# of 0, disables log rolling by size. May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix. For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLOUT it will be automatically
added as suffix of the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
*****# Wrapper Windows Service and Posix Daemon Properties*****
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
*****# Wrapper System Tray Properties*****
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find the port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
*****# Exit Code Properties
# Restart on non zero exit code
*****wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART

```

```
*****
# Trigger actions on console output
*****
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\\trayMessage.gv
wrapper.filter.script.0.args=Exception
*****
# genConfig: further Properties generated by genConfig
*****
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\\bin\\java
wrapper.java.classpath.1 = ${java_home}\\lib\\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\\bin\\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath\\:/a:${carbon_home}\\lib\\xboot\\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 =
-XX:HeapDumpPath=${carbon_home}\\repository\\logs\\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 =
-Djava.endorsed.dirs=${carbon_home}\\lib\\endorsed;${java_home}\\jre\\lib\\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=\
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\\bin\\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\\lib\\tomcat
wrapper.java.additional.15 =
-Djava.util.logging.config.file=${carbon_home}\\repository\\conf\\log4j.properties
wrapper.java.additional.16 = -Dcarbon.config.dir.path=${carbon_home}\\repository\\conf

wrapper.java.additional.17 = -Dcarbon.logs.path=${carbon_home}\\repository\\logs
wrapper.java.additional.18 =
-Dcomponents.repo=${carbon_home}\\repository\\components\\plugins
wrapper.java.additional.19 = -Dconf.location=${carbon_home}\\repository\\conf
wrapper.java.additional.20 =
-Dcom.atomikos.icatch.file=${carbon_home}\\lib\\transactions.properties
wrapper.java.additional.21 = -Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.22 =
-Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
```

```
wrapper.java.additional.23 = -Dcom.sun.jndi.ldap.connect.pool.authentication=simple
wrapper.java.additional.24 = -Dcom.sun.jndi.ldap.connect.pool.timeout=3000
wrapper.java.additional.25 = -Dorg.terracotta.quartz.skipUpdateCheck=true
```

### **Setting up CARBON\_HOME**

Extract the Carbon-based product that you want to run as a Windows service, and then set the Windows environment variable `CARBON_HOME` to the extracted product directory location. For example, if you want to run ESB 4.5.0 as a Windows service, you would set `CARBON_HOME` to the extracted `wso2esb-4.5.0` directory.



### **Running the product in console mode**

You will now verify that YAJSW is configured correctly for running the Carbon-based product as a Windows service.

1. Open a Windows command prompt and go to the `<YAJSW_HOME>/bat/` directory. For example:

```
cd C:\Documents and Settings\yajsw_home\bat
```

2. Start the wrapper in console mode using the following command:

```
runConsole.bat
```

For example:

```
C:\Documents and Settings\yajsw_home\bat>runConsole.bat
```

If the configurations are set properly for YAJSW, you will see console output similar to the following and can now access the WSO2 management console from your web browser via <https://localhost:9443/carbon>.

```
C:\Documents and Settings\yajsw_home\bat>runConsole.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -
jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -c "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 11:12:27 AM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUMENTS\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
```

### Working with the WSO2CARBON service

To install the Carbon-based product as a Windows service, execute the following command in the <YAJSW\_HOME>/bat/ directory:

```
installService.bat
```

The console will display a message confirming that the WSO2CARBON service was installed.

```
C:\Documents and Settings\yajsw_home\bat>installService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -
jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -i "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 12:51:42 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** INSTALLING "WSO2CARBON" *****
Service "WSO2CARBON" installed
Press any key to continue . . .
```

To start the service, execute the following command in the same console window:

```
startService.bat
```

The console will display a message confirming that the WSO2CARBON service was started.

```
C:\Documents and Settings\yajsw_home\bat>startService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -
jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -t "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 1:09:00 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** STARTING "WSO2CARBON" *****
Service "WSO2CARBON" started
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

To stop the service, execute the following command in the same console window:

```
stopService.bat
```

The console will display a message confirming that the WSO2CARBON service has stopped.

```
C:\Documents and Settings\yajsw_home\bat>stopService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\../tmp" -
jar "C:\Documents and Settings\yajsw_home\bat\../wrapper.jar" -p "C:\Documents and Settings\yajsw_home\bat\../conf/wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 11:11:31 AM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** STOPPING "WSO2CARBON" *****
Service "WSO2CARBON" stopped
Press any key to continue . . .

```

To uninstall the service, execute the following command in the same console window:

```
uninstallService.bat
```

The console will display a message confirming that the WSO2CARBON service was removed.

```
C:\Documents and Settings\yajsw_home\bat>uninstallService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\../tmp" -
jar "C:\Documents and Settings\yajsw_home\bat\../wrapper.jar" -r "C:\Documents and Settings\yajsw_home\bat\../conf/wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 1:19:14 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** REMOVING "WSO2CARBON" *****
Service "WSO2CARBON" removed
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

## Building from Source

WSO2 invites you to contribute by downloading the source code from the GitHub source control system, building the product and making changes, and then committing your changes back to the source repository. The following sections describe this process:

- Downloading the source
- Editing the source code
- Building the product
- Committing your changes

**i** Building from source is optional. Users who do not want to make changes to the source code can simply [download the binary distribution](#) of the product and install it.

## Downloading the source

WSO2 products are built on top of WSO2 Carbon Kernel, which contains the Kernel libraries used by all products. When there are changes in the Carbon Kernel, they are bundled and released in a new WSO2 Carbon version (for example, WSO2 Carbon 4.3.0). You can download the complete WSO2 Kernel release using the following repository: <https://github.com/wso2/carbon4-kernel>, which is recommended if you intend to modify the source.

After downloading the source of the Carbon Kernel, execute the following command to download the source of the product: `git clone https://github.com/wso2/product-apim`.

 After the source code is downloaded, you can start editing. However, it is recommended to run a build prior to changing the source code to ensure that the download is complete.

## Editing the source code

Now that you have downloaded the source code for the Carbon project from GitHub, you can prepare your development environment and do the required changes to the code.

1. To edit the source code in your IDE, set up your development environment by running one of the following commands:

IDE	Command	Additional information
Eclipse	<code>mvn eclipse:eclipse</code>	<a href="http://maven.apache.org/plugins/maven-eclipse-plugin">http://maven.apache.org/plugins/maven-eclipse-plugin</a>
IntelliJ IDEA	<code>mvn idea:idea</code>	<a href="http://maven.apache.org/plugins/maven-idea-plugin">http://maven.apache.org/plugins/maven-idea-plugin</a>

2. Add the required changes to the source code.

## Building the product

Ensure that the following prerequisites are in place before you build:

1. Make sure the build server has an active Internet connection to download dependencies while building.
2. Install Maven and JDK. For compatible versions, see [Installation Prerequisites](#).
3. Set the environment variable `MAVEN_OPTS= "-Xms1024m -Xmx4096m -XX:MaxPermSize=1024m"` to avoid the Maven `OutOfMemoryError`.

Use the following Maven commands to build your product:

Command	Description
<code>mvn clean install</code>	The binary and source distributions.
<code>mvn clean install -Dmaven.test.skip=true</code>	The binary and source distributions, without running any of the unit tests.
<code>mvn clean install -Dmaven.test.skip=true -o</code>	The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you have already built the source at least once.

## Committing your changes

You can contribute to WSO2 products by committing your changes to GitHub. Whether you are a committer or a non-committer, you can contribute with your code as explained in the [Get Involved](#) section.

## Running the Product

To run WSO2 products, you start the product server at the command line. You can then run the Management Console to configure and manage the product.



-  The Management Console uses the default HTTP-NIO transport, which is configured in the <PRODUCT\_HOME>/repository/conf/tomcat/catalina-server.xml file. (<PRODUCT\_HOME> is the directory where you installed the WSO2 product you want to run.) You must properly configure the HTTP-NIO transport in this file to access the Management Console. For more information on the HTTP-NIO transport, see the related topics section at the bottom of this page.

The following sections describe how to run the product.

- [Starting the server](#)
- [Accessing the Management Console](#)
- [Stopping the server](#)

#### Starting the server

To start the server, you run <PRODUCT\_HOME>/bin/wso2server.bat (on Windows) or <PRODUCT\_HOME>/bin/wso2server.sh (on Linux/Solaris/Mac OS) from the command prompt as described below. Alternatively, you can install and run the server as a Windows or Linux service (see the related topics section at the end of this page).

1. Open a command prompt by following the instructions below.
  - On Windows: Click **Start -> Run**, type cmd at the prompt, and then press **Enter**.
  - On Linux/Solaris/Mac OS: Establish an SSH connection to the server, log in to the text Linux console, or open a terminal window.
2. Execute one of the following commands:
  - To start the server in a typical environment:
    - On Windows: <PRODUCT\_HOME>\bin\wso2server.bat --run
    - On Linux/Solaris/Mac OS: sh <PRODUCT\_HOME>/bin/wso2server.sh
  - To start the server in the background mode of Linux: sh <PRODUCT\_HOME>/bin/wso2server.sh start
   
To stop the server running in this mode, you will enter: sh <PRODUCT\_HOME>/bin/wso2server.sh stop
  - To provide access to the production environment without allowing any user group (including admin) to log into the Management Console:
    - On Windows: <PRODUCT\_HOME>\bin\wso2server.bat --run -DworkerNode
    - On Linux/Solaris/Mac OS: sh <PRODUCT\_HOME>/bin/wso2server.sh -DworkerNode
  - To check for additional options you can use with the startup commands, type -help after the command, such as:
   
sh <PRODUCT\_HOME>/bin/wso2server.sh -help (see the related topics section at the end of this page).
3. The operation log appears in the command window. When the product server has successfully started, the log displays the message "WSO2 Carbon started in 'n' seconds".

#### Accessing the Management Console

Once the server has started, you can run the Management Console by typing its URL in a Web browser. The following sections provide more information about running the Management Console:

- [Working with the URL](#)
- [Signing in](#)
- [Getting help](#)
- [Configuring the session time-out](#)
- [Restricting access to the Management Console and Web applications](#)

#### Working with the URL

The URL appears next to "Mgt Console URL" in the start script log that is displayed in the command window. For example:

```
[2014-12-04 17:53:26,547] INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} - WSO2 Carbon started in 45 sec
[2014-12-04 17:53:26,787] INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL : https://localhost:9443/carbon/
```

The URL should be in the following format: `https://<Server Host>:9443/carbon`

You can use this URL to access the Management Console on this computer from any other computer connected to the Internet or LAN. When accessing the Management Console from the same server where it is installed, you can type `localhost` instead of the IP address as follows: `https://localhost:9443/carbon`

You can change the Management Console URL by modifying the value of the `<MgtHostName>` property in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. When the host is internal or not resolved by a DNS, map the hostname alias to its IP address in the `/etc/hosts` file of your system, and then enter that alias as the value of the `<MgtHostName>` property in `carbon.xml`. For example:

```
In /etc/hosts:  
127.0.0.1      localhost  
  
In carbon.xml:  
<MgtHostName>localhost</MgtHostName>
```

## **Signing in**

At the sign-in screen, you can sign in to the Management Console using **admin** as both the username and password.

**i** When the Management Console sign-in page appears, the Web browser typically displays an "insecure connection" message, which requires your confirmation before you can continue.

The Management Console is based on the HTTPS protocol, which is a combination of HTTP and SSL protocols. This protocol is generally used to encrypt the traffic from the client to server for security reasons. The certificate it works with is used for encryption only, and does not prove the server identity. Therefore, when you try to access the Management Console, a warning of untrusted connection is usually displayed. To continue working with this certificate, some steps should be taken to "accept" the certificate before access to the site is permitted. If you are using the Mozilla Firefox browser, this usually occurs only on the first access to the server, after which the certificate is stored in the browser database and marked as trusted. With other browsers, the insecure connection warning might be displayed every time you access the server.

This scenario is suitable for testing purposes, or for running the program on the company's internal networks. If you want to make the Management Console available to external users, your organization should obtain a certificate signed by a well-known certificate authority, which verifies that the server actually has the name it is accessed by and that this server actually belongs to the given organization.

## **Getting help**

The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed. To view information about a particular page, click the **Help** link at the top right corner of that page, or click the **Docs** link to open the documentation for full information on managing the product.

## **Configuring the session time-out**

If you leave the Management Console unattended for a defined time, its login session will time out. The default timeout value is 15 minutes, but you can change this in the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml` file as follows.

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

## Restricting access to the Management Console and Web applications

You can restrict access to the Management Console of your product by binding the Management Console with selected IP addresses. You can either restrict access to the Management Console only, or you can restrict access to all Web applications in your server as explained below.

- To control access only to the Management Console, add the IP addresses to the <PRODUCT\_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml file as follows:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="|<IP-address-02>|<IP-address-03>" />
```

The RemoteAddrValve Tomcat valve defined in this file only applies to the Management Console, and thereby all outside requests to the Management Console are blocked.

- To control access to all Web applications deployed in your server, add the IP addresses to the <PRODUCT\_HOME>/repository/conf/context.xml file as follows.

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="|<IP-address-02>|<IP-address-03>" />
```

The RemoteAddrValve Tomcat valve defined in this file applies to each Web application hosted on the WSO2 product server. Therefore, all outside requests to any Web application are blocked.

- You can also restrict access to particular servlets in a Web application by adding a Remote Address Filter to the <PRODUCT\_HOME>/repository/conf/tomcat/web.xml file and by mapping that filter to the servlet URL. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet. The following example from a web.xml file illustrates how access to the Management Console page (/carbon/admin/login.jsp) is granted only to one IP address.

```
<filter>
    <filter-name>Remote Address Filter</filter-name>
    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
    <init-param>
        <param-name>allow</param-name>
        <param-value>127.0.01</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>Remote Address Filter</filter-name>
    <url-pattern>/carbon/admin/login.jsp</url-pattern>
</filter-mapping>
```

- i** Any configurations (including valves defined in the <PRODUCT\_HOME>/repository/conf/tomcat/catalina-server.xml file) apply to all Web applications and are globally available across the server, regardless of the host or cluster. For more information about using remote host filters, see the [Apache Tomcat documentation](#).

## Stopping the server

To stop the server, press **Ctrl+C** in the command window, or click the **Shutdown/Restart** link in the navigation pane in the Management Console. If you started the server in background mode in Linux, enter the following command instead:

```
sh <PRODUCT_HOME>/bin/wso2server.sh stop
```

- [Installing as a Windows Service](#)
- [Installing as a Linux Service](#)

## Upgrading from the Previous Release

The following information describes how to upgrade your API Manager server from the previous release, which is APIM 1.8.0. To upgrade from a version older than 1.8.0, start from the doc that was released immediately after your current release and upgrade incrementally.

- [Migrating the configurations](#)
- [Upgrading the API Manager from 1.8.0 to 1.9.0](#)

### Migrating the configurations

In this section, you move all existing API Manager configurations from the current environment to the new one.

1. Back up the databases in your API Manager instances and synapse configs in of all the tenants, including the `super tenant`. You find the synapse configs in `<APIM_1.8.0_HOME>/repository/deployment/server/synapse-config/default`. For the synapse configurations of tenants, look in `<APIM_1.8.0_HOME>/repository/tenants/<tenant_id>/deployment/server/synapse-config/default`. If you use a **clustered/distributed API Manager setup**, back up the configs in the API Gateway node.
2. Download the API Manager 1.9.0 from <http://wso2.com/products/api-manager/>.
3. Open the `<APIM_1.9.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases. You can copy the configurations from the same file in the API Manager 1.8.0 instance.
  - User Store
  - Registry database
  - API Manager Databases
4. Edit the registry configurations in the `<APIM_1.9.0_HOME>/repository/config/registry.xml` and the user database in the `<APIM_1.9.0_HOME>/repository/conf/user-mgt.xml` file.
5. Move all your synapse configurations by copying and replacing `<APIM_1.8.0_HOME>/repository/deployment/server/synapse-config/default` directory to `<APIM_1.9.0_HOME>/repository/deployment/server/synapse-config/default` directory.

**!** **NOTE:** Do not replace the `_TokenAPI_.xml`, `_RevokeAPI_.xml` and `_AuthorizeAPI_.xml` files in the `/default/api` sub directory unless you use a custom token endpoint. They are application-specific APIs.

**!** When you move the synapse configs, you must copy and replace the files. Make sure you do not delete any existing files in the `<APIM_1.9.0_HOME>/repository/deployment/server/synapse-config/default` directory, such as the `_cors_request_handler.xml`.

This causes errors such as `{org.apache.synapse.mediators.base.SequenceMediator} - Sequence named Value {name = 'null', keyValue = '_cors_request_handler_'} cannot be found {org.apache.synapse.mediators.base.SequenceMediator}`.

### Upgrading the API Manager from 1.8.0 to 1.9.0

1. Stop all running API Manager server instances.
2. Make sure you backed up all the databases and synapse configs as instructed in step 1 of the previous section.
3. Download the [WSO2 API Manager Migration Client v1.9.X](#).
4. Before you run the migration client, open the <APIM\_1.9.0\_HOME>/repository/conf/datasources/master-datasources.xml file, and set the <username>, and <password> elements of the AM\_DB JNDI to that of a user who has permissions to alter tables in the database.

 **Tip:** After you are done running the migration client, you can switch these credentials back to a user with lesser privileges.

For example,

```
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <username>xxxxxxxx</username>
      <password>xxxxxxxx</password>
      ...
    </configuration>
  </definition>
</datasource>
```

5. If the API Manager is running, restart it for the changes to take effect.
6. Extract the file you downloaded in the previous step and do the following:
  - a. Copy the org.wso2.carbon.apimgt.migrate.client-1.9.X.jar file to <APIM\_1.9.0\_HOME>/repository/components/dropins. If you use a **clustered/distributed API Manager setup**, copy the JAR file to all nodes.
  - b. Copy the migration-script folder into <APIM\_1.9.0\_HOME>/. If you use a **clustered/distributed API Manager setup**, copy the migration-script folder to the node that hosts your database.
7. If you are not using the MySQL database with the API Manager, change the query inside <APIM\_1.9.0\_HOME>/migration-scripts/18-19-migration/drop-fk.sql according to your database type. The scripts for each database type are given below:

Database type	Script
MySQL	No changes are required as the default drop-fk.sql file already contains the scripts for MySQL.
H2	SELECT DISTINCT constraint_name FROM information_schema.constraints WHERE table_name = 'AM_APP_KEY_DOMAIN_MAPPING'; ALTER TABLE AM_APP_KEY_DOMAIN_MAPPING DROP CONSTRAINT <temp_key_name>;
Oracle	SELECT constraint_name FROM user_cons_columns WHERE table_name = 'AM_APP_KEY_DOMAIN_MAPPING' GROUP BY constraint_name HAVING COUNT(constraint_name) = 1; ALTER TABLE AM_APP_KEY_DOMAIN_MAPPING DROP CONSTRAINT <temp_key_name>;

MS SQL Server	<pre>SELECT CONSTRAINT_NAME FROM INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE WHERE TABLE_NAME = 'AM_APP_KEY_DOMAIN_MAPPING' AND COLUMN_NAME='CONSUMER_KEY'; ALTER TABLE AM_APP_KEY_DOMAIN_MAPPING DROP CONSTRAINT &lt;temp_key_name&gt;;</pre>
Postgresql	<pre>SELECT CONSTRAINT_NAME FROM information_schema.constraint_table_usage WHERE TABLE_NAME = 'AM_APP_KEY_DOMAIN_MAPPING'; ALTER TABLE AM_APP_KEY_DOMAIN_MAPPING DROP CONSTRAINT &lt;temp_key_name&gt;;</pre>

8. Start the API Manager 1.9.0 with the following command-line options to migrate the database, registry and the file system together or separately.

**!** If you have a **multitenant API Manager setup**, copy the contents from your previous <APIM\_HOME>/repository/tenants directory to the same directory in the API Manager 1.9.0 (do not replace the \_TokenAPI\_.xml, \_RevokeAPI\_.xml and \_AuthorizeAPI\_.xml files in the /default/api sub directory) before you start the server with the following commands.

Description	Command
To migrate the database, registry and file system together, at the same time. Ideally, you use this to migrate a standalone pack.	-Dmigrate=true -DmigrateToVe
To migrate the database only. This migrates the AM_DB database. Please ensure that the <APIM_PUBLISHER>/repository/conf/datasources/master-datasources.xml file has an entry for AM_DB.	-DmigrateDB=t -DmigrateToVe
To migrate the registry only. This migrates the registry-related resources such as .rxt and swagger definitions.	-DmigrateReg= -DmigrateToVe
To migrate the file system only. This migrates the synapse config files such as APIs that reside in the file system. Therefore, you must run this command on the Gateway node/s of a distributed APIM setup.	-DmigrateFS=t -DmigrateToVe

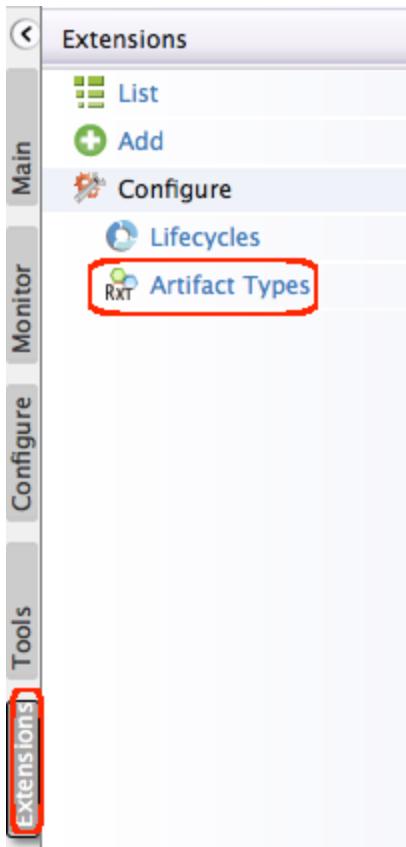
**Tip:** Make sure you do not delete any new files added in <APIM\_1.9.0> or if you get an 'artifact not found' exception

**Tip:** If you are using a **standalone API Manager setup**, you can migrate all the resources together with the **-Dmigrate=true -DmigrateToVersion=1.9** command, or perform single-resource migrations using the other commands separately. If you execute **-Dmigrate=true -DmigrateToVersion=1.9**, you do not need to execute the rest of the commands.

**Tip:** If you are using a **clustered/distributed API Manager setup**, run with the following options **-DmigrateDB=true -DmigrateReg=true -DmigrateToVersion=1.9** in the API Publisher node and the **-DmigrateFS=true -DmigrateToVersion=1.9** options in the API Gateway node.

**Optionally**, to migrate the API stats DB, use the following options when running the API Publisher node: **-DmigrateStats=true -DmigrateToVersion=1.9** when running on the API Publisher node.

9. Log in to the Management Console of the API Manager and select **Extensions -> Artifact Types** menu.



10. Click the **Delete** link associated with the api artifact type. This removes the reference to the old api artifact type, allowing the latest type to be automatically loaded when the server is restarted.

Name	Actions
api	<a href="#">View/Edit</a> <a href="#">Delete</a>
documentation	<a href="#">View/Edit</a> <a href="#">Delete</a>
provider	<a href="#">View/Edit</a> <a href="#">Delete</a>
reply	<a href="#">View/Edit</a> <a href="#">Delete</a>
topic	<a href="#">View/Edit</a> <a href="#">Delete</a>

[+ Add new Artifact](#)

11. Do the following to re-index the artifacts in the registry:

- Rename the `<lastAccessTimeLocation>` element in the `<APIM_1.9.0_HOME>/repository/conf/registry.xml` file. If you use a **clustered/distributed API Manager setup**, change the file in the API Publisher node. For example, change the `/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccessstime` registry path to `/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccessstime_1`.
- Shut down the API Manager 1.9.0, back up and delete the `<APIM_1.9.0_HOME>/solr` directory and restart the server.

This concludes the upgrade process.

 **Tip:** If you are using a **clustered/distributed API Manager setup**, note that all config file changes must be

done in the API Gateway node.

 **Tip:** The migration client that you use in this guide automatically migrates your tenants, workflows, external user stores etc. to the upgrade environment. There is no need to migrate them manually.

 **Tip:** If you are using WSO2 BAM with the APIM, be sure to change the versions of the existing APIM statistics-related stream definitions of BAM in the <APIM\_HOME>/repository/conf/api-manager.xml file. You also have to change the BAM Hive scripts accordingly.

## Get Involved

All WSO2 products are 100% open source and released under the Apache License Version 2.0. WSO2 welcomes anyone who is interested in WSO2 products to become a contributor by getting involved in the WSO2 community and helping with the development of WSO2 projects.

**How can I get involved in the community?**

Contributing as a non-committer – anyone can do it!

- Overview of the WSO2 repository
- Contributing to the WSO2 code base
- [WSO2 GitHub Guidelines](#)

**How can I get involved in the community?**

You can get involved in the WSO2 community in various ways:

- **Use WSO2 products**

The latest binary packs that correspond to the WSO2 product releases can be downloaded freely via the respective product pages on the [WSO2 website](#). We recommend that you download and use WSO2 products so that you can discover the advantages of our lean middleware stack. Your feedback on our products is much appreciated, as it will help us to drive our product roadmaps and the underlying technology. For information on product releases, go to the [Release Matrix](#). For tutorials, articles, white papers, webinars, WSO2 documentation, and other learning resources, look in the Resources menu on the [WSO2 website](#).

- **Join WSO2 mailing lists**

Many WSO2 mailing lists are open to the public, so anyone interested in WSO2 products can monitor the mail threads. You can subscribe to the [dev@wso2.org](mailto:dev@wso2.org) and [architecture@wso2.org](mailto:architecture@wso2.org) mailing lists to get involved in the discussions on WSO2 development. For more information on subscribing to these mailing lists, see [WSO2 Mailing Lists](#).

- **Participate in user forums**

The WSO2 team monitors and participates in the discussions on [Stack Overflow](#). If you have any technical or programming questions related to WSO2 products, post them on Stack Overflow. Be sure to tag your question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you cannot find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at [WSO2 Mailing Lists](#). We also encourage you to contribute by answering your fellow users' questions on Stack Overflow.

- **Report bugs**

WSO2 has a public [bug-tracking system](#) that you can use to report issues, submit enhancement requests, and track and comment on issues. You can also use this system to report issues related to [WSO2 product documentation](#). If you find a bug, first search the dev mailing list to see if someone has faced the same issue, and also check the bug-tracking system. If you can't find any information on the issue, create an issue in the bug-tracking system with as much information as possible, including the product version you were using, how to reproduce the issue, etc.

- **Contribute to the WSO2 code base**

WSO2 invites you to contribute by providing patches for bug fixes or features. For this purpose you can check out the source of the relevant GitHub repository, build the product, and make changes. You can then contribute your changes by sending a [pull a request](#) for review. For more information, see the next section.

## Contributing as a non-committer – anyone can do it!

**Anyone** (not just committers) can share contributions to WSO2's open-source software products. Your work will be recognized: if your contribution – feature enhancement, bug fix, or other improvements – is accepted, your name will be included as an author in the official commit logs. Read on for details on how you can contribute.

### Overview of the WSO2 repository

WSO2 uses [Git](#) as its source control management system. The [WSO2 Git repository](#) maintains the code repository and the active build for continuous delivery incorporated with integrated automation.

### Contributing to the WSO2 code base

Follow these instructions to contribute to the WSO2 code base. Be sure to follow the [WSO2 GitHub Guidelines](#).

1. [Fork](#) the respective code base to your Git account.
2. Clone the code base to your local machine.

```
git clone <GitHub-REPOSITORY-URL>
```

If you are not sure which repository needs to be cloned, send an email to [dev@wso2.org](mailto:dev@wso2.org).

3. Build the product using Maven.

#### Prerequisites

- Install Maven and JDK. See the [Installation Prerequisites](#) page for compatible versions.
- Set the environment variable `MAVEN_OPTS= "-Xms768m -Xmx3072m -XX:MaxPermSize=1200m"` to avoid the maven `OutOfMemoryError`.
- Make sure the build server has an active Internet connection to download dependencies while building.

Use the following commands to create complete release artifacts of a WSO2 product, including the binary and source distributions.

Command	Description
<code>mvn clean install</code>	The binary and source distributions.
<code>mvn clean install -Dmaven.test.skip=true</code>	The binary and source distributions, without running any of the unit tests.
<code>mvn clean install -Dmaven.test.skip=true -o</code>	The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you've already built the source at least once.

4. If you need to add a new file to the repository:

- a. Add the new file.

```
git add <FILE-NAME>
```

For example:

```
git add mycode.java
```

- b. Commit the newly added file to your local repository.

```
git commit -m "<COMMIT-MESSAGE>"
```

For example:

```
git commit -m "Adding a new file"
```

5. If you need to update an existing file in the repository:

- a. Open the file that you want to update and make the necessary changes.

- b. Commit the changes to your local repository.

```
git commit -m "<COMMIT-MESSAGE>" -a
```

For example:

```
git commit -m "Updated the clauses in the terms and conditions file" -a
```

## 6. Sync your changes with the upstream repository.

```
git remote add <TAG-NAME> <UPSTREAM-GIT-REPO-URL>
git fetch <TAG-NAME>
git merge <TAG-NAME>/<BRANCH-NAME>
```

For example:

```
git remote add wso2_upstream https://github.com/wso2/wso2-synapse.git
git fetch wso2_upstream
git merge wso2_upstream/master
```

## 7. Push the changes to your own Git repository.

```
git push
```

## 8. Send a [Git pull request](#) to the WSO2 Git repository and add the URL of the Git pull request in the JIRA that corresponds to the patch. Your pull request will be authorized only after it is reviewed by the team lead or release manager or responsible person for the corresponding Git repository.

For more information on using GitHub, see the related help articles [Fork a Repo](#) and [Using Pull Requests](#).

### **WSO2 GitHub Guidelines**

- **The respective WSO2 Git repository should be forked**

When contributing to WSO2 code base by way of a patch, make sure you identify the correct Git repository that needs to be forked. For more information on WSO2 Git repositories, see [WSO2 GitHub Repositories](#). If you still are not sure which repository needs to be cloned, send an email to [dev@wso2.org](mailto:dev@wso2.org) so that a WSO2 team member can advise you.

- **Do not build any dependencies**

You do not need to build any dependencies, as everything you need will be automatically fetched from the Maven repository (Nexus) when you are building the product on your machine. Make sure the build server has an active Internet connection to download dependencies while building.

- **Always sync with the forked repository before issuing a pull request**

There is a high possibility that the forked repository may differ from the upstream repository (remote repository that was forked) that you initially forked. Therefore, always sync the repository to prevent pull requests from being rejected.

## **WSO2 GitHub Repositories**

The following are the WSO2 GitHub repositories that need to be forked, so that you can contribute to the WSO2 community by offering patches for bug fixes or features for WSO2 products.

- Kernel level Git repositories
- Platform level Git repositories
- Mobile platform Git repositories
- Product level Git repositories
- Other WSO2 Git repositories

### **Kernel level Git repositories**

Repo URL	Description
carbon-kernel	Carbon 5 kernel repo
carbon4-kernel	Carbon 4 kernel repo

***Platform level Git repositories***

Repo URL	Description
carbon-analytics	Contains components and features related to analytics services.
carbon-apimgt	Contains components and features related to API management.
carbon-appmgt	Contains components and features related to application management.
carbon-business-messaging	Contains the components and features related to business messaging.
carbon-business-process	Contains components and features related to business processes.
carbon-commons	Contains common components and features shared across the platform projects.
carbon-data	Contains components and features related to data services.
carbon-deployment	Contains components and features related to web application and service development (i.e., JavaEE WebProfile support, JAX-WS/RS service deployment, Webapp monitoring dashboards etc. ).
carbon-event-processing	Contains components and features related to event processing services.
carbon-governance	Contains components and features related to governance services.
carbon-mediation	Contains components and features related to mediation services.
carbon-ml	Contains components and features related to machine learner.
carbon-multitenancy	
carbon-parent	
carbon-platform-automated-test-suite	Contains WSO2 product integration test suites and Platform test suites with ant based test executor.
carbon-platform-integration	Contains WSO2 test automation framework modules.
carbon-platform-integration-utils	Contains utilities related to WSO2 test automation framework which is common to the whole product platform.
carbon-qos	Contains components and features related to quality of service.
carbon-registry	Contains components and features related to registry services.
carbon-rules	Contains components and features related to business rules.
carbon-storage-management	Contains sources corresponding to the components that are primarily being used for storage provisioning and management related tasks. Out of all the components being maintained within this particular repository some components (i.e., Cassandra, HDFS) are used across the platform. In addition, some of the tools developed for storage browsing (i.e., Cassandra-Explorer etc.) too are part of this repository.
carbon-store	
carbon-utils	Contains ntask, remote-tasks, ndatasource etc.

***Mobile platform Git repositories***

Repo URL	Description

emm-agent-android	Maintains the Android agent that is used to enroll the device to EMM server.
emm-agent-ios	Maintains the iOS agent that is used to enroll the device to EMM server.

**Product level Git repositories**

Product Name	Repo URL	Description
API Manager	product-apim	Maintains sources corresponding to building and packaging of WSO2 API manager distribution.
App Factory	product-af	Maintains sources corresponding to building and packaging of WSO2 APP Factory distribution.
Application Server	product-as	Maintains sources corresponding to building and packaging of WSO2 Application Server distribution.
Business Activity Monitor	product-bam	Maintains sources corresponding to building and packaging of WSO2 Business Activity Monitor distribution.
Business Process Server	product-bps	Maintains sources corresponding to building and packaging of WSO2 Business Process Server distribution.
Business Rules Server	product-brs	Maintains sources corresponding to building and packaging of WSO2 Business Rules Server distribution.
Complex Event Processor	product-cep	Maintains sources corresponding to building and packaging of WSO2 Complex Event Processor distribution.
Connected Device Management Framework	product-cdm	Maintains sources corresponding to building and packaging of WSO2 Connected Device Management Framework distribution .
Data Analytics Server	product-bam	Maintains sources corresponding to building and packaging of WSO2 Data Analytics Server distribution.
Data Services Server	product-dss	Maintains sources corresponding to building and packaging of WSO2 Data Services Server distribution.
Enterprise Mobility Manager	product-emm	Maintains sources corresponding to building and packaging of WSO2 Enterprise Mobility Manager distribution.
Enterprise Service Bus	product-esb	Maintains sources corresponding to building and packaging of WSO2 Enterprise Service Bus distribution.
Enterprise Store	product-es	Maintains sources corresponding to building and packaging of WSO2 Enterprise Store distribution.
Governance Registry	product-greg	Maintains sources corresponding to building and packaging of WSO2 Governance Registry distribution.
Identity Server	product-identity	Maintains sources corresponding to building and packaging of WSO2 Identity Server distribution.
Message Broker	product-mb	Maintains sources corresponding to building and packaging of WSO2 Message Broker distribution.
Private PaaS	product-private-paas	Maintains sources corresponding to building and packaging of WSO2 Private PaaS distribution.
Storage Server	product-ss	Maintains sources corresponding to building and packaging of WSO2 Storage Server distribution.

Task Server	product-ts	Maintains sources corresponding to building and packaging of WSO2 Task Server distribution.
Developer Studio	developer-studio	Maintains sources corresponding to building and packaging of WSO2 Developer Studio distribution.

#### ***Other WSO2 Git repositories***

The following are GitHub repository URLs that correspond to independent projects managed by WSO2:

Repo URL	Description
andes	Message broker core engine implementation.
balana	XACML core engine implementation.
charon	SCIM core engine implementation.
esb-connectors	Collection of connectors that allows you to interact with WSO2 ESB's third-party product function.
jaggery	This repo contains Jaggeryjs. Jaggery is a framework used to write webapps and HTTP-focused web services for all aspects of the application: front-end, communication, Server-side logic and persistence in pure Javascript.
jaggery-extensions	This contains extensions for the Jaggery framework.
orbit	Used to create OSGi bungles out of third-part dependencies.
siddhi	Complex event processing core engine implementation.

# User Guide

The user guide provides information about the features, functionality, solution development, testing and debugging options of WSO2 API Manager.

- [Key Concepts](#)
- [Tutorials](#)
- [Configuring the API Manager](#)
- [Extending the API Manager](#)
- [Working with Security](#)
- [Working with Statistics](#)

## Key Concepts

Let's take a look at some concepts and terminology that you need to know in order to follow the use cases.

[ [API Manager components](#) ] [ [Users and roles](#) ] [ [API lifecycle](#) ] [ [Applications](#) ] [ [Access tokens](#) ] [ [Throttling tiers](#) ] [ [API visibility and subscription](#) ] [ [API documentation visibility](#) ] [ [API resources](#) ] [ [HTTP methods](#) ] [ [Cross-origin resource sharing](#) ] [ [OAuth scopes](#) ] [ [API templates](#) ] [ [Endpoints](#) ] [ [Sequences](#) ] [ [Caching](#) ]

### **API Manager components**

A component is made up of one or more [OSGi](#) bundles. A bundle is the modularization unit in OSGi, similar to a JAR file in Java. The component-based architecture of all WSO2 products gives developers flexibility to remove or add features with minimum dependencies.

The API Manager comprises the following high-level components:

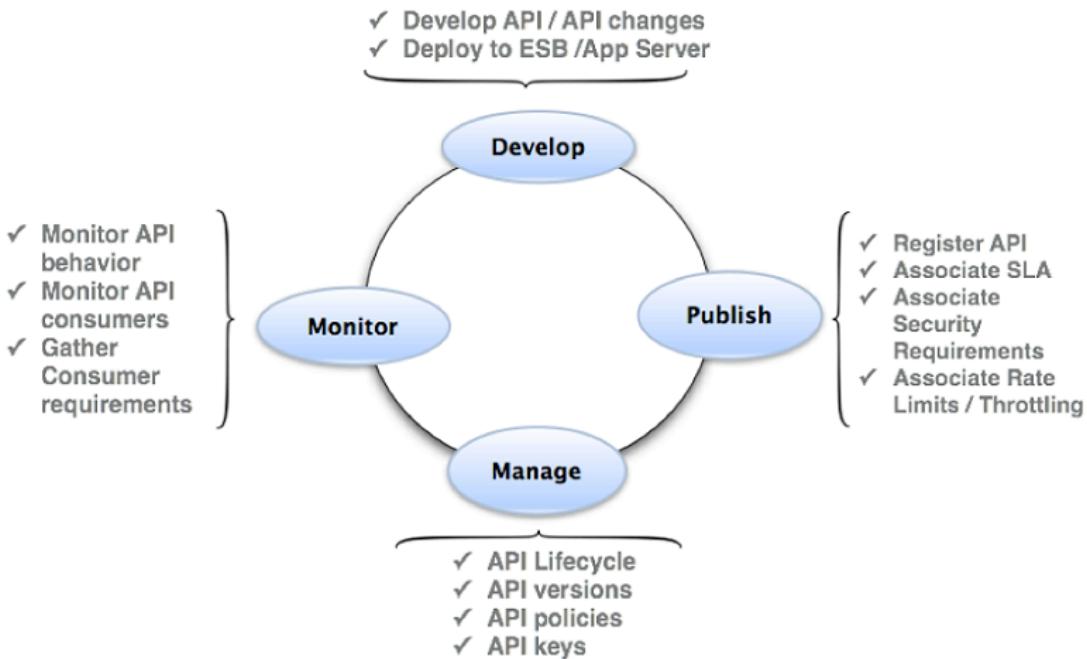
#### ***API Publisher***

API development is usually done by someone who understands the technical aspects of the API, interfaces, documentation, versions etc., while API management is typically carried out by someone who understands the business aspects of the APIs. In most business environments, API development is a responsibility that is distinct from API publication and management.

WSO2 API Manager provides a simple Web interface called **WSO2 API Publisher** for API development and management. It is a structured GUI designed for API creators to develop, document, scale and version APIs, while also facilitating more API management-related tasks such as publishing API, monetization, analyzing statistics, and promoting.

The API Publisher URL is `https://<YourHostName>:9443/publisher` and it is accessible on HTTPS only. The default credentials are admin/admin.

The diagram below shows the common lifecycle activities of an API developer/manager:

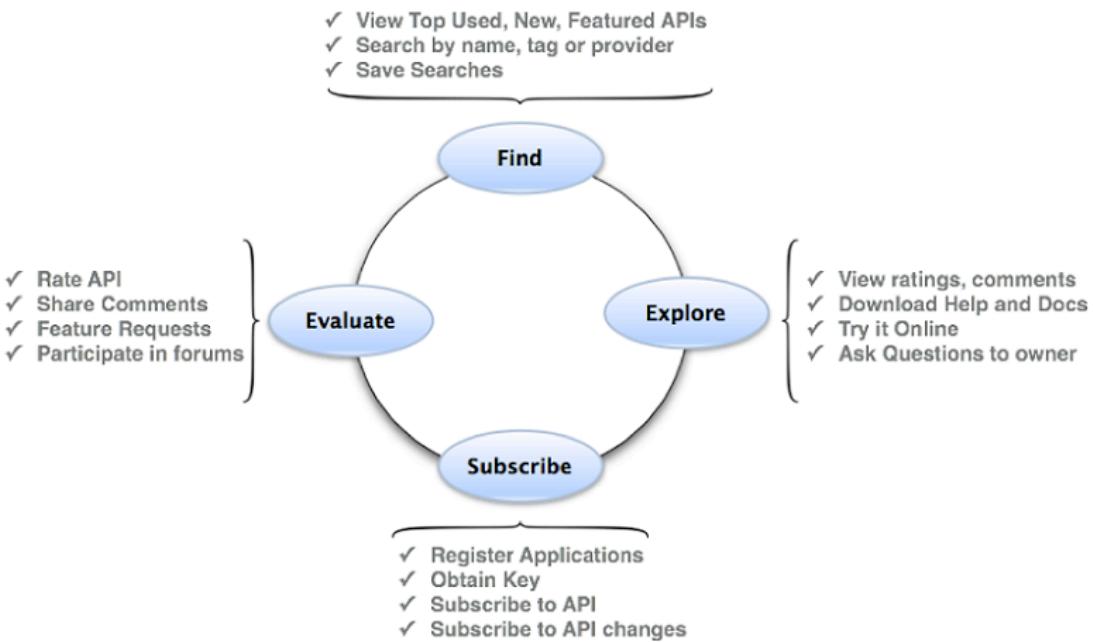


### API Store

The API Store Web application provides a collaborative interface for API publishers to host and advertise their APIs and for API consumers to self register, discover, evaluate, subscribe to and use secured, protected, authenticated APIs.

The API Store URL is <https://<YourHostName>:9443/store> and it is accessible on HTTPS only. The default credentials are admin/admin.

The diagram below shows common API consumer lifecycle activities:



### API Gateway

A runtime, backend component (an API proxy) developed using WSO2 ESB. API Gateway secures, protects, manages, and scales API calls. It intercepts API requests, applies policies such as throttling and security using handlers and manages API statistics. Upon validation of a policy, the Gateway passes Web service calls to the actual backend. If the service call is a token request, the Gateway passes it directly to the [Key Manager](#).

When the API Manager is running, you can access the Gateway using the URL <https://localhost:9443/carbon>. You integrate a monitoring and statistics component to the API Manager without any additional configuration effort. This monitoring component integrates with WSO2 Business Activity Monitor, which can be deployed separately to analyze events. For more information, see [Publishing API Runtime Statistics](#).

- i** Although the API Gateway contains ESB features, it is recommended not to use it for ESB-specific tasks. Use it only for Gateway functionality related to API invocations. For example, if you want to call external services like SAP, use a separate ESB cluster for that.

### **Key Manager**

Manages all clients, security and access token-related operations. The Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions and API invocations. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database.

The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway. All tokens used for validation are based on the OAuth 2.0.0 protocol. Secure authorization of APIs is provided by the OAuth 2.0 standard for key management. The API Gateway supports API authentication with OAuth 2.0, and enables IT organizations to enforce rate limits and throttling policies. The Key Manager properly decouples the operations for creating OAuth applications and validating access tokens so that you can even plug in a third party-authorization server for key validations.

You can avoid making the Gateway connect with the Key Manager every time it receives an API invocation call, by enabling API Gateway [caching](#). When caching is not enabled, a verification call happens every time the Gateway receives an API invocation call.

F  
a Web service call  
• Through a [Thrift](#) call (Thrift is the default communication protocol and is much faster than SOAP over HTTP)

If your setup has a cluster of multiple Key Manager nodes that are fronted by a load balancer that does not support Thrift, change the key management protocol from Thrift to WSClient using the `<KeyValidatorClientType>` element in `<APIM_HOME>/repository/conf/api-manager.xml` file. Thrift uses TCP load balancing.

In a typical production environment, you can configure one of the following setups:

- Configure a WSO2 API Manager instance as the Key Manager in a separate server. See [Product Profiles](#).
- Configure an instance of WSO2 Identity Server as the Key Manager. See [Configuring WSO2 Identity Server as the Key Manager](#).
- Configure a third-party authorization server for key validations and an API Manager instance for the rest of the key management operations. See [Configuring a Third-Party Key Manager](#).

### **Handlers**

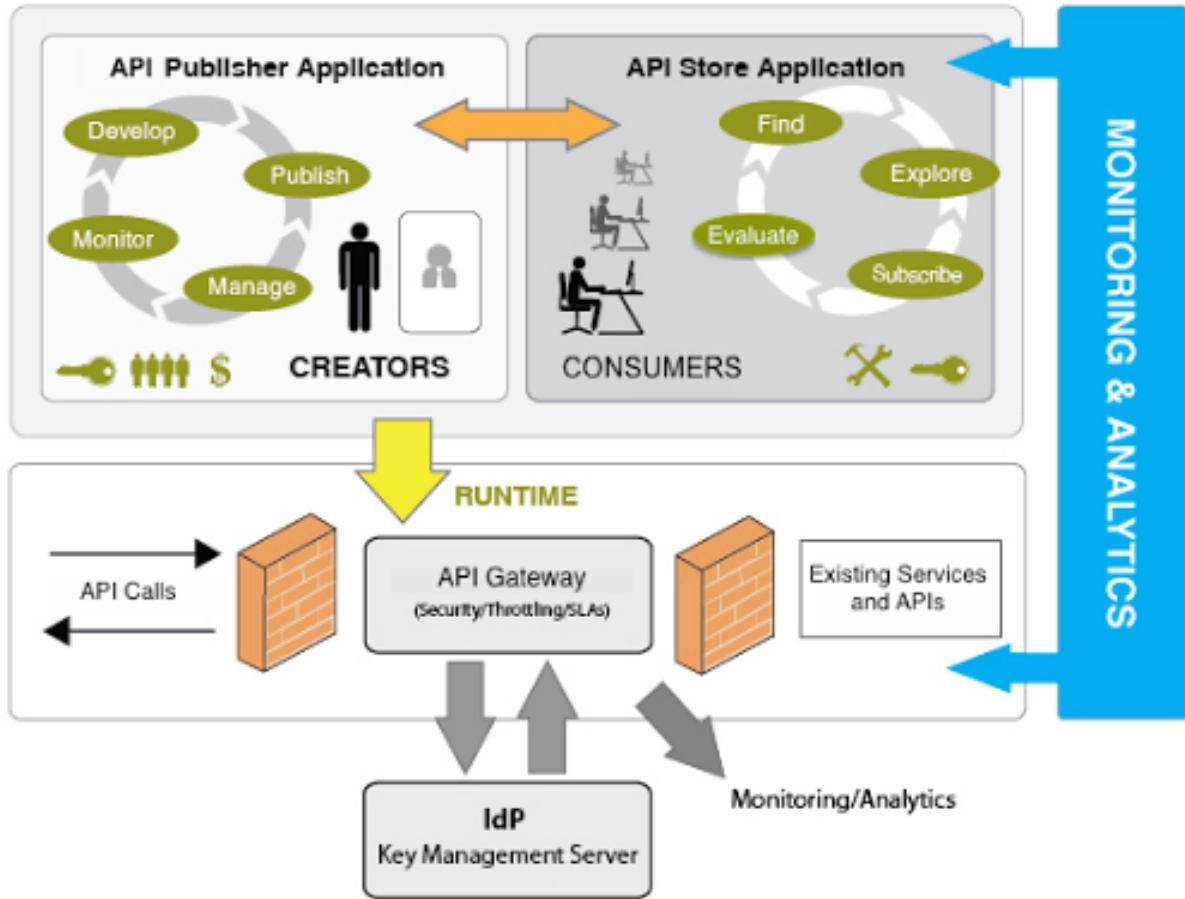
W You find the default handlers in any API's Synapse definition when an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration.

a detailed description of handlers and how to write a custom handler, see [Writing Custom Handlers](#).

### **Statistics**

Additionally, statistics are provided by the monitoring component, which integrates with WSO2 BAM.

The components are depicted in the diagram below:



## Users and roles

The API Manager offers four distinct community roles that are applicable to most enterprises:

- **Admin**: The API management provider who hosts and manages the API Gateway. S/he is responsible for creating user roles in the system, assign them roles, managing databases, security etc. The Admin role is available by default with credentials admin/admin.
- **Creator**: a creator is a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions etc.) and uses the API publisher to provision APIs into the API store. The creator uses the API Store to consult ratings and feedback provided by API users. Creator can add APIs to the store but cannot manage their lifecycle.
- **Publisher** : a publisher manages a set of APIs across the enterprise or business unit and controls the API lifecycle, subscriptions and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **Subscriber** : a subscriber uses the API store to discover APIs, read the documentation and forums, rate/comment on the APIs, subscribes to APIs, obtain access tokens and invoke the APIs.

**Tip:** See [Managing Users and Roles](#) for more information.

## API lifecycle

An API is the published interface, while the service is the implementation running in the backend. APIs have their own lifecycles that are independent to the backend services they rely on. This lifecycle is exposed in the API publisher Web interface and is managed by the API publisher role.

The following stages are available in the default API lifecycle:

- **CREATED:** API metadata is added to the API Store, but it is not deployed in the API gateway and therefore, is not visible to subscribers in the API Store.
  - **PROTOTYPED:** the API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. Users can invoke the API without a subscription.
  - **PUBLISHED:** The API is visible in the API Store and available for subscription.
  - **DEPRECATED:** When an API is deprecated, new subscriptions are disabled. But the API is still deployed in the Gateway and is available at runtime to existing subscribers. Existing subscribers can continue to use it as usual until the API is retired.
  - **RETIRED:** The API is unpublished from the API gateway and deleted from the store.
  - **BLOCKED:** Access to the API is temporarily blocked. Runtime calls are blocked and the API is not shown in the API Store anymore.
- 

## Applications

An application is a logical collection of APIs. An application is primarily used to decouple the consumer from the APIs. It allows you to :

- Generate and use a single key for multiple APIs
- Subscribe multiple times to a single API with different SLA levels

You subscribe to APIs through an application. Applications are available at different SLA levels, and have application-level throttling tiers engaged in them. A throttling tier determines the maximum number of calls you can make to an API during a given period of time.

The API Manager comes with a pre-created, default application, which allows unlimited access by default. You can also create your own applications.

---

## Access tokens

An **access token** is a simple string that is passed as an HTTP header of a request. For example, "Authorization : Bearer NtBQkXoKELu0H1a1fQ0DWf06IX4a." Access tokens authenticate API users and applications, and ensure better security (e.g., prevent **DoS attacks**). If a token that is passed with a request is invalid, the request is discarded in the first stage of processing. Access tokens work equally well for SOAP and REST calls.

There are two types of access tokens:

- User access tokens
- Application access tokens

### User access tokens

Tokens to authenticate the final user of an API. User access tokens allow you to invoke an API even from a third-party application like a mobile app. You generate/renew a user access token by calling the Login API through a REST client. For more information, see [Token API](#).

### Application access tokens

Tokens to authenticate an application, which is a logical collection of APIs. You can access all APIs associated with an application using a single token, and also subscribe multiple times to a single API with different SLA levels. Application access tokens leverage OAuth2 to provide simple key management.

The steps below describe how to generate/renew application access tokens:

1. Log in to the API Store.
2. Click the **My Subscriptions** menu, select the application from the drop-down list and click the **Generate** or **Regenerate** buttons to create and renew access tokens.

The screenshot shows the 'Subscriptions' section of the WSO2 API Manager interface. At the top, there are tabs for 'API Store', 'APIs', 'Prototyped APIs', 'My Applications', 'My Subscriptions' (which is highlighted with a red box), 'Statistics', and 'Tools'. Below the tabs is a search bar with a magnifying glass icon and a link to 'Go to API Publisher'. On the left, a sidebar titled 'Recently Added' shows a single entry: 'PhoneVerification-1...' by 'nirdesha.wso2.com@...', with a 5-star rating. The main content area is titled 'Subscriptions' and contains a message about generating keys for multiple APIs. It lists 'Applications With Subscriptions' (DefaultApplication) and shows 'Keys - Production' and 'Keys - Sandbox'. The 'Production' section displays an 'Access Token' (K7va7zRj1EJS2PQuQ24TkR5Zqy8a), a 'Re-generate' button (highlighted with a red box), a 'Token Validity' field set to '3600 Seconds', and an 'Allowed Domains' section with 'ALL' selected. The 'Sandbox' section shows a message that keys are not yet generated, a 'Generate' button (highlighted with a red box), and an 'Allowed Domains' section with 'ALL' selected. A checkbox labeled 'Show Keys' is checked in the top right corner.

Whenever an API call happens, the Gateway checks if the request originated from an allowed domain and grants access accordingly. You can specify these domains in the **Allowed Domains** text box. This ensures that clients from a restricted domain cannot access an API even if an application key is stolen (when the key is placed in client-side JS code).

**i** **Tip:** When the client makes a request to an API that is only allowed to some domains, the request message must have an HTTP header to specify its domain name. Sending this header is mandatory only if the API is restricted to certain domains. An admin can configure this header name using `<ClientDomainHeader>` element under the `<APIGateway>` element in `<APIM_HOME>/repository/conf/api-manager.xml`.

For example, if the file contains `<ClientDomainHeader>domain</ClientDomainHeader>`, then the API invocation request must contain an HTTP header called `domain` with values as shown in the example below: `curl -v -H "Authorization: Bearer xxx" -H "domain: wso2.com" h ttp://localhost:8280/twitter/1.0.0/search.atom?q=cat`

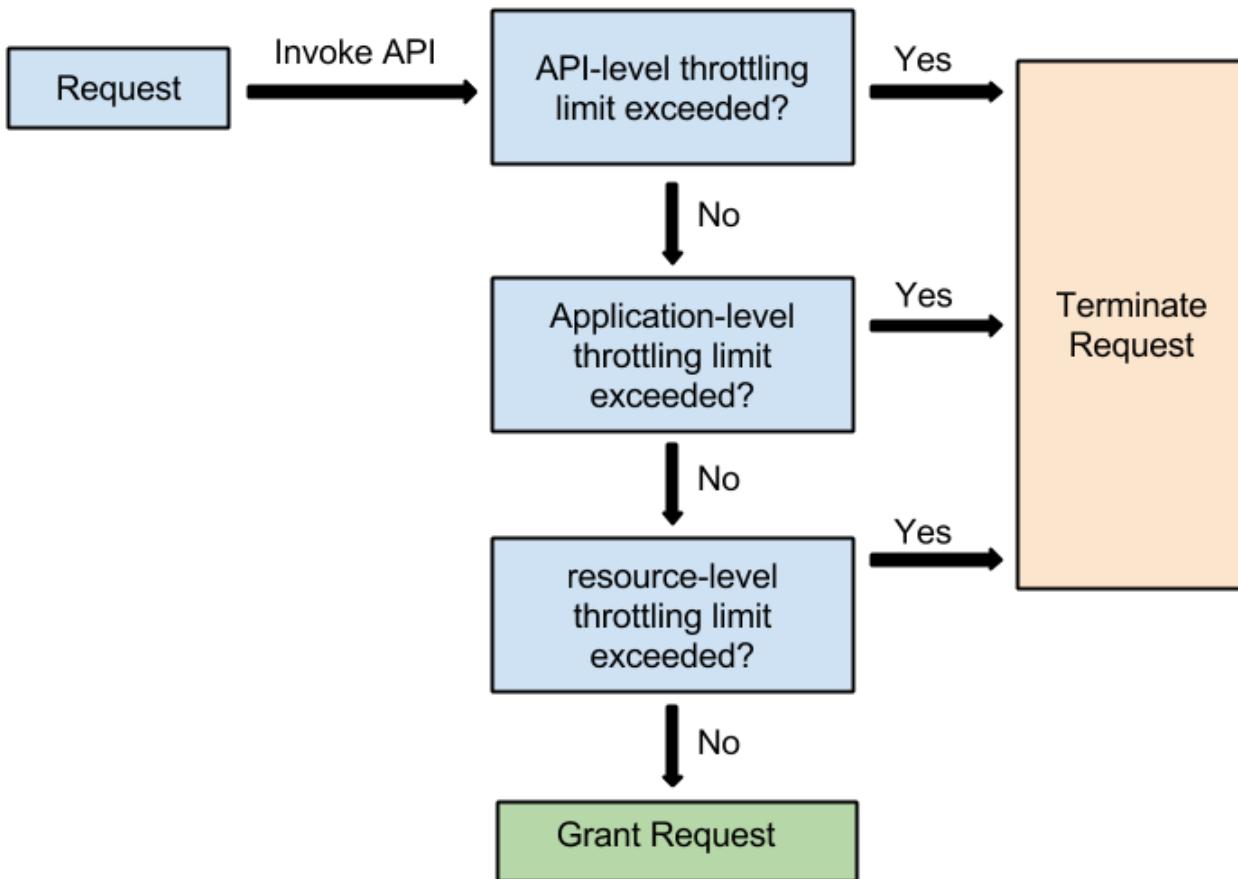
**✓** **Tip:** When you generate access tokens to APIs protected by scope/s, a **Select Scopes** button is displayed in the **My Subscriptions** page for you to select the scope/s first and then generate the token to it.

## Throttling tiers

Throttling allows you to limit the number of successful hits to an API during a given period of time, typically in cases such as the following:

- To protect your APIs from common types of security attacks such as denial of service (DOS)
- To regulate traffic according to infrastructure availability
- To make an API, application or a resource available to a consumer at different levels of service, usually for monetization purpose

You can define throttling in the API, application and resource levels. The final request limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together.



**Example:** Lets say two users subscribed to an API using the Gold subscription, which allows 20 requests per minute. They both use the application App1 for this subscription, which again has a throttling tier set to 20 requests per minute. All resource level throttling tiers are unlimited. In this scenario, although both users are eligible for 20 requests per minute access to the API, each ideally has a limit of only 10 requests per minute. This is due to the application-level limitation of 20 requests per minute.

#### Different levels of throttling

Let's take a look at the different levels of throttling:

[ [API-level throttling](#) ] [ [Application-level throttling](#) ] [ [Resource-level throttling](#) ] [ [IP-level throttling](#) ]

#### API-level throttling

API-level throttling tiers are defined when managing APIs using the API Publisher portal. The UI looks as follows:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'Browse', 'Add' (which is highlighted with a red box), 'All Statistics', 'My APIs', 'Subscriptions', and 'Statistics'. The main area has a progress bar at the top with steps '1 Design', '2 Implement', and '3 Manage' (also highlighted with a red box). Below the progress bar, the title 'PhoneVerification : /phoneverify/1.0.0' is displayed. Underneath the title, the section 'Configurations' is shown. There are two dropdown menus: 'Tier Availability:' set to 'Bronze' (highlighted with a red box) and 'Applications' set to 'DefaultApplication'. A checkbox labeled 'Make this default version' is checked.

After API-level throttling tiers are set and the API is published, at subscription time, the consumers of the API can log in to the **API Store** and select which tier they are interested in as follows:

The screenshot shows the API Store interface for the 'PhoneVerification - 1.0.0' API. On the left, it shows a user profile for 'admin'. The main details are listed: Rating (N/A), Version (1.0.0), Status (PUBLISHED), and Updated (23/May/2014 11:04:57 AM IST). To the right, there are sections for 'Applications' (set to 'DefaultApplication') and 'Tiers' (set to 'Bronze'). A note below says 'Allows 1 request(s) per minute.' At the bottom is a 'Subscribe' button.

According to the tiers the subscriber selects, s/he is granted a maximum number of requests to the API. The default tiers are as follows:

- **Bronze**: 1 request per minute
- **Silver**: 5 requests per minute
- **Gold**: 20 requests per minute
- **Unlimited**: Allows unlimited access (you can disable the Unlimited tier by editing the <TierManagement> node of the <APIM\_HOME>/repository/conf/api-manager.xml file)

**Setting tier permissions:** Users with Manage Tiers permission can set role-based permissions to API-level access throttling tiers. This is done using the **Tier Permissions** menu in the API Publisher as shown below. For each tier, you can specify a comma-separated list of roles and either Allow or Deny access to the list.

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following navigation items:

- APIS
- Browse
- Add
- All Statistics
- MY APIS**
- Subscriptions
- Statistics
- TIER PERMISSIONS**
- Tier Permissions** (highlighted in blue)

The main content area is titled "Tier Permissions". It contains a table with two rows, each representing a tier:

Tier	Permissions
Bronze	<input checked="" type="radio"/> Allow <input type="radio"/> Deny roles Internal/everyone <small>Comma separated list (Ex: role1,role2,role3)</small>
Gold	<input checked="" type="radio"/> Allow <input type="radio"/> Deny roles Internal/everyone

A blue "Update Permissions" button is located at the bottom right of the table.

A subscriber logged into the API Store can consume APIs using a specific tier only if s/he is assigned to a role that is allowed access. In the API Store, the subscriber sees a list of tiers that is filtered based on the subscriber's role. Only the ALLOWED roles appear here. By default, all tiers are allowed to everyone.

### Application-level throttling

Application-level throttling tiers are defined at the time an application is created in the API Store as shown below:

Use applications to subscribe to APIs and manage access keys. There is DefaultApplication pre-created to use and it can add more applications on this page.

**Add New Application**

Characters left: 70

Name:

Throttling Tier: **Unlimited** Allows unlimited requests

Callback URL:

Description:

**Add**

An application is a logical collection of one or more APIs and is required to subscribe to an API. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times with different SLA levels.

An application is available to a consumer at different levels of service. For example, if you have infrastructure limitations in facilitating more than a certain number of requests to an application at a time, the throttling tiers can be set accordingly so that the application can have a maximum number of requests within a defined time. The default throttling levels are as follows:

- **Bronze:** 1 request per minute
- **Silver:** 5 requests per minute
- **Gold:** 20 requests per minute
- **Unlimited:** Unlimited access. The **Default Application**, which is provided out of the box has Unlimited tier set. You have the option to set it to a restricted limit.

### Resource-level throttling

An API is made up of one or more resources. Each resource handles a particular type of request and is analogous to a method (function) in a larger API. Resource-level throttling tiers are set to HTTP verbs of an API's resources when Managing APIs using the API Publisher portal as shown below:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like APIs, Browse, Add, All Statistics, My APIs, Subscriptions, Statistics, Tier Permissions, and Tier Permissions. The 'Tier Permissions' section is currently selected. At the top, there's a navigation bar with three steps: Design, Implement, and Manage, where 'Manage' is highlighted with a red box. The main content area is titled 'PhoneVerification : /phoneverify/1.0.0'. It has two sections: 'Configurations' and 'Resources'. In 'Configurations', there are fields for 'Make this default version' (checkbox checked), 'Tier Availability' (set to 'Bronze'), 'Transports' (HTTP and HTTPS checked), 'Sequences' (checkbox checked for custom sequences), and 'Response Caching' (disabled). In 'Resources', there's a button to 'Add Scopes' and a list entry for '/path1' with a 'GET' method, 'path1/resource' URL, and 'Application & Application User' scope. The 'Bronze' tier is also highlighted with a red box.

The default throttling levels are Gold, bronze, silver and unlimited, as explained in the previous sections.

When a subscriber views an API using the **API Store**, s/he can see the resource-level throttling tiers using the **Throttle Info** tab as follows:

The screenshot shows the API Store interface for the 'PhoneVerification - 1.0.0' API. At the top, it shows the API details: admin, Rating (N/A), Version (1.0.0), Status (PUBLISHED), and Updated (23/May/2014 11:04:57 AM IST). To the right, there are sections for Applications (DefaultApplication) and Tiers (Bronze). Below these, it says 'Allows 1 request(s) per minute.' A 'Subscribe' button is available. The bottom navigation tabs include Overview, Documentation, API Console, Throttling Info (which is highlighted with a red box), and Forum. A table in the 'Throttling Info' section lists the URL Prefix (/phoneverify/1.0.0), URL Pattern (/path1/resource), and Throttling Limit (GET Allows unlimited requests).

Subscribers are not allowed to change these throttling tiers. They are simply notified of the limitations.

## IP-level throttling

In IP-based throttling, you can limit the number of requests sent by a client IP (e.g., 10 calls from single client).

1. Log in to the management console and click the **Resources -> Browse** menu.
2. Navigate to the `tiers.xml` file in the registry location `/_system/governance/apimgt/applicationdata`.

The screenshot shows the 'Browse' interface of the WSO2 API Manager. At the top, there is a 'Location' input field containing '/'. Below it are two buttons: 'Tree view' (highlighted in red) and 'Detail view'. The main area displays a hierarchical tree structure under 'Root /'. The nodes are as follows:

- /
- \_system
- config
- governance
- apimgt
  - applicationdata
    - api-docs
    - provider
      - sign-up-config.xml
      - tiers.xml (highlighted with a red box)
    - workflow-extensions.xml

3. Add your policy. For example, the throttling policy shown below allows only 1 API call per minute for a client from 10.1.1.1 and 2 calls per minute for a client from any other IP address.

```

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle">
  <throttle:MediatorThrottleAssertion>
    <wsp:Policy>
      <throttle:ID throttle:type="IP">10.1.1.1</throttle:ID>
      <wsp:Policy>
        <throttle:Control>
          <wsp:Policy>
            <throttle:MaximumCount>1</throttle:MaximumCount>

            <throttle:UnitTime>60000</throttle:UnitTime>
          </wsp:Policy>
        </throttle:Control>
      </wsp:Policy>
    </wsp:Policy>

    <wsp:Policy>
      <throttle:ID throttle:type="IP">other</throttle:ID>
      <wsp:Policy>
        <throttle:Control>
          <wsp:Policy>
            <throttle:MaximumCount>2</throttle:MaximumCount>
            <throttle:UnitTime>60000</throttle:UnitTime>
          </wsp:Policy>
        </throttle:Control>
      </wsp:Policy>
    </wsp:Policy>
  </throttle:MediatorThrottleAssertion></wsp:Policy>

```

## API visibility and subscription

### *API visibility*

Visibility settings prevent certain user roles from viewing and modifying APIs created by another user role.

- **Public:** The API is visible to all users (registered and anonymous), and can be advertised in multiple stores (central and non-WSO2 stores).
- **Restricted by Roles:** The API is visible to its tenant domain and only to the user roles that you specify.
- **Visible to my domain:** The API is visible to all users who are registered to the API's tenant domain. This option is **available only in a multi-tenanted environment**. It's not applicable when there is only one active tenant in the system.

Given below is how visibility levels work for users in different roles:

- API creator and publisher roles can see all APIs in their tenant store even if you restrict access to them. This is because those roles have permission to view and edit all APIs in the API Publisher, and therefore, does not have to be restricted in the Store.
- Anonymous users can only see APIs that have visibility as Public.
- Registered users can see
  - public APIs of all tenant domains
  - all APIs in the registered user's tenant domain as long as the API is not restricted to a role that the user is assigned to

### *Subscription availability*

The subscription availability option has three values as follows. You can set subscription availability to an API

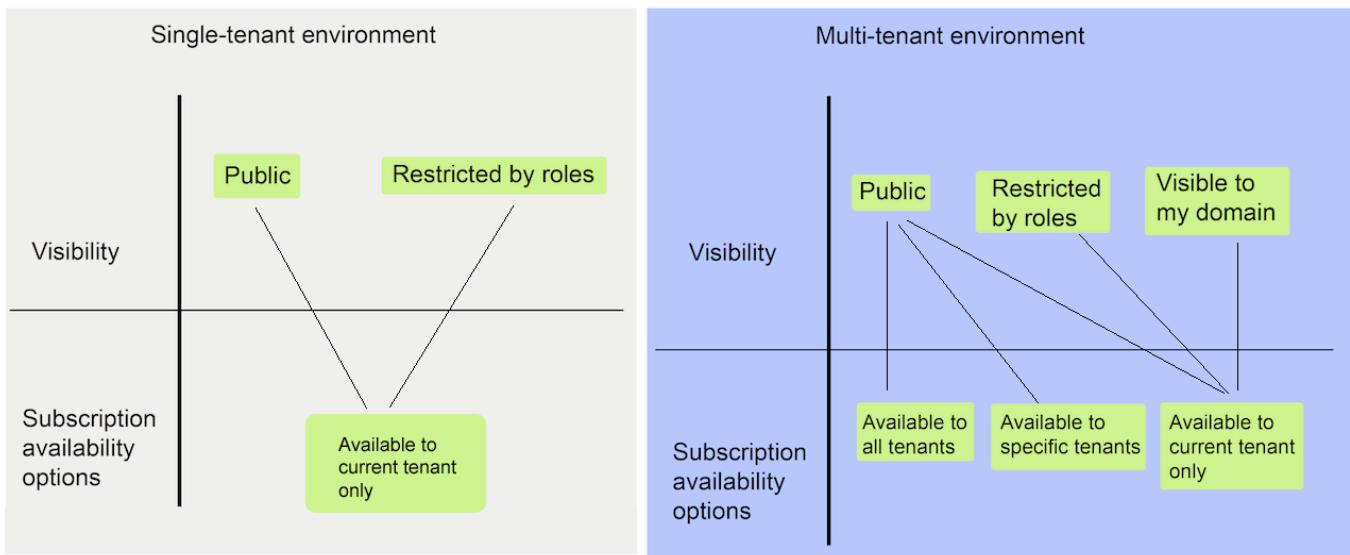
through the API Publisher's **Manage** tab.

- **Available to current Tenant Only:** only users in the current organization/tenant domain can subscribe to the API
- **Available to All the Tenants:** users of all organizations/tenant domains can subscribe to the API
- **Available to Specific Tenants:** users of the organizations/tenant domains that you specify as well as the current tenant domain can subscribe to the API

Subscription is available only to the current tenant in the following instances:

- When there is only one tenant in your system.
- Even if there are multiple tenants in your system, when you select **Visible to my domain** or **Restricted by roles** as the API's visibility in the previous step.

The diagram below depicts the relationship between the API's visibility and subscription availability:



#### API documentation visibility

By default, any document associated with an API has the same visibility level of the API. That is, if the API is public, its documentation is also visible to all users (registered and anonymous). To enable other visibility levels to the documentation, go to `<AM_HOME>/repository/conf/api-manager.xml` file, uncomment and set the following element to true:

```
<APIPublisher>
  ...
  <EnableAPIDocVisibilityLevels>true</EnableAPIDocVisibilityLevels>
</APIPublisher>
```

Then, log in to the API Publisher, go to the **Doc** tab and click **Add new Document** to see a new drop-down list added to select visibility from:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'APIs', 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', and 'Statistics'. The main area shows the 'PhoneVerify1 - 1.0.0' API. At the top, there are tabs for 'Overview', 'Lifecycle', 'Versions', 'Docs' (which is highlighted with a red box), and 'Users'. Below these tabs, there's a 'Add New Document' button. A form follows with fields for 'Name\*' (empty), 'Summary' (empty), 'Type' (radio buttons for 'How To' (selected), 'Samples & SDK', 'Public Forum', 'Support Forum', and 'Other (specify)'), and 'Source' (radio buttons for 'In-line' (selected), 'URL', and 'File'). Under 'Visibility', a dropdown menu is set to 'Visible to my domain' (highlighted with a red box). At the bottom are 'Add Document' and 'Cancel' buttons.

You set visibility in the following ways:

- **Same as API visibility:** Visible to the same user roles who can see the API. For example, if the API's visibility is public, its documentation is visible to all users.
- **Visible to my domain:** Visible to all registered users in the API's tenant domain.
- **Private:** Visible only to the users who have permission to log in to the API Publisher Web interface and create and/or publish APIs to the API Store.

## API resources

An API is made up of one or more resources, each of which handles a particular type of request. A resource has a set of methods that operate on it. The methods are analogous to a method or a function, and a resource is analogous to an object instance or a class in an object-oriented programming language. There are a few standard methods defined for a resource (corresponding to the standard HTTP GET, POST, PUT and DELETE methods.)

The diagram below shows a resource by the name `CheckPhoneNumber` added with four HTTP methods.

## API Definition

	URL Pattern	/phoneverify/1.0.0	Url Pattern Ex: path/to/resource
<input type="checkbox"/> GET			
<input type="checkbox"/> POST			
<input type="checkbox"/> PUT			
<input type="checkbox"/> DELETE			
<a href="#">+ Add</a>			
<b>GET</b>	<b>/CheckPhoneNumber</b>	+ Summary	
<b>POST</b>	<b>/CheckPhoneNumber</b>	+ Summary	
<b>PUT</b>	<b>/CheckPhoneNumber</b>	+ Summary	
<b>DELETE</b>	<b>/CheckPhoneNumber</b>	+ Summary	

When you add resources to an API, you define a URL pattern and **HTTP methods**. A resource can also have a list of **OAuth scopes**.

<b>URL Pattern</b>	<p>A URL pattern can be one of the following types:</p> <ul style="list-style-type: none"> <li>• As a url-mapping. E.g., /state/town/*</li> <li>• As a uri-template. E.g., /{state}/{town}</li> </ul> <p>The terms url-mapping and uri-template come from <a href="#">synapse configuration language</a>. When an API is published in the API Publisher, a corresponding XML definition is created in the API Gateway. This XML file has a dedicated section for defining resources. See examples below:</p> <pre style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">&lt;resource methods="POST GET" url-mapping="/state/town/*"&gt; &lt;resource methods="POST GET" uri-template="/{state}/{town}"&gt;</pre> <p>url-mapping performs a one-to-one mapping with the request URL, whereas the uri-template performs a pattern matching.</p> <p>Parametrizing the URL allows the API Manager to map the incoming requests to the defined resource templates based on the message content and request URI. Once a uri-template is matched, the parameters in the template are populated appropriately. As per the above example, a request made to <a href="http://gatewa_host:gateway_port/api/v1/texas/houston">http://gatewa_host:gateway_port/api/v1/texas/houston</a> sets the value of state to texas and the value of town to houston. You can use these parameters within the synapse configuration for various purposes and gain access to these property values through the <code>uri.var.province</code> and <code>uri.var.district</code> properties. For more information on how to use these properties, see <a href="#">Introduction to REST API</a> and the <a href="#">HTTP Endpoint</a> of the WSO2 ESB documentation.</p> <p>Also see <a href="http://tools.ietf.org/html/rfc6570">http://tools.ietf.org/html/rfc6570</a> on URI templates.</p>
--------------------	--

Once a request is accepted by a resource, it will be mediated through an in-sequence. Any response from the backend is handled through the out-sequence. Fault sequences are used to mediate errors that might occur in either sequence. The default in-sequence, out-sequence and fault sequences are generated when the API is published.

## HTTP methods

HTTP methods specify the desired action to be performed on an API's resource. You can select multiple methods from GET, POST, PUT, DELETE and OPTIONS. A method has attributes such as an OAuth scope, authentication type, response content type, parameters etc. as the diagram below shows:

## Resources

[+ Add Scopes](#)

<b>POST</b>	<a href="#">/CheckPhoneNumber</a> <small>+ Summary</small>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>Description :</b>				
<a href="#">+ Add Implementation Notes</a>				
<b>Response Content Type :</b>				
<b>Parameters :</b>				
<b>PUT</b>	<a href="#">/CheckPhoneNumber</a> <small>+ Summary</small>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>GET</b>	<a href="#">/CheckPhoneNumber</a> <small>+ Summary</small>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>DELETE</b>	<a href="#">/CheckPhoneNumber</a> <small>+ Summary</small>	Application & Application User	Unlimited	<a href="#">+ Scope</a>

The main attributes of a method are described below:

OAuth scopes	You can define a list of OAuth scopes to an API's resource and assign one of them to each HTTP method.
Authentication type	<p>The authentication type can be one of the following:</p> <ul style="list-style-type: none"> <li>• <b>None</b>: No authentication is applied and the API Gateway skips the authentication process</li> <li>• <b>Application</b>: Authentication is done by the application. The resource accepts application access tokens.</li> <li>• <b>Application User</b>: Authentication is done by the application user. The resource accepts user access tokens.</li> <li>• <b>Application and Application User</b>: Both application and application user level authentication is applied. Note that if you select this option in the UI, it appears as <b>Any</b> in the API Manager's internal data storage and data representation, and <b>Any</b> will appear in the response messages as well.</li> </ul> <p><b>Note</b> that for the resources that have HTTP verbs (GET, POST etc.) requiring authentication (i.e., Auth Type is not NONE), set <b>None</b> as the Auth type of <b>OPTIONS</b>. This is to support <b>CORS</b> (Cross Origin Resource Sharing) between the API Store and Gateway. (The above screenshot shows this).</p> <p>The auth type is cached in the API Manager for better performance. If you change the auth type through the UI, it takes about 15 minutes to refresh the cache. During that time, the server returns the old auth type from the cache. If you want the changes to be reflected immediately, please restart the server after changing the auth type.</p>
Response content type	You can use this attribute to document in what type the backend sends the response back to the API Manager. <b>Note that this attribute doesn't do any message type conversion</b> , but used simply as a way of letting the user know what type the response will be.
Parameters	Parameters of an HTTP method are analogous to arguments of a function in an object-oriented programming language. A resource's parameters are cached in the <b>resource cache</b> at the API Gateway.

## Cross-origin resource sharing

**Cross-origin resource sharing (CORS)** is a mechanism that allows restricted resources (e.g., fonts, JavaScript) of a Web page to be requested from another domain outside the domain from which the resource originated.

The Swagger API Console that is integrated in the API Manager runs as a JavaScript client in the API Store and makes calls from the Store to the API Gateway. Therefore, if you have the API Store and Gateway running on different ports, enable CORS between them.

The CORS configuration is in <APIM\_HOME>/repository/conf/api-manager.xml file. Given below is a sample code.

```
<CORSConfiguration>
    <Enabled>true</Enabled>

    <Access-Control-Allow-Origin>https://localhost:9443,http://localhost:9763</Access-Control-Allow-Origin>

    <Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type</Access-Control-Allow-Headers>
</CORSConfiguration>
```

The elements are described below:

XML Elements	Values	Description
<Enabled>	True/False	Used to enable/d CORS headers by default, CORS is needed for Swagger to work properly.
<Access-Control-Allow-Origin>	HTTP and HTTPS Store Address. Change the Host and Port for correct values of your store. For example, https://localhost:9443,http://localhost:9763	The value of the -Allow-Origin values are API Store required for Swagger to work properly.
<Access-Control-Allow-Headers>	Header values you need to pass when invoking the API. For example, authorization, Access-Control-Allow-Origin, Content-Type	Default values are used by Swagger to work properly.

Change your code according to the sample given here.

If you try to invoke an API with inline endpoints, you add the CORS Handler in the <handlers> section of the API's configuration as follows. Find the API's configuration in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/api folder.

```
<handlers>
    <handler
        class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler" />
</handlers>
```

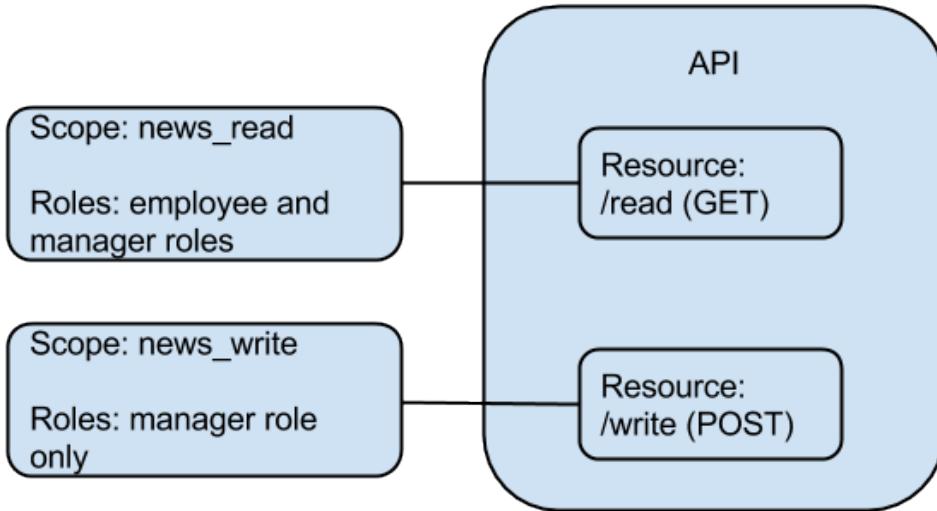
## OAuth scopes

Scopes enable fine-grained access control to API resources based on user roles. You define scopes to an API's

resources. When a user invokes the API, his/her OAuth 2 bearer token cannot grant access to any API resource beyond its associated scopes.

#### How scopes work

To illustrate the functionality of scopes, assume you have the following scopes attached to resources of an API:



Assume that users named **Tom** and **John** are assigned the employee role and both the employee and manager roles respectively.

Tom requests a token through the Token API as `grant_type=password&username=tom&password=xxxx&scope=news_read news_write`. However, as Tom is not in the manager role, he will only be granted a token bearing the `news_read` scope. The response from the Token API will be similar to the following:

```

{
  "scope": "news_read",
  "token_type": "bearer",
  "expires_in": 3299,
  "refresh_token": "8579facb65d1d3eba74a395a2e78dd6",
  "access_token": "eb51eff0b4d85cdaleb1d312c5b6a3b8"
}
  
```

Next, John requests a token as `grant_type=password&username=john&password=john123&scope=news_read news_write`. As john has both roles assigned, the token will bear both the requested scopes and the response will be similar to the following:

```

{
  "scope": "news_read news_write",
  "token_type": "bearer",
  "expires_in": 3299,
  "refresh_token": "4ca244fb321bd555bd3d555df39315",
  "access_token": "42a377a0101877d1d9e29c5f30857e"
}
  
```

This means that Tom can only access the GET operation of the API while John can access both as he is assigned to both the employee and manager roles. If Tom tries to access the POST operation, there will be an HTTP 403 Forbidden error as follows:

```

<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
    <ams:code>900910</ams:code>
    <ams:message>The access token does not allow you to access the requested resource</ams:message>
    <ams:description>Access failure for API: /orgnews, version: 1.0.0 with key: eb51eff0b4d85cdal1eb1d312c5b6a3b8</ams:description>
</ams:fault>

```

### Applying a scope

You apply scopes to an API resource at the time the API is created or modified. In the API Publisher, click the **API -> Add** menu (to add a new API) or the **Edit** link next to an existing API. Then, navigate to the **Manage** tab and scroll down to see the **Add Scopes** button under **Resources**. A screen such as the following appears:

The screenshot shows a modal dialog titled "Define Scope". It contains the following fields:

- Scope Key:** news\_read
- Scope Name:** Read News
- Roles:** employee, manager
- Description:** Eg: This scope will group all the administration APIs

At the bottom right of the dialog are two buttons: "Close" and "Add Scope".

<b>Scope Key</b>	A unique key for identifying the scope. Typically, it is prefixed by part of the API's name for uniqueness, but is not necessarily reader-friendly.
<b>Scope Name</b>	A human-readable name for the scope. It typically says what the scope does.
<b>Roles</b>	The user role(s) that are allowed to obtain a token against this scope. E.g., manager, employee.

**Tip:** When you generate **access tokens** to APIs protected by scope/s, a **Select Scopes** button is displayed in the **My Subscriptions** page for you to select the scope/s first and then generate the token to it.

### Scope whitelisting

A scope is not always used for controlling access to a resource. You can also use it to simply mark an access token. There are scopes that cannot be associated to roles (e.g., openid, device\_). Such scopes do not have to have roles associated with them. Skipping role validation for scopes is called scope whitelisting.

If you do not want a role validation for a scope in an API's request, add the scope under the `APIKeyValidation` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and restart the server. It will be whitelisted. For example,

```
^device_.*
some_random_scope
```

Next, invoke the Token API to get a token for the scope that you just whitelisted. For example,

```
curl -k -d "grant_type=password&username=admin&password=admin&scope=some_random_scope"
-H "Authorization: Basic
WmRFUFBvZmZwYVFnR25ScG5iZldtcUtSS3IwYTpSaG5ocEVJYUVCMEN3T1FReWpiZTJwaDBzc1Vh,
Content-Type: application/x-www-form-urlencoded" https://10.100.0.3:8243/token
```

Note that the issued token has the scope you requested. You get the token without any role validation as the scope is whitelisted.

```
{ "scope": "some_random_scope", "token_type": "bearer", "expires_in": 3600, "refresh_token": "59e6676db0addca46e68991e44f2b8b8", "access_token": "48855d444db883171c347fa21ba77e8" }
```

## API templates

An API template is its XML representation, which is saved in <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file. This file comes with the API Manager by default. You can edit this default template to change the synapse configuration of all APIs that are created.

If you are using a distributed API Manager setup (i.e., Publisher, Store, Gateway and Key Manager components are running on separate JVMs), edit the template in the Publisher node.

## Endpoints

An endpoint is a specific destination for a message such as an address, WSDL, a failover group, a load-balance group etc.

WSO2 API Manager has support for a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends. It supports [HTTP endpoints](#), [URL endpoints](#) (also termed as address endpoint), [WSDL endpoints](#), [Failover endpoints](#), [Load-balanced endpoints](#). For more information about endpoints and how to add, edit or delete them, see the [WSO2 ESB documentation](#).

Note the following:

- You can expose both REST and SOAP services to consumers through APIs.
- You cannot call backend services secured with OAuth through APIs created in the API Publisher. At the moment, you can call only services secured with username/password.
- The system reads gateway endpoints from the <JAVA\_HOME>/Repository/conf/api-manager.xml file. When there are multiple gateway environments defined, it picks the gateway endpoint of the production environment. You can define both HTTP and HTTPS gateway endpoints as follows:

```
<GatewayEndpoint>http://${carbon.local.ip}:${http.nio.port},https://${carbon.local.ip}:${https.nio.port}</GatewayEndpoint>
```

- If both types of endpoints are defined, the HTTPS endpoint will be picked as the server endpoint.

 **Tip:** When you define secure (HTTPS) endpoints, set the <parameter name="HostnameVerifier"> element to AllowAll in <APIM\_HOME>/repository/conf/axis2/axis2.xml file's HTTPS transport sender configuration:

```
<parameter name="HostnameVerifier">AllowAll</parameter>
```

If not, the server throws an exception.

- When creating (or updating) Failover endpoints through the Publisher UI (in the Implement tab), you need to go into the **Advanced Options** of each endpoint and specify a set of Error Codes for the endpoint to fail over on and take off the Initial Duration by setting its value to -1.

The screenshot shows a modal dialog titled "Advanced Endpoint Configuration". Under the "Endpoint Suspend State" section, there is a dropdown menu for "Error Codes" containing "101508, 101503, 101504". Below it is a note: "A list of error codes. If these error codes are received from the endpoint, the endpoint will be suspended." Another section shows "Initial Duration (ms)" set to "-1". A note below says: "The duration that the endpoint is suspended for the first time after the receiving the suspend error codes."

## Sequences

The API Manager has a default mediation flow that is executed in each API invocation. There are 3 default sequences engaged as `in`, `out` and `fault`.

## Caching

For information on configuring caching response messages and caching API calls at the Gateway and Key Manager server, see [Configuring Caching](#).

## Tutorials

This section covers the following usecases of the product:

- [Developer Portal](#)
- [API Consumer](#)
- [Security](#)

## Developer Portal

See the following topics:

- [Create and Publish an API](#)
- [Create an API with an Inline Script](#)
- [Edit an API Using the Swagger UI](#)
- [Add API Documentation](#)
- [Manage the API Lifecycle](#)
- [Publish to Multiple External API Stores](#)
- [Engage a new Throttling Policy](#)
- [Change the Default Mediation Flow of API Requests](#)
- [Map the Parameters of your Backend URLs with the API Publisher URLs](#)
- [Convert a JSON Message to SOAP and SOAP to JSON](#)
- [Publish through Multiple API Gateways](#)
- [Enforce Throttling and Resource Access Policies](#)
- [Include Additional Headers in the API Console](#)

## Create and Publish an API

**API creation** is the process of linking an existing backend API implementation to the API Publisher so that you can manage and monitor the API's lifecycle, documentation, security, community and subscriptions. Alternatively, you can provide the API implementation in-line in the API Publisher itself.



Click the following topics for a description of the concepts that you need to know when creating an API:

- [API visibility](#)
- [Resources](#)

- Endpoints
- Throttling tiers
- Sequences
- Response caching

The steps below show how to create a new API.

1. Log in to the API Publisher.
2. Select the option to design a new API and click **Start Creating.**

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'APIs', 'Browse', 'Add' (which is highlighted with a red box), 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions' (which is also highlighted with a red box), and 'Tier Permissions'. The main area has a title 'APIs / Add New API' and a heading 'Lets get started!'. It lists three options: 'I have an Existing API', 'I have a SOAP Endpoint', and 'Design new API' (which is selected and highlighted with a red box). A blue button labeled 'Start Creating' is located at the bottom right of this section.

3. Give the information in the table below and click the **Add** button to add the resource.

Field		Sample value
Name		PhoneVerification
Context		/phoneverify
<p><b>Tip:</b> You can define the API's version as a parameter of its context by adding the {version} into the context. For example, {version}/phon verify. The API Manager assigns the actual version of the API to the { version} parameter internally. For example, <a href="https://localhost:8243/1.0.0/phoneverify">https://localhost:8243/1.0.0/phoneverify</a>. Note that the version appears before the context, allowing you to group your APIs according to versions.</p>		
Version		1.0.0
Visibility		Public
Resources	URL pattern	CheckPhoneNumber
	Request types	GET, POST

**General Details**

Name:  ?

Context:  ?

Version:   
E.g.: v1.0.0 ,v1.0 , 1.0.0, 1.0

Visibility:  ?

Thumbnail Image:  No file chosen  Dimensions (max): 100 x 100 pixels

Description:

Tags:  ?  
Type a tag and Enter

**API Definition**

URL Pattern:  CheckPhoneNumber

GET  POST  PUT  DELETE more

4. After the resource is added, expand its GET method, add the following parameters. You use them to invoke the API using our integrated API Console in later tutorials. Once done, click **Implement**.

Parameter Name	Description	Parameter Type	Data Type	Required
PhoneNumber	Give the phone number to be validated	Query	String	True
LicenseKey	Give the license key as 0 for testing purpose	Query	String	True

**API Definition**

URL Pattern: /phoneverify/1.0.0 | Url Pattern Ex: path/to/resource

GET     POST     PUT     DELETE    more

**+ Add**

<b>POST</b>	/CheckPhoneNumber	+ Summary	
<b>GET</b>	/CheckPhoneNumber	+ Summary	

Description:

+ Add Implementation Notes

Response Content Type: application/json

Parameters:

Parameter Name	Description	Parameter Type	Data Type	Required
PhoneNumber	Give the phone number to be validated	query	string	True
LicenseKey	Give the license key as 0 for testing purpose	query	string	True

Parameter Name **+ Add Parameter**

**Save** **Next: Implement >**

5. Click the Managed API option.

**PhoneVerification(1.0.0) : /phoneverify/1.0.0**

**Managed API**  
Provide endpoints of the production API and sandbox API to be managed.

**Prototype**  
Use the inbuilt javascript engine to prototype the API or provide an endpoint to a prototype API. Inbuilt javascript engine do not have support to prototype SOAP APIs

6. The Implement tab opens. Give the information in the table below.

Field	Sample value
Endpoint type	HTTP endpoint

Production endpoint	This sample service has two operations as CheckPhoneNumber and CheckPhoneNumbe rs. Let's use CheckPhoneNumber here. <a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a> To verify the URL, click the <b>Test</b> button next to it.
---------------------	--

The screenshot shows the 'Implement' tab of the API configuration interface. At the top, there are three tabs: 'Design' (step 1), 'Implement' (step 2, highlighted with a red box), and 'Manage' (step 3). Below the tabs, the title 'PhoneVerification : /phoneverify/1.0.0' is displayed. A section for 'Implementation Method' includes radio buttons for 'Backend Endpoint' (selected) and 'Specify Inline'. The main area is titled 'Endpoints'. It shows two sections: 'Production Endpoint' and 'Sandbox Endpoint'. Both sections have an 'Endpoint Type' dropdown set to 'HTTP Endpoint'. The 'Production Endpoint' field contains the value 'im/phoneverify/phoneverify.asmx'. Below these fields are examples: 'E.g.: http://appserver/resource'. At the bottom of the endpoint section are buttons for 'Save', 'Deploy Prototype', 'Manage', and 'Cancel'.

7. Click **Manage** to go to the Manage tab and give the information in the table below.

Field	Sample value	Description
Tier Availability	Select all	The API can be available at different level of service. They allow you to limit the number of successful hits to an API during a given period of time.

**1 Design**    **2 Implement**    **3 Manage**

## Manage API: PhoneVerification : /phoneverify/1.0.0/1.0.0

### Configurations

Make this default version  [?](#)  
No default version defined for the current API

Tier Availability: **4 selected** [?](#)

Transports:  HTTPS  HTTP [?](#)

Sequences:  Check to select a custom sequence to be executed in the message flow

Response Caching: **Disabled** [?](#)

### Gateway Environments

### Business Information

### Resources

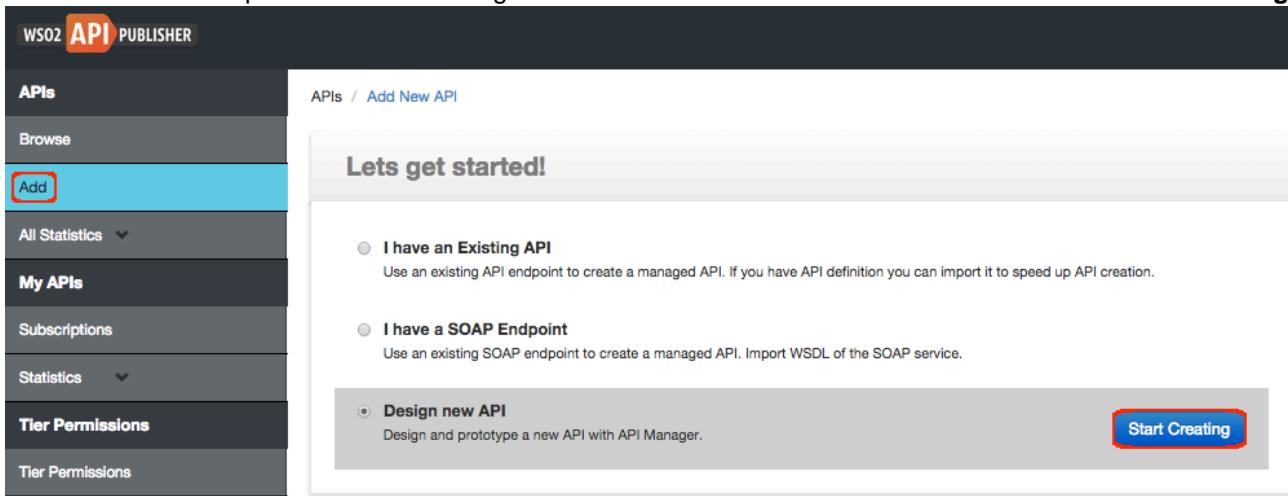
8. Click **Save & Publish**. This will publish the API that you just created in the API Store so that subscribers can use it.

You have created an API.

#### Create an API with an Inline Script

In this tutorial, you create an API with an inline script, deploy it as a prototype and invoke it using the API Console integrated in the API Store. You create APIs with inline scripts typically for testing purpose.

1. Log in to the API Publisher.
2. Select the option to design a new API and click **Start Creating**.



3. Give the information in the table below. To add resources, click the **Add** button.

Field		Sample value
Name		Location_API
Context		/location
Version		1.0.0
Resources	URL pattern	{town}
	Request types	GET

The screenshot shows the 'Design API' interface in WSO2 API Publisher. The left sidebar has a red box around the 'Add' button under 'APIs'. The main area has three tabs at the top: 'Design' (highlighted with a red box), 'Implement', and 'Manage'. Below is a 'General Details' section with the following fields:

- Name: \* Location\_API
- Context: \* /location
- Version: \* 1.0.0
- Visibility: Public
- Thumbnail Image: Choose File (No file chosen) - Dimensions (max): 100 x 100 pixels
- Description: (empty text area)
- Tags: Add tags

4. After the resource is added, expand its GET method and note that a parameter by the name town is added under the resource. You use it to pass the payload to the backend. Once done, click **Implement**.

The screenshot shows the 'Prototype' section for a GET method. At the top, it shows the URL path: **GET** /{town} + Summary. Below is a 'Description' section with a link to '+ Add Implementation Notes'. Under 'Response Content Type : application/json', there's a 'Parameters' table:

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
town	+ Empty	path	string	True	

At the bottom, there are buttons for 'Save' and 'Next: Implement >' (highlighted with a red box).

5. In the **Prototype** sections under the **Implement** tab, click the implementation method as **Inline**.

1 Design

2 Implement

3 Manage

**Weather\_API(1.0.0) : /weather/1.0.0****Managed API**

Provide endpoints of the production API and sandbox API to be managed.

**Prototype**

Use the inbuilt JavaScript engine to prototype the API or provide an endpoint to a prototype API. Inbuilt JavaScript engine does not have support to prototype SOAP APIs

 Implementation Method     Inline     Endpoint

6. Expand the GET method and give the following as the script. It reads the payload that the user sends with the API request and returns it as a JSON value.

```
mc.setProperty('CONTENT_TYPE', 'application/json');
var town = mc.getProperty('uri.var.town');
mc.setPayloadJSON('{"Town" : "'+town+'"}');
```

**GET****/{town}****+ Summary****Description :**[+ Add Implementation Notes](#)**Response Content Type :****Parameters :**

Parameter Name	Description	Parameter Type	Data Type	Required
town	<a href="#">+ Empty</a>	path	string	True

**Script :**

```
1 mc.setProperty('CONTENT_TYPE', 'application/json');
2 var town = mc.getProperty('uri.var.town');
3 mc.setPayloadJSON('{"Town" : "'+town+'"}');
4 |
```

7. Click the **Deploy as a Prototype** button.

8. Go to the API Store, click the **Prototyped APIs** menu and note that the newly deployed API is listed there.



**Tip:** You can invoke prototyped APIs without signing in to the API Store or subscribing to the API. The purpose of a prototype is advertising and giving an early implementation for users to test.

The screenshot shows the WSO2 API Manager dashboard. At the top, there is a navigation bar with links for 'API STORE', 'APIs', 'Prototyped APIs' (which is highlighted with a red box), 'My Applications', 'My Subscriptions', 'Forum', and 'Statistics'. Below the navigation bar is a search bar labeled 'Search API' with a magnifying glass icon. The main content area has two sections: 'Recently Added' on the left and 'Prototyped APIs' on the right. The 'Prototyped APIs' section displays a blue square icon with three white gears and the text 'Location\_API' below it, followed by '1.0.0 admin'.

9. Click the API to open it and go to its API **Console** tab.

The screenshot shows the WSO2 API Manager interface with the 'Prototyped APIs' tab highlighted in the top navigation bar. Below the navigation bar is a search bar labeled 'Search API' with a magnifying glass icon. The main content area displays the 'Location\_API - 1.0.0' page, which includes a profile picture for 'admin', the version '1.0.0', the status 'PROTOTYPED', and the update date '26/Jun/2015 14:41:28 PM IST'. At the bottom, there are tabs for 'Overview', 'Documentation', 'API Console' (which is highlighted with a red box), and 'Throttling Info'.

10. Expand the GET method, give any value for the town (say London) and invoke the API.

The screenshot shows the WSO2 API Manager's integrated Swagger UI. At the top, a blue bar indicates a **GET** method and the endpoint **/{{town}}**. Below this, under the **Parameters** section, there is a table with two columns: **Parameter** and **Value**. A row for **town** has its value, **London**, highlighted with a red box. To the right of the table, the **Description**, **Parameter Type**, and **Data Type** are listed as path and string respectively. Under the **Response Messages** section, a table lists **HTTP Status Code**, **Reason**, and **Response Model** for status code **200**. A red box highlights the **Try it out!** button at the bottom of this section.

11. Note the payload you gave as a JSON output in the response.

The screenshot shows the **Response Body** section of the Swagger UI. It displays a JSON object with one entry: `{ "Town": "London" }`.

You have created an API with inline script, deployed it as a prototype and invoked it through the integrated API Console.

## Edit an API Using the Swagger UI

WSO2 API Manager has an integrated Swagger UI, which is part of the Swagger project.

Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe a services in the same way that interfaces describe lower-level programming code.

The [Swagger UI](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation and more discoverability. The Swagger UI has JSON code and its UI facilitates easier code indentation, keyword highlighting and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI.

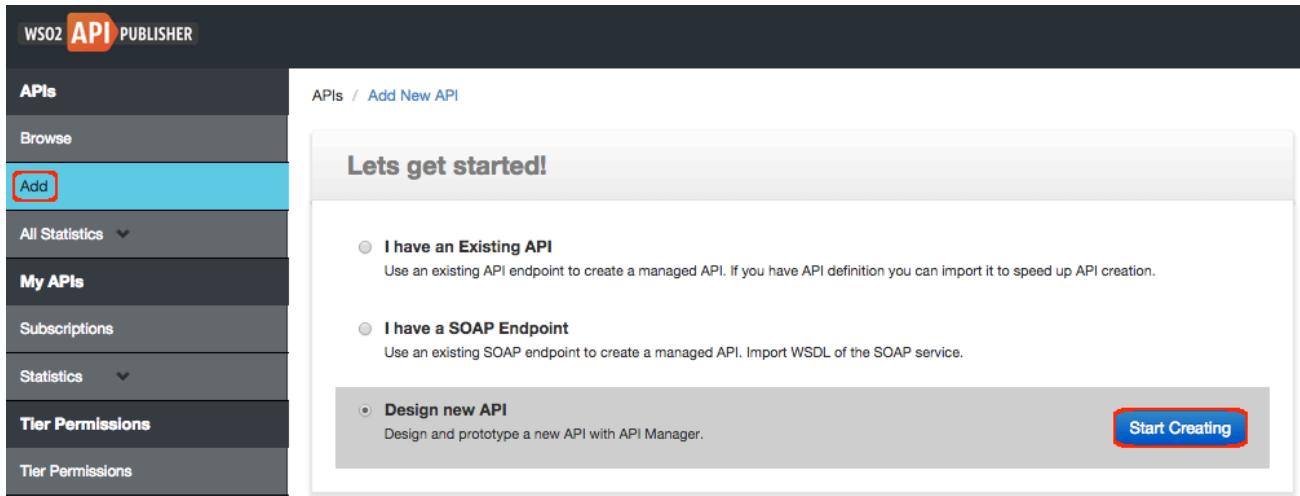
Also, see the [Swagger 2.0 specification](#).

In this tutorial, let's see how you can add interactive documentation to an API by directly editing the Swagger code and through the API Publisher UI.



This tutorial uses the PhoneVerification API created in Create and Publish an API.

1. Log in to the API Publisher and choose to design a new API.



2. In the **Design** tab, give an API name, a context and a version, and click the **Edit Source** button under the **AP  
I  
D  
e  
f  
i  
n  
i  
t  
i  
o  
n** section.

The screenshot shows the 'Design API' page. The sidebar on the left is identical to the previous one. The main area has a navigation bar with '1 Design' (highlighted with a red box), '2 Implement', and '3 Manage'. Below it is a section titled 'Design API' with 'General Details'. It includes fields for 'Name' (PhoneVerification2), 'Context' (/phoneverify2), 'Version' (1.0.0), 'Visibility' (Public), 'Thumbnail Image' (choose file), 'Description' (empty text area), and 'Tags' (add tags). At the bottom right of this section is a red box around the 'Edit Source' button. Below this is another section titled 'API Definition'.

3. The Swagger UI opens. Add the following code under the paths object and click **Save**. It adds a resource with two HTTP methods into the API.

**Tip:**

In the below code, note that you have a resource defined with the URL pattern /CheckPhoneNumber under the `paths` object. This is followed by the HTTP methods GET and POST. For each HTTP method, the following parameters are defined.

- **x-auth-type:** WSO2-specific object to define the authentication type of the method.
- **x-throttling-tier:** WSO2-specific object to define the throttling tier of the method.
- **responses:** An object to hold responses that can be used across operations. See the

Swagger specification for details.

```
/CheckPhoneNumber:
  get:
    x-auth-type: "Application & Application User"
    x-throttling-tier: Unlimited
    responses:
      "200": {}
  post:
    x-auth-type: "Application & Application User"
    x-throttling-tier: Unlimited
    responses:
      "200": {}
```

The screenshot shows the WSO2 API Publisher interface. On the left, there is a code editor window titled 'PhoneVerification' containing a Swagger 2.0 specification. The specification defines a resource '/CheckPhoneNumber' with both GET and POST methods. Both methods have the same authentication requirements ('x-auth-type: "Application & Application User"', 'x-throttling-tier: Unlimited') and return a '200' response. On the right, the 'PhoneVerification' API is listed under 'Paths'. It shows two methods: 'GET /CheckPhoneNumber' and 'POST /CheckPhoneNumber'. Each method has a 'Responses' section indicating a single '200' response.

```

1  swagger: "2.0"
2  paths:
3  /CheckPhoneNumber:
4  get:
5    x-auth-type: "Application & Application User"
6    x-throttling-tier: Unlimited
7    responses:
8      "200": {}
9  post:
10   x-auth-type: "Application & Application User"
11   x-throttling-tier: Unlimited
12   responses:
13     "200": {}
14  info:
15    title: PhoneVerification
16    version: 1.0.0
17

```

**PhoneVerification**  
Version 1.0.0  
Paths  
/CheckPhoneNumber  
GET /CheckPhoneNumber  
Responses  
Code Description  
200  
POST /CheckPhoneNumber  
Responses  
Code Description  
200

4. Back in the API Publisher, note that a resource with two HTTP methods are added to the API.

## API Definition

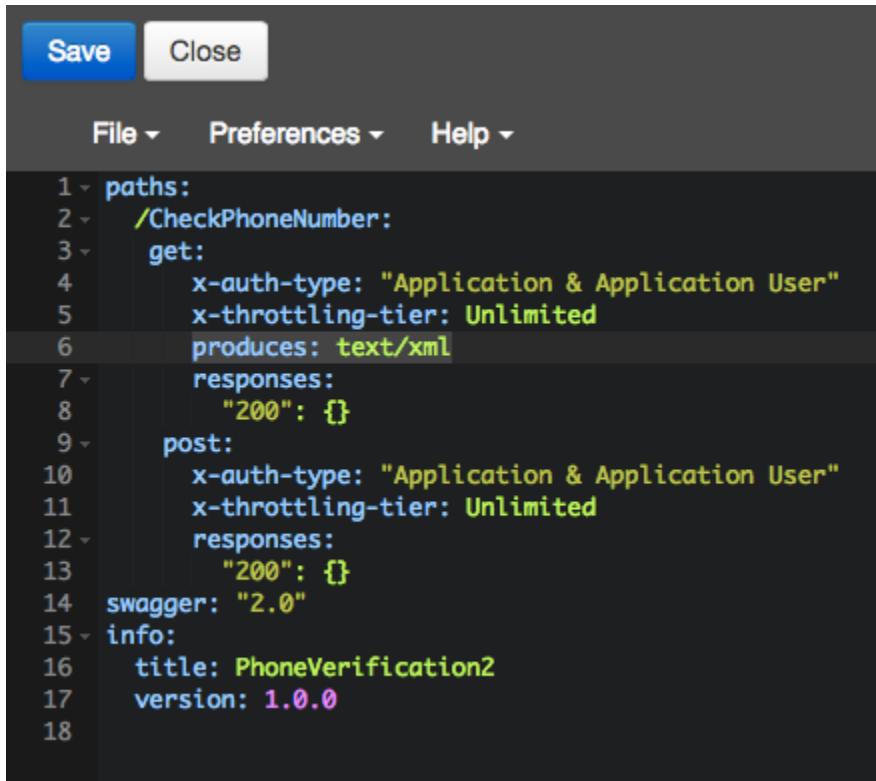
The screenshot shows the 'API Definition' section of the WSO2 API Publisher. It includes fields for 'URL Pattern' (/{context}/{version}/) and 'Method Selection' (checkboxes for GET, POST, PUT, DELETE). Below these, a table lists the methods for the '/CheckPhoneNumber' resource. The table has columns for 'Method', 'Path', and 'Summary'. Two rows are present: one for 'POST /CheckPhoneNumber' and one for 'GET /CheckPhoneNumber'. Both rows have a '+ Summary' link next to them. A red box highlights the 'GET /CheckPhoneNumber' row.

Method	Path	Summary
POST	/CheckPhoneNumber	+ Summary
GET	/CheckPhoneNumber	+ Summary

Let's say the backend of this API sends the response in XML format. Let's document this under the GET method in the resource that we just added.

5. Add the following code under the GET method and click **Save**.

```
produces: text/xml
```



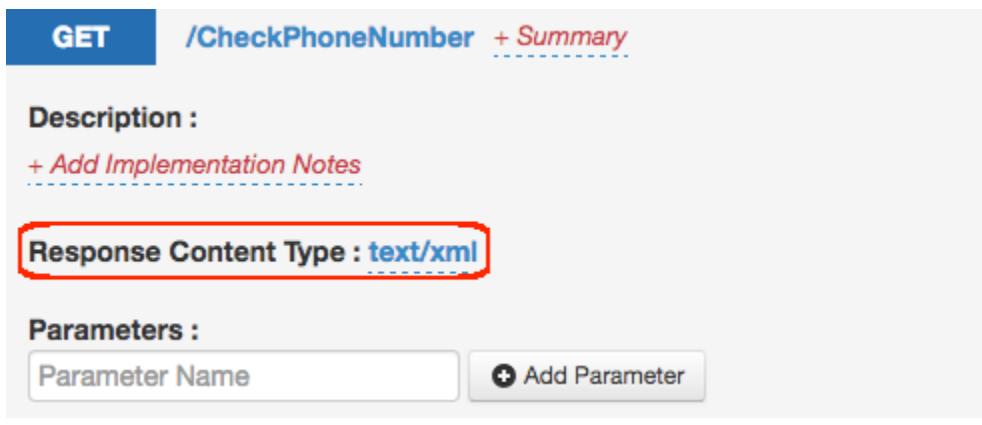
```

1  paths:
2    /CheckPhoneNumber:
3      get:
4        x-auth-type: "Application & Application User"
5        x-throttling-tier: Unlimited
6        produces: text/xml
7        responses:
8          "200": {}
9      post:
10     x-auth-type: "Application & Application User"
11     x-throttling-tier: Unlimited
12     responses:
13       "200": {}
14   swagger: "2.0"
15   info:
16     title: PhoneVerification2
17     version: 1.0.0
18

```

- Back in the API Publisher, note that the response content type has changed to XML.

 **Tip:** You can use this attribute to document the type of the response message that the backend sends. **It doesn't do any message type conversion.** You can add multiple values as a comma-separated list. E.g., produces: "text/xml, application/json"



**GET**    [/CheckPhoneNumber](#) + Summary

**Description :**  
[+ Add Implementation Notes](#)

**Response Content Type :** **text/xml**

**Parameters :**

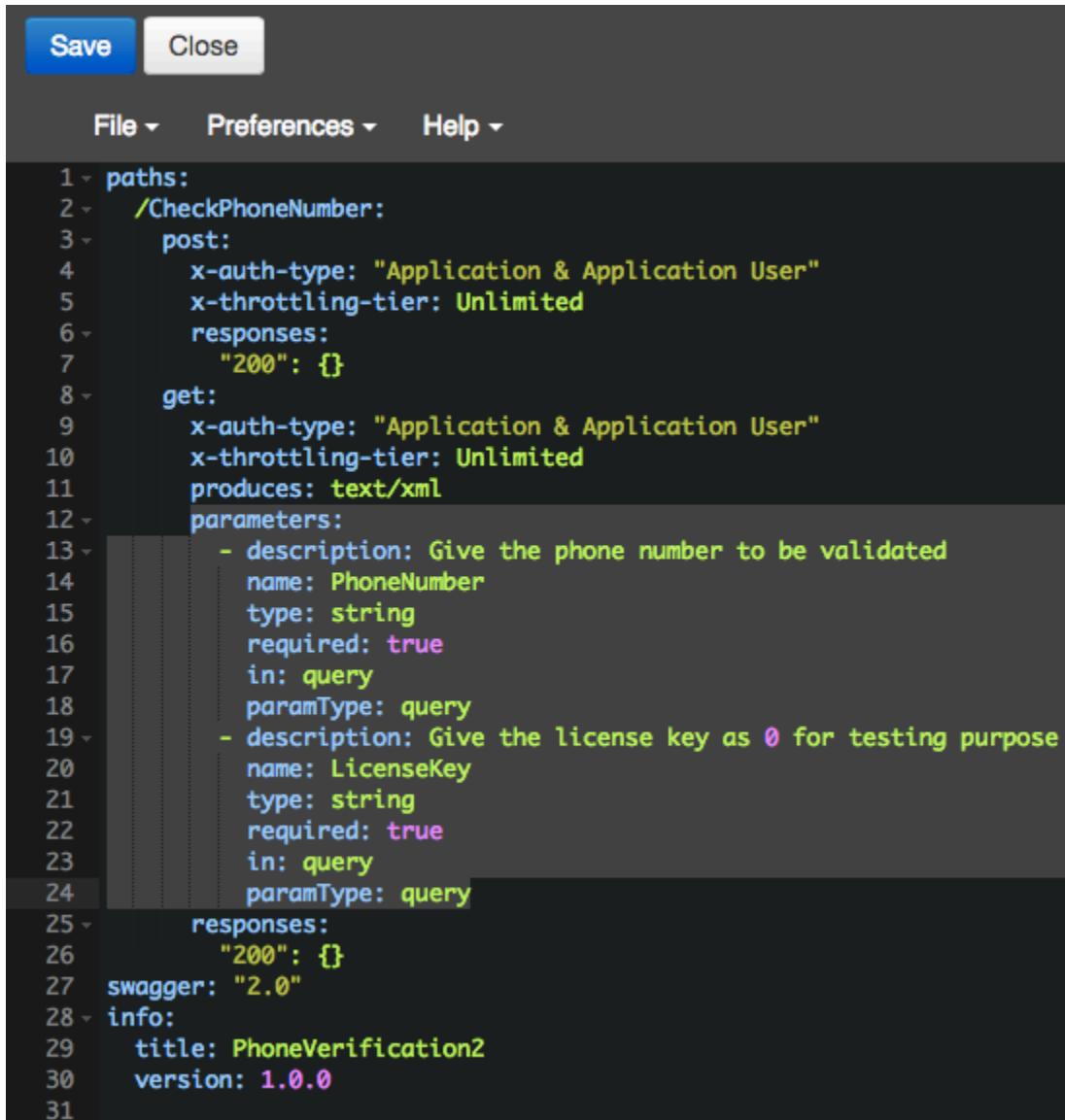
Parameter Name	<a href="#">+ Add Parameter</a>
----------------	---------------------------------

- Add the following code under the GET method and click **Save**. It adds two parameters to the method.

```

parameters:
  - description: Give the phone number to be validated
    name: PhoneNumber
    type: string
    required: true
    in: query
  - description: Give the license key as 0 for testing purpose
    name: LicenseKey
    type: string
    required: true
    in: query

```



The screenshot shows the API Publisher interface with the following configuration:

```

1 paths:
2   /CheckPhoneNumber:
3     post:
4       x-auth-type: "Application & Application User"
5       x-throttling-tier: Unlimited
6       responses:
7         "200": {}
8     get:
9       x-auth-type: "Application & Application User"
10      x-throttling-tier: Unlimited
11      produces: text/xml
12      parameters:
13        - description: Give the phone number to be validated
14          name: PhoneNumber
15          type: string
16          required: true
17          in: query
18          paramType: query
19        - description: Give the license key as 0 for testing purpose
20          name: LicenseKey
21          type: string
22          required: true
23          in: query
24          paramType: query
25        responses:
26          "200": {}
27      swagger: "2.0"
28      info:
29        title: PhoneVerification2
30        version: 1.0.0
31

```

The parameters section (lines 12-24) is highlighted with a gray background, matching the background of the code block above it.

- Back in the API Publisher, note the two parameters with their descriptions that are added under the GET method.

**GET** /CheckPhoneNumber [+ Summary](#) 

**Description :**

[+ Add Implementation Notes](#)

**Response Content Type : text/xml**

**Parameters :**

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
PhoneNumber	Give the phone number to be validated	query	string	True	
LicenseKey	Give the license key as 0 for testing purpose	query	string	True	

9. In the Swagger UI, add the following summary and description to the GET method and click **Save**.

```
summary: Check the validity of your phone number
description: "Phone Verification validates a telephone number and returns carrier
information, location routing etc."
```

**Save** **Close**

**File** ▾ **Preferences** ▾ **Help** ▾

```

1  paths:
2    /CheckPhoneNumber:
3      get:
4        x-auth-type: "Application & Application User"
5        x-throttling-tier: Unlimited
6        parameters:
7          - description: Give the phone number to be validated
8            name: PhoneNumber
9            type: string
10           required: true
11           in: query
12           paramType: query
13          - description: Give the license key as 0 for testing purpose
14            name: LicenseKey
15            type: string
16            required: true
17            in: query
18            paramType: query
19        produces: text/xml
20        responses:
21          "200": {}
22      summary: Check the validity of your phone number
23      description: "Phone Verification validates a telephone number and returns
carrier information, location routing etc."
24      post:
25        x-auth-type: "Application & Application User"
26        x-throttling-tier: Unlimited
27        responses:
28          "200": {}
29      swagger: "2.0"
30      info:
31        title: PhoneVerification2
32        version: 1.0.0
33

```

10. Back in the API Publisher, note the summary and description appearing under the GET method.

**GET** /CheckPhoneNumber Check the validity of your phone number Delete

**Description :**  
Phone Verification validates a telephone number and returns carrier information, location routing etc.

**Response Content Type :** [text/xml](#)

**Parameters :**

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
PhoneNumber	Give the phone number to be validated	query	string	True	<span style="border: 1px solid red; padding: 2px;">Delete</span>
LicenseKey	Give the license key as 0 for testing purpose	query	string	True	<span style="border: 1px solid red; padding: 2px;">Delete</span>

11. In the Swagger UI, add the following code under the POST method and click **Save**. It adds two parameters as `PhoneNumber` and `LicenseKey` to pass the payload in. It also changes the datatypes of the parameters to `application/x-www-form-urlencoded` as the backend expects that datatype.

```

parameters:
  - description: Give the phone number to be validated
    name: PhoneNumber
    required: true
    type: string
    in: formData
  - description: Give the license key as 0 for testing purpose
    name: LicenseKey
    required: true
    type: string
    in: formData
consumes: application/x-www-form-urlencoded
  
```

```

1 paths:
2   /CheckPhoneNumber:
3     post:
4       x-auth-type: "Application & Application User"
5       x-throttling-tier: Unlimited
6       parameters:
7         - description: Give the phone number to be validated
8           name: PhoneNumber
9           required: true
10          type: string
11          in: formData
12         - description: Give the license key as 0 for testing purpose
13           name: LicenseKey
14           required: true
15           type: string
16           in: formData
17       consumes:
18         - application/x-www-form-urlencoded
19       responses:
20         "200": {}
21     get:
22       x-auth-type: "Application & Application User"
23       summary: Check the validity of your phone number
24       x-throttling-tier: Unlimited
25       description: "Phone Verification validates a telephone number and returns carrier information, location routing etc."
26       produces:
27         - text/xml
28       parameters:

```

12. Back in the API Publisher, note the parameter that you added are visible under the POST method.

**POST** /CheckPhoneNumber [+ Summary](#)

Description :  
+ Add Implementation Notes

Response Content Type : application/json

Parameters :

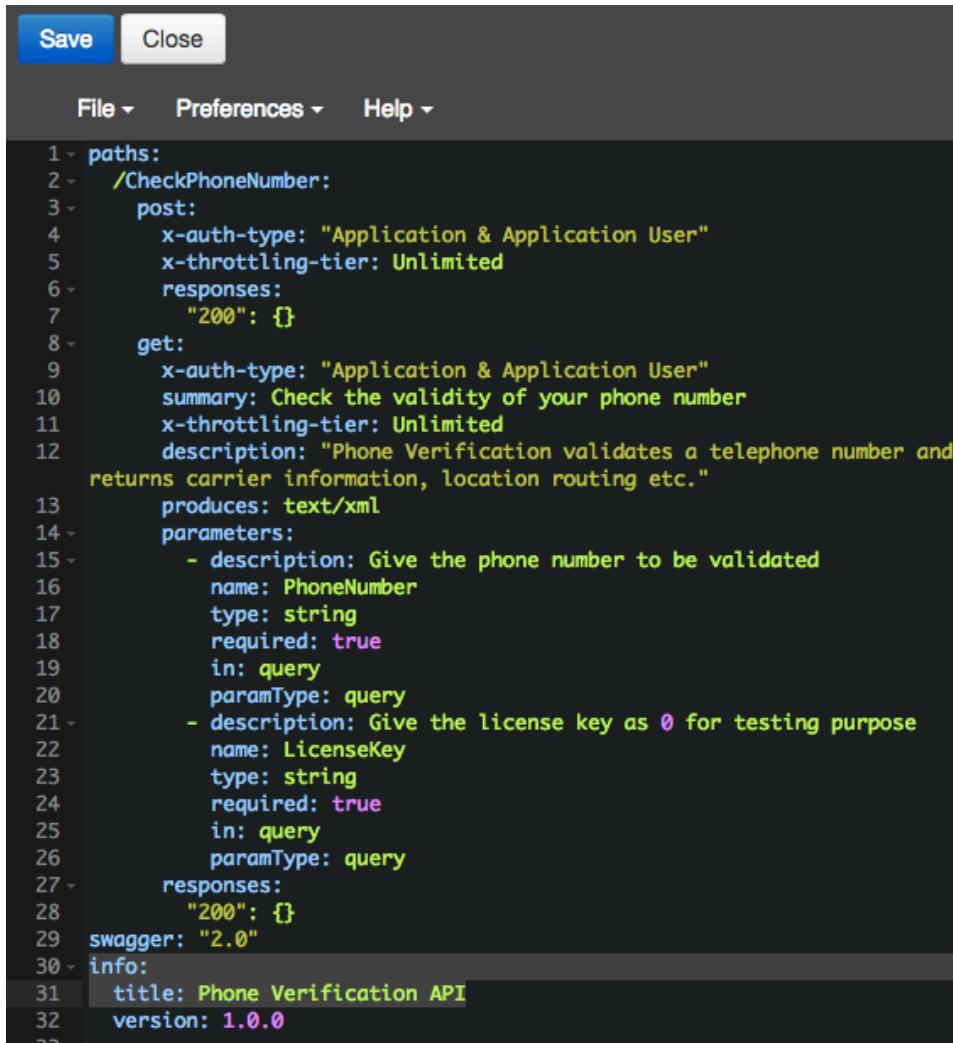
Parameter Name	Description	Parameter Type	Data Type	Required	Delete
PhoneNumber	Give the phone number to be validated	formData	string	True	
LicenseKey	Give the license key as 0 for testing purpose	formData	string	True	

13. In the Swagger API, change the title of the API as follows. This is the title that is visible to the consumers in the API Console of the API Store, after the API is published.

```

info:
  title: Phone Verification API

```



The screenshot shows the WSO2 API Manager's configuration interface. At the top, there are 'Save' and 'Close' buttons. Below them is a menu bar with 'File', 'Preferences', and 'Help'. The main area displays the API definition in a code editor:

```

1 paths:
2   /CheckPhoneNumber:
3     post:
4       x-auth-type: "Application & Application User"
5       x-throttling-tier: Unlimited
6       responses:
7         "200": {}
8     get:
9       x-auth-type: "Application & Application User"
10      summary: Check the validity of your phone number
11      x-throttling-tier: Unlimited
12      description: "Phone Verification validates a telephone number and returns carrier information, location routing etc."
13      produces: text/xml
14      parameters:
15        - description: Give the phone number to be validated
16          name: PhoneNumber
17          type: string
18          required: true
19          in: query
20          paramType: query
21        - description: Give the license key as 0 for testing purpose
22          name: LicenseKey
23          type: string
24          required: true
25          in: query
26          paramType: query
27      responses:
28        "200": {}
29    swagger: "2.0"
30    info:
31      title: Phone Verification API
32      version: 1.0.0

```

You will see how this change is reflected in the API Store in the last step of this tutorial.

14. Complete the rest of the API creation process. Once you are done, click **Save and Publish** on the **Manage** tab to publish the API to the API Store.



## Manage API: PhoneVerification2 : /phoneverify2/1.0.0

### Configurations

Make this default version  ⓘ No default version defined for the current API

Tier Availability\*: **4 selected** ⓘ

Transports\*:  HTTPS  HTTP ⓘ

Sequences:  Check to select a custom sequence to be executed in the message flow

Response Caching: **Disabled** ⓘ

### Gateway Environments»

### Business Information »

### Resources

**Add Scopes**

POST	/CheckPhoneNumber <a href="#">+ Summary</a>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
GET	/CheckPhoneNumber Check the validity of your phone number	Application & Application User	Unlimited	<a href="#">+ Scope</a>

**Save** **Save & Publish** **Cancel**

15. Log in to the API Store and click on the API that you just published.

The screenshot shows the WSO2 API Manager interface. At the top, there's a navigation bar with links for 'API STORE', 'APIs', 'Prototyped APIs', 'My Applications', 'My Subscriptions', 'Forum', and 'Statistics'. Below the navigation bar is a search bar labeled 'Search API' with a magnifying glass icon and an information icon. On the left, there's a 'Recently Added' section showing two new API entries: 'PhoneVerification-1.0.0' and 'PhoneVerification2-1.0.0', both created by 'admin' with five-star ratings. The main area is titled 'APIs' and lists 'PhoneVerification' and 'PhoneVerification2'. 'PhoneVerification2' is highlighted with a red box around its icon and details. Below the APIs, there's a note about Swagger and a link to 'Swagger (/swagger.json)'. The 'Phone Verification API' page is shown in detail, featuring a 'GET /CheckPhoneNumber' operation with a red box around it. The 'Implementation Notes' section contains a note about validating phone numbers. The 'Parameters' table has two rows: 'PhoneNumber' (required, query, string) and 'LicenseKey' (required, query, string). Both parameter input fields are also highlighted with red boxes. The 'Response Messages' section shows a single row for status code 200 with a 'Try it out!' button. A 'POST /CheckPhoneNumber' operation is also shown at the bottom.

In this tutorial, you have seen how the integrated Swagger UI can be used to design, describe and document your API, so that the API consumers get a better understanding of what the API's functionality is.

## Add API Documentation

This section covers the following:

- Add API Documentation In-line, using a URL or a File
- Add Apache Solr-Based Indexing

## Add API Documentation In-line, using a URL or a File

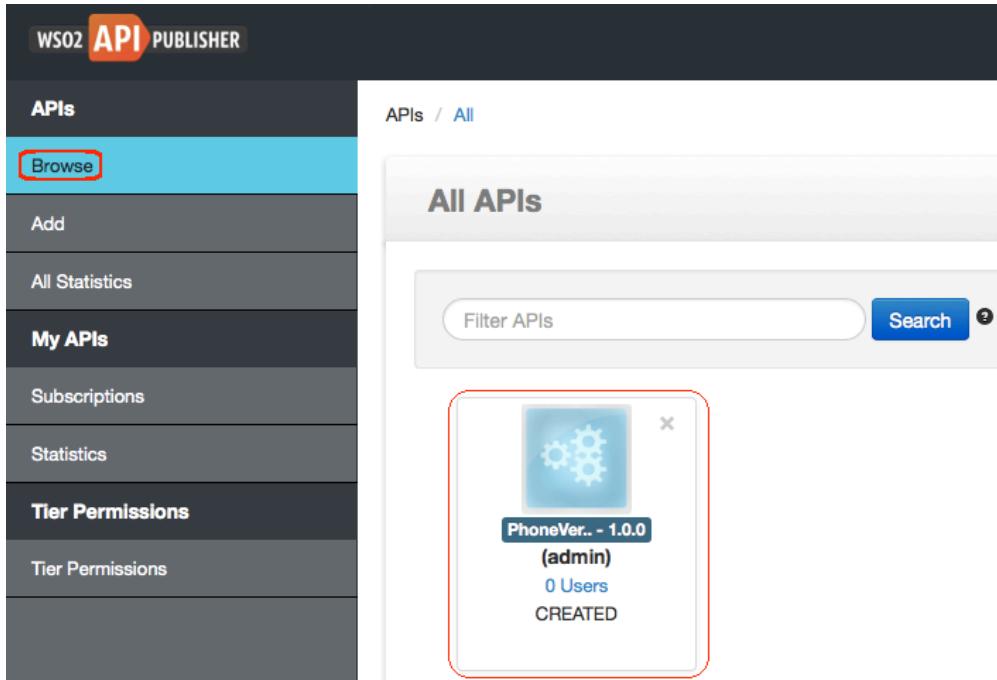
**API documentation** helps API subscribers understand the functionality of the API, and API publishers market their APIs better and sustain competition. Using the API Publisher, you can add different types of documentation from different sources. All documents created in the API Publisher have unique URLs to help improve SEO support.

The documentation types supported in the API Publisher are as follows:

- **In-line:** Hosts documentation (How-tos, Samples, SDK, forums etc.) in the API Publisher itself and allows it to be edited directly from the UI.
- **URL:** Links to file references (URLs) of an external configuration management system.
- **File:** Allows to upload the documentation directly to the server
- **Using the integrated API Console**

 **Tip:** Do you want to set different visibility levels to the API documentation than the API? See [API documentation visibility](#).

1. Log in to WSO2 API Publisher.
2. Click the API to which you want to add documentation to (e.g., PhoneVerification 1.0.0).



The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following menu items: APIs (selected), Browse (highlighted with a red box), Add, All Statistics, My APIs, Subscriptions, Statistics, Tier Permissions (selected), and Tier Permissions. The main area is titled "All APIs" and contains a search bar with "Filter APIs" and "Search" buttons. Below the search bar is a list of APIs. One API entry is highlighted with a red box: "PhoneVer.. - 1.0.0" by "(admin)" with "0 Users" and the status "CREATED".

3. Select the **Docs** tab of the API and click the **Add New Document** link.  
In-line documentation
4. Provide the following details to create a doc In-line.

Name	PhoneVerification
Type	How To
Source	In-line
Summary	Check the validity of a phone number

**PhoneVerification - 1.0.0** [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[+ Add New Document](#)

Name\*  
PhoneVerification

Summary  
Check the validity of a phone number

Type  
 How To  
 Samples & SDK  
 Public Forum  
 Support Forum  
 Other (specify)

Source  
 In-line [?](#)  
 URL [?](#)  
 File [?](#)

[Add Document](#) [Cancel](#)

5. Click the **Add Document** button.
6. After adding the document, click the **Edit Content** link associated with it.

**PhoneVerification - 1.0.0** [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[+ Add New Document](#)

Name	Type	Modified On	Actions
PhoneVerification	How To	5/19/2015, 2:27:47 PM	<a href="#">Edit Content</a> <a href="#">Update</a> <a href="#">Delete</a>

7. The embedded editor opens allowing you to edit the document's content in-line. Add in the content and click **Save** and **Close**.

# PhoneVerification

**Font Tools:**

Determine whether a phone number is wireless or a landline - in real time. Even if the phone number has been ported between service providers, Phone Verification will return the latest information.

Phone Verification validates the ten digits of a U.S. telephone number and returns carrier information as well as the Location Routing Number (LRN) for telephone numbers administered by the North American Numbering Plan Administration. LRN is a unique 10-digit number that represents a telephone switch through which multiple phone numbers are routed. The LRN enables Local Number Portability (LNP), which allows phone numbers to be ported to different carriers.

The benefits of using the Phone Verification API include:

- Determine if a phone number is valid
- Remove dashes, parenthesis and spaces
- Receive latest SMTP email string data
- Define time zone to restrict calling times
- Validate wireless vs landline data for compliance

**Save** **Save and Close** **Cancel**

8. The API's **Doc** tab opens. Click the **Add New Document** link again.
- Documentation using a URL
9. Then provide the following information to create another doc using a URL.

Name	CDYNE Wiki
Type	Other (Summary)
Source	URL <a href="http://api-portal.anypoint.mulesoft.com/cdyne/api/cdyne-phone-verification-api">http://api-portal.anypoint.mulesoft.com/cdyne/api/cdyne-phone-verification-api</a>
Summary	<b>CDYNE Phone Verification API</b>

**PhoneVerification - 1.0.0** [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name\*  
CDYNE Wiki

Summary  
CDYNE Phone Verification API

Type  
 How To  
 Samples & SDK  
 Public Forum  
 Support Forum  
 Other (specify)  
 \*

Source  
 In-line  
 URL  
 \*

/cdyne-phone-verification-api

File

Add Document Cancel

10. Click the **Add Document** button.
11. The API's **Doc** tab opens again. Click the **Add New Document** link again to add yet another document using a file.
- Documentation using a file
12. Enter the following information:

Name	API Manager Samples
Type	Samples & SDK
Source	You can provide any file format (common formats are PDF, HTML, .doc, text) of any size. For example, use the sample PDF file <a href="#">here</a> .

**PhoneVerification - 1.0.0** [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name*	<input type="text" value="API Manager Samples"/>
Summary	<input type="text"/>
Type	<input type="radio"/> How To <input checked="" type="radio"/> Samples & SDK <input type="radio"/> Public Forum <input type="radio"/> Support Forum <input type="radio"/> Other (specify)
Source	<input type="radio"/> In-line <input type="radio"/> URL <input checked="" type="radio"/> File * <input type="button" value="Browse..."/> WSO2 API Mana...

[Add Document](#) [Cancel](#)

13. After adding the details, click the **Add Document** button. You have now added three documents to the API: in-line, using a URL and a file.

**PhoneVerification - 1.0.0** [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name	Type	Modified On	Actions
CDYNE Wiki	Other	01/09/2014 14:16:45	<a href="#">View</a>   <a href="#">Update</a>   <a href="#">Delete</a>
API Manager Samples	Samples	01/09/2014 14:33:26	<a href="#">Open</a>   <a href="#">Update</a>   <a href="#">Delete</a>
SimpleClient	How To	01/09/2014 14:34:09	<a href="#">Edit Content</a>   <a href="#">Update</a>   <a href="#">Delete</a>

14. Log in to the API Store and click the PhoneVerification 1.0.0 API.

The screenshot shows the WSO2 API Manager dashboard. At the top, there's a navigation bar with 'WSO2 API STORE' on the left and three tabs on the right: 'APIs' (which is highlighted with a red box), 'Prototyped APIs', and 'My Applications'. Below the navigation bar is a search bar labeled 'Search API'. On the left, there's a 'Recently Added' section featuring a card for 'PhoneVerification-1.0.0' by 'admin' with a 5-star rating. On the right, there's a larger 'APIs' section also featuring the same 'PhoneVerification-1.0.0' card, which is also highlighted with a red box around its icon and title.

15. Go to the API's **Documentation** tab and see the documents listed by type. As a subscriber, you can read the doc and learn about the API.

The screenshot shows the 'PhoneVerification - 1.0.0' API documentation page. At the top, it displays basic information: 'Version: 1.0.0', 'Status: PUBLISHED', and 'Updated: 26/Aug/2014 22:27:21 PM IST'. Below this, there are five tabs: 'Overview', 'Documentation' (which is highlighted with a dashed border), 'API Console', 'Throttling Info', and 'Forum'. Under the 'Documentation' tab, there are three expandable sections: 'Other', 'Samples', and 'How To', each represented by a dark grey button with white text.

16. Expand the categories and click the **View Content** or **Download** links to see the documentation content.

Overview    **Documentation**    API Console    Throttling

Other

## Summary

Name:

CDYNE Wiki

Summary:

CDYNE Phone Verification API

[View Content](#)

Samples

Name:

API Manager Samples

[Download](#)

How To

You have created documentation using the API Publisher and viewed them as a subscriber in the API Store.

### Add Apache Solr-Based Indexing

The API Manager has [Apache Solr](#) based indexing for API documentation content. It provides both the API Publisher and Store full-text search facility to search through API documentation, find documents and related APIs. The search syntax is **doc:keyword**. Search criteria looks for the keyword in any word/phrase in the documentation content and returns both the matching documents and associated APIs.

The following media types have Apache Solr based indexers by default, configured using the `<Indexers>` element in `<APIM_HOME>/repository/conf/registry.xml`.

- Text : text/plain
- PDF : application/pdf
- MS word : application/msword
- MS Powerpoint : application/vnd.ms-powerpoint
- MS Excel : application/vnd.ms-excel
- XML : application/xml

### ***Writing a custom index***

In addition to the default ones, you can write your own indexer implementation and register it as follows:

1. Write a custom indexer. Given below is a sample indexer code.

```

package org.wso2.indexing.sample;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Arrays;
import org.apache.solr.common.SolrException;
import org.wso2.carbon.registry.core.exceptions.RegistryException;
import org.wso2.carbon.registry.core.utils.RegistryUtils;
import org.wso2.carbon.registry.indexing.IndexingConstants;
import org.wso2.carbon.registry.indexing.AsyncIndexer.File2Index;
import org.wso2.carbon.registry.indexing.Indexer;
import org.wso2.carbon.registry.indexing.solr.IndexDocument;

public class PlainTextIndexer implements Indexer {
    public IndexDocument getIndexedDocument(File2Index fileData) throws
SolrException,
    RegistryException {
        /* Create index document with resource path and raw content*/
        IndexDocument indexDoc = new IndexDocument(fileData.path,
RegistryUtils.decodeBytes(fileData.data), null);

        /* You can specify required field/value pairs for this indexing
document.
         * When searching we can query on these fields */
        Map<String, List<String>> fields = new HashMap<String,
List<String>>();
        fields.put("path", Arrays.asList(fileData.path));

        if (fileData.mediaType != null) {
            fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
Arrays.asList(fileData.mediaType));
        } else {
            fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
Arrays.asList("text/plain"));
        }

        /* set fields for index document*/
        indexDoc.setFields(fields);
        return indexDoc;
    }
}

```

2. Add the custom indexer JAR file to <APIM\_HOME>/repository/components/lib directory.
3. Update the <Indexers> element in <APIM\_HOME>/repository/conf/registry.xml file with the new indexer. The content is indexed using this media type. For example,

```

<indexers>
    <indexer class="org.wso2.indexing.sample.PlainTextIndexer"
mediaTypeRegEx="text/plain" profiles="default,api-store,api-publisher" />
</indexers>

```

The attributes of the above configuration are described below:

class	Java class name of the indexer
mefiaTypeRegEx	A regex pattern to match the media type
profiles	APIM profiles in which the indexer is available

4. Restart the server.
5. Add API documentation using the new media type and then search some term in the documentation using the syntax (**doc:keyword**). You will see how the documentation has got indexed according to the media type.

## Manage the API Lifecycle

In this section, we show you how to

- [Create a new API Version](#)
- [Deploy and Test as a Prototype](#)
- [Publish the new Version and Deprecate the old](#)

### Create a new API Version

A new **API version** is created when you want to change a published API's behaviour, authentication mechanism, resources, throttling tiers, target audiences etc. It isn't recommended to modify a published API that has subscribers plugged to it.

After creating a new version, you typically deploy it as a prototype for early promotion. A prototype can be used for testing, without subscription, along with the published versions of the API. After a period of time during which the new version is used in parallel with the older versions, the prototyped API can be published and its older versions deprecated.



The examples here use the PhoneVerification API, which is created in section [Create and Publish an API](#).

The steps below show how to create a new version of an existing API.

1. Log in to the API Publisher.
2. Click the **Browse** menu and select the API that you want to create a version of (e.g., PhoneVerification 1.0.0).
3. The API opens. Click the **Create New Version** button that appears in its **Overview** tab.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'APIs', 'Browse', 'Add', 'All Statistics', and 'My APIs'. Under 'My APIs', there are 'Subscriptions' and 'Statistics'. The main area shows the 'PhoneVerification - 1.0.0' API details. At the top, there are tabs for 'Overview' (which is highlighted with a red box), 'Versions', 'Docs', and 'Users'. Below the tabs, there's a summary card with icons for users (0), created (1.0.0), and docs. To the right is a table with API metadata:

Visibility	Public
Context	/phoneverify/1.0.0
Production URL	http://ws.cdyne.com/phoneverify/phon
Date Last Updated	5/19/2015, 3:19:29 PM
Tier Availability	Bronze,Unlimited,Gold,Silver
Default API Version	None
Published Environments	Production and Sandbox

At the bottom, there's a 'Create New Version' button.

4. Give a version number, check the default version option and click **Done**.

The dialog box has a title 'To Version'. It contains a text input field with '2.0.0' and a placeholder 'Ex:v1.0.1'. Below the input is a checkbox labeled 'Default Version' with the checked state highlighted by a red box. A message next to it says 'No default version defined for the current API'. At the bottom are two buttons: 'Done' (highlighted with a blue box) and 'Cancel'.

**Tip:** The **Default Version** option means that you make this version the default in a group of different versions of the API. A default API can be invoked without specifying the version number in the URL. For example, if you mark <http://host:port/youtube/2.0> as the default version when the API has 1.0 and 3.0 versions as well, requests made to <http://host:port/youtube/> get automatically routed to version 2.0.

If you mark any version of an API as the default, you get two API URLs in its **Overview** page in the API Store. One URL is with the version and the other is without. You can invoke a default version using both URLs.

If you mark an unpublished API as the default, the previous default, published API will still be used as the default until the new default API is published (or prototyped).

5. The **All APIs** page opens. Click on the new API version to open it.

The screenshot shows the 'APIs' dashboard with a sidebar on the left containing links like 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The main area is titled 'All APIs' with a search bar. It lists two API versions: 'PhoneVer.. - 1.0.0' and 'PhoneVer.. - 2.0.0'. The 'PhoneVer.. - 2.0.0' entry is circled in red.

6. Click the **Edit** link next to the API's name.

The screenshot shows the 'PhoneVerification - 2.0.0' API details page. The 'Lifecycle' tab is selected. The page displays the following information:

Context	/phoneverify
Production URL	<a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>
Sandbox URL	<a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>
Date Last Updated	04/12/2014 14:14:53
Tier Availability	Bronze,Gold,Unlimited,Silver
Default API Version	None

The sidebar on the left is identical to the one in the first screenshot, with 'Browse' highlighted.

7. Do the required modifications to the API. For example, assuming that the POST method is redundant, let's delete it from the resource that we added to the API at the time it was created.

## API Definition

URL Pattern  Url Pattern Ex: path/to/resource

GET     POST     PUT     DELETE    more

[+ Add](#)

<b>POST</b>	/CheckPhoneNumber <a href="#">+ Summary</a>	<a href="#"></a>
<b>GET</b>	/CheckPhoneNumber <a href="#">+ Summary</a>	<a href="#"></a>

[Save](#) [Next: Implement >](#)

- Click **Save** once the edits are done.

**Tip:** By default, only the latest version of an API is shown in the API Store. If you want to display multiple versions, set the `<DisplayMultipleVersions>` element to true in `<APIM_HOME>/repository/conf/api-manager.xml` file.

You have created a new version of an API. In the next tutorial, you deploy this API as a prototype and test it with its older versions.

### Deploy and Test as a Prototype

An **API prototype** is created for the purpose of early promotion and testing. You can deploy a new API or a new version of an existing API as a prototype. It gives subscribers an early implementation of the API that they can try out without a subscription or monetization, and provide feedback to improve. After a period of time, publishers can make changes the users request and publish the API.

The examples here use the API PhoneVerification 2.0.0, which is created in the previous tutorial.

- Log in to the API Publisher and select the API (e.g., PhoneVerification 2.0.0) that you want to p r o t o t y p e .

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following options: APIs, Browse (highlighted with a red box), Add, All Statistics, My APIs (highlighted with a red box), Subscriptions, Statistics, Tier Permissions, and Tier Permissions. The main area is titled "All APIs" and contains a search bar with "Filter APIs" and "Search". Below the search bar are two API entries: "PhoneVer.. - 1.0.0 (admin)" and "PhoneVer.. - 2.0.0 (admin)". Both entries have a blue gear icon and the status "PUBLISHED". The "PhoneVer.. - 2.0.0 (admin)" entry is highlighted with a red box.

2. Click the **Lifecycle** tab of the API and change its state to PROTOTYPED. After creating a new version, you typically deploy it as a prototype for the purpose of testing and early promotion.

**Tip :** The **Propagate Changes to API Gateway** option is used to automatically change the API metadata in the API Gateway according to the information published in the API Store.

**Tip :** You can also prototype an API by going to its **Implement** tab in the API Publisher and clicking the **Deploy as a Prototype** button .

3. Log in to the API Store, click the **Prototyped APIs** menu and then click the newly prototyped API.

4. The APIs **Overview** page opens. Note the following:

- There are no subscription options.
- There are two sets of URLs (with and without the version). This is because you marked the 2.0.0 version as the default version in step 4 of the previous tutorial.
- Other features like documentation, social media and forums are available.

## PhoneVerification - 2.0.0

 admin



<b>Rating:</b>	Your rating: N/A 
<b>Version:</b>	2.0.0
<b>Status:</b>	PROTOTYPED
<b>Updated:</b>	04/Sep/2014 19:14:16 PM IST

**Overview**

Documentation    API Console    Throttling Info

### Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/2.0.0](http://[REDACTED]:8280/phoneverify/2.0.0)

[http://\[REDACTED\]:8280/phoneverify](http://[REDACTED]:8280/phoneverify)

[https://\[REDACTED\]:8243/phoneverify/2.0.0](https://[REDACTED]:8243/phoneverify/2.0.0)

[https://\[REDACTED\]:8243/phoneverify](https://[REDACTED]:8243/phoneverify)

- Click the **API Console** tab and note that the POST method is not available as we removed that in the new version.

## PhoneVerification - 2.0.0

 admin



<b>Rating:</b>	Your rating: N/A 
<b>Version:</b>	2.0.0
<b>Status:</b>	PROTOTYPED
<b>Updated:</b>	20/May/2015 12:37:09 PM IST

Overview

Documentation

**API Console**

Throttling Info

### PhoneVerification

↗ Swagger ( /swagger.json )

Show/Hide | List Operations | Expand Operations

**GET** /CheckPhoneNumber

[ BASE URL: /phoneverify/2.0.0 , API VERSION: 1.0.0 ]

Let's invoke the prototyped API.

6. In the **API Console** of the prototyped API, expand the GET method, give the parameter values and invoke the API.

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			
<b>Try it out!</b>			

7. Note the response that appears in the console. You do not have to subscribe to the API or pass an authorization key to invoke a prototyped API.

### Response Body

```

<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query"> <Company>Toll Free</Company>
  <Valid>true</Valid>
    <Use>Assigned to a code holder for normal use.</Use>
    <State>TF</State>
    <RC />
    <OCN />
    <OriginalNumber>18006785432</OriginalNumber>
    <CleanNumber>8006785432</CleanNumber>
    <SwitchName />
    <SwitchType />
    <Country>United States</Country>
    <CLLI />
    <PrefixType>Landline</PrefixType>
    <LATA />
    <sms>Landline</sms>
    <Email />
    <AssignDate />
    <TelecomCity />
    <TelecomCounty />
  
```

8. Similarly, try to invoke the 1.0.0 version of the API without an access token and note that you get an authentication error as "Missing credentials."

Please subscribe to the API to generate an access token. If you already have an access token please provide it below.

Set Request Header Authorization : Bearer Access Token

## PhoneVerification

Swagger ( /swagger.json )

Show/Hide | List Operations | Expand Operations

**GET** /CheckPhoneNumber

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			

**Try it out!** Hide Response

**Request URL**

https://**.....**:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0

**Response Body**

```
<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
<ams:code>900902</ams:code>
<ams:message>Missing Credentials</ams:message>
<ams:description>Required OAuth credentials not provided. Make sure your API invocation call has a header: "Authorization: Bearer ACCESS_TOKEN"</ams:description>
</ams:fault>
```

In this tutorial, you have prototyped an API and tested it along with its older and published versions. In the next tutorial, you **publish** the prototyped API and deprecate its older versions.

#### Publish the new Version and Deprecate the old

You **publish an API** to make it available for subscription in the API Store. If you set up multiple tenants, your tenant store will be visible to other tenants as well. Therefore, users of the other tenants can view the APIs that are published in your default API Store. This allows you to advertise your APIs to a wider audience. Although the APIs that are published in your tenant store are visible to the users of other tenant stores, they need to log in to your tenant store in order to subscribe to and use them.



For a description of the API lifecycle stages, see [API lifecycle](#).

The steps below show how to publish an API to its default API Store:

1. Log in to the API Publisher as a user who has the publisher role assigned.
2. Click the API that you deprecated in the previous tutorial (e.g., PhoneVerification 2.0.0).

The screenshot shows the WSO2 API Manager dashboard. At the top, there's a navigation bar with tabs: 'API STORE', 'APIs', 'Prototyped APIs' (which is highlighted with a red box), 'My Applications', and 'My Subscriptions'. Below the navigation bar is a search bar labeled 'Search API'. On the left, there's a 'Recently Added' section showing a card for 'PhoneVerification-1.0.0' by 'admin' with a 5-star rating. On the right, under 'Prototyped APIs', there's a card for 'PhoneVerif.. (2.0...)' by 'admin'. A red box highlights this card.

3. Go to the API's **Lifecycle** tab and select the **PUBLISHED** state from the drop-down list. Then, select all the options and click **Update**.

**Tip:** The **Lifecycle** tab is only visible to users with publisher privileges.

The screenshot shows the 'PhoneVerification - 2.0.0' API details page. On the left is a sidebar with options: 'Browse' (highlighted with a red box), 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The main area shows the API details with tabs: 'Overview' (disabled), 'Lifecycle' (selected and highlighted with a red box), 'Versions', 'Docs', and 'Users'. Under the 'Lifecycle' tab, the 'State:' dropdown is set to 'PUBLISHED'. Below it are three checked checkboxes: 'Propagate Changes to API Gateway', 'Deprecate Old Versions', and 'Require Re-Subscription'. At the bottom are 'Update' and 'Reset' buttons, with 'Update' highlighted with a red box.

The three options are described below:

Option	Description
Propagate Changes to API Gateway	Automatically changes the API metadata in the API Gateway according to the information published in the API Store. If unselected, you have to manually configure the Gateway.
Deprecate Old Versions	Automatically deprecates all prior versions of the API, if any.

**Require Re-Subscription**

If **unchecked**, all users who are subscribed to the older version of the API will be automatically subscribed to the new version. If not, they need to subscribe to the new version again.

The API is now published to the default API Store and all its previous versions are deprecated.

- Log in to the default Store and click the **APIs** menu to see the API that you just published listed there. The older version (e.g., PhoneVerification 1.0.0) is no longer listed here as it is deprecated.

The screenshot shows the 'APIs' section of the WSO2 API Store. On the left, under 'Recently Added', there is a card for 'PhoneVerification...' version 1.0.0, which is marked as deprecated. On the right, the main list shows 'PhoneVerif.. (2.0...)' which is the current active version. The 'APIs' menu item in the top navigation bar is highlighted with a red box.

- Click the **My Subscriptions** menu and look under the **Subscribed APIs** section. The subscriptions made to the older API versions must be deprecated now.

The screenshot shows the 'My Subscriptions' page. At the top, the 'My Subscriptions' menu item is highlighted with a red box. Below it, the 'Subscriptions' section is displayed. Under 'Subscribed APIs', there is a list for 'DefaultApplication' showing a single entry for 'PhoneVerif.. - 1.0.0' which is marked as deprecated.

## Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times at different SLA levels.

### Applications With Subscriptions

DefaultApplication

[Keys - Production >](#)

[Keys - Sandbox >](#)

#### Subscribed APIs ▾

Deprecated

PhoneVerif.. - 1.0.0

admin

Bronze Subscription

 **Tip:** When an API is deprecated, new subscriptions are disabled (you cannot see the subscription options) and existing subscribers can continue to use the API as usual until it is eventually retired.

You have published an API to the API Store and deprecated its previous versions.

## Publish to Multiple External API Stores

You can share an API to application developers who are subscribed to the API Stores of other tenants. This allows you to advertise your APIs to a wider community. Subscribers of other tenant stores can view and browse your APIs but to subscribe to them, the users must visit your (the original publisher's) store.

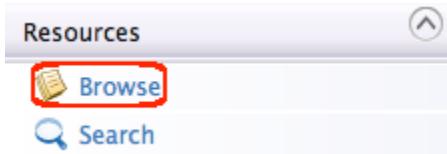
Capability to publish to external API Stores is not there by default. Follow the steps below to configure it. In this guide, we use two separate instances of the API Manager and we publish from one instance to the Store of the other instance.

1. Copy the API Manager product pack to two different locations.
2. Go to the <APIM\_HOME>/repository/conf/carbon.xml file of the **second** instance and change its port by an offset of 1. You do this to avoid the port conflicts that occur when you run more than one WSO2 products on the same host.

```
<Offset>1</Offset>
```

 **Tip:** Please check whether the Thrift port under the <ThriftClientPort> and <ThriftServerPort> elements are changed according to the port offset in the <APIM\_HOME>/repository/conf/api-manager.xml file. The default Thrift port is 10397.

3. Start both servers. Let's publish from the first instance of the API Manager to the Store of the second instance.
4. Log in to APIM admin console of the **first** instance (<https://<Server Host>:9443/carbon>) as admin. In a **multitenant environment**, you must log in using the tenant's credentials.
5. Select **Browse** menu under **Resources**.



6. The Registry opens. Go to the /\_system/governance/apimgt/externalstores/external-api-stores.xml resource.

The screenshot shows the 'Browse' interface of the WSO2 API Manager. At the top, there's a navigation bar with 'Home > Resources > Browse'. Below it, a title 'Browse' is displayed. On the left, a tree view shows the directory structure starting from 'Root /'. Under 'Root /', there are several folders: '\_system', 'config', 'governance', 'apimgt', 'applicationdata', 'customsequences', 'externalstores', and 'external-api-stores.xml'. The 'external-api-stores.xml' file is highlighted with a red box. At the bottom of the tree view, there are buttons for 'Tree view' and 'Detail view'. A 'Location:' input field is also present above the tree view.

7. Click the **Edit as Text** link and change the `<ExternalAPIStores>` element of each external API store that you need to publish APIs to. In this example,

- `http://localhost:9764/store` is the API Store of the second API Manager instance.
- You publish to its super tenant's Store (admin/admin).
- The port is 9764 as you incremented it by 1 earlier.
- If the second API Manager instance has multiple tenants and you want to a tenant's Store, the tenant's Store URL and credentials must be given here.

```
<ExternalAPIStores>
    <StoreURL>http://localhost:9763/store</StoreURL>
        <ExternalAPIStore id="Store1" type="wso2">
            className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
                <DisplayName>Store1</DisplayName>
                <Endpoint>http://localhost:9764/store</Endpoint>
                <Username>admin</Username>
                <Password>admin</Password>
            </ExternalAPIStore>
        </ExternalAPIStores>
```

- In a **multitenant environment**, each tenant can publish to different external Stores by changing the above file in their tenant space. Also, see [API visibility and subscription](#) on how APIs appear and are available for subscription in a multitenant environment. Note that publishing to an external Store only means that the API is advertised there. To subscribe, you must always register and log in to the original publisher's tenant Store.

Note the following in the configuration above:

Element	Description
<ExternalAPIStore id="" type="" className="">	id: The external store identifier, which is a unique value. type: Type of the Store. This can be a WSO2-specific API Store or an external one. className: The implementation class inside the API Manager distribution.
< StoreURL>	URL of the API store of the current APIM deployment. This is the URL to the API in the original publisher's store. APIs that are published to external stores will be redirected to this URL.
< DisplayName>	The name of the Store that is displayed in the publisher UI.
< Endpoint>	URL of the API Store.
<Username> & <Password>	Credentials of a user who has permissions to create and publish APIs.

Registry changes are applied dynamically. You do not need to restart the server.

8. Log in to the API Publisher of the first instance as admin/admin and [create an API](#). In a multitenant environment, log in to the API Publisher using your tenant's credentials.
9. Click on the newly created API to see a new tab called **External API Stores** added to the API Publisher console. Select the Store that you want to publish to (in this case, Store1) and click **Save**.

The screenshot shows the WSO2 API Publisher interface. On the left, there is a sidebar with the following menu items:

- APIs (highlighted with a red box)
- Add
- All Statistics
- My APIs
- Subscriptions
- Statistics
- Tier Permissions
- Tier Permissions

The main content area shows the details for an API named "PhoneVerification-1.0.0". The URL is "APIs / All / PhoneVerification-1.0.0". Below the URL, there is a "PhoneVerification - 1.0.0" card with the following tabs: Overview, Lifecycle, Versions, Docs, Users, and External API Stores (highlighted with a red box). Under the "Overview" tab, there is a section for "Display In External Stores" with a checked checkbox next to "Store1". At the bottom of the card, there are "Save" and "Cancel" buttons (the "Save" button is highlighted with a red box).

Note the following:

- You can select multiple external API stores and click **Save** to publish your API to them.
- If the API creator updates the API after publication to external stores, either the creator or a publisher can simply push those changes to the published stores by selecting the stores and clicking **Save** again.
- If the API creator deletes the API, each external store that it is published to will receive a request to delete the API.

10. Log in to an external API Store (in this case, `http://localhost:9764/store`) and click on the API that you just published.
11. A link appears as **View Publisher Store** and it directs you to the original publisher's store (in this case, `http://localhost:9763/store`) through which you can subscribe to the API.

You have added multiple external stores to your registry and published your APIs to them.

### Engage a new Throttling Policy

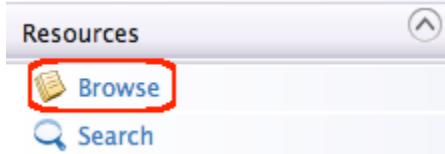
The steps below show how to engage a throttling policy to an API.

**Tip:** For a description of throttling, see [Throttling Tiers](#).

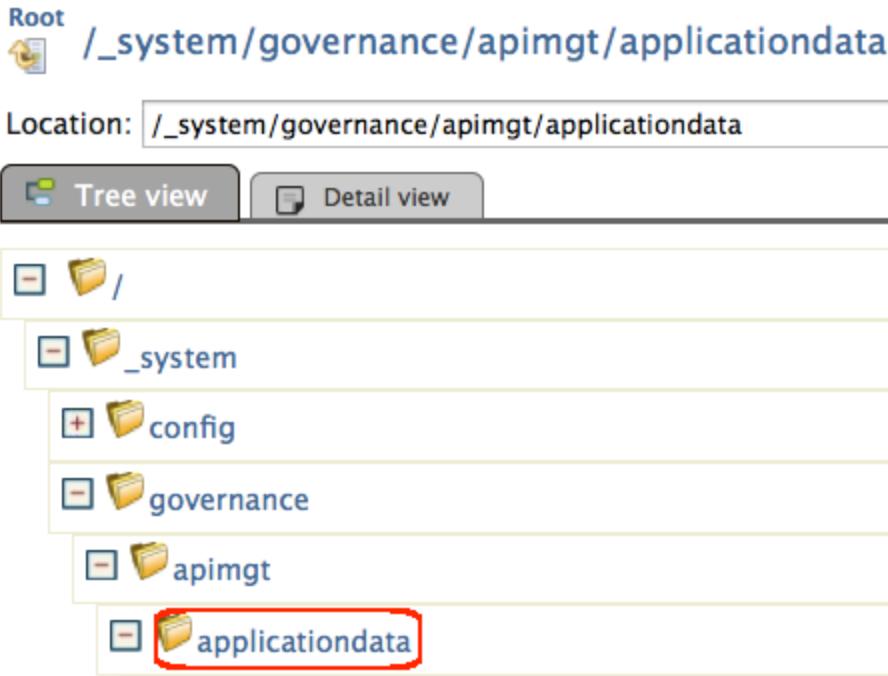
1. Write your throttling policy. For example, the following sample throttling policy points to a backend service and allows 1000 concurrent requests to a service.

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
  lity-1.0.xsd"
  xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle"
  wsu:Id="WSO2MediatorThrottlingPolicy">
  <throttle:MediatorThrottleAssertion>
    <throttle:MaximumConcurrentAccess>1000</throttle:MaximumConcurrentAccess>
    <wsp:Policy>
      <throttle:ID throttle:type="IP">other</throttle:ID>
    </wsp:Policy>
  </throttle:MediatorThrottleAssertion>
</wsp:Policy>
```

2. Log in to the API Manager's management console (<https://localhost:9443/carbon>) and click the **Resource > Browse** menu to view the registry.



3. Click the `/_system/goverence/apimgt/applicationdata` path to go to its detailed view.



4. In the detail view, click the **Add Resource** link.

### Browse

Root `/_system/governance/apimgt/applicationdata`

Location: `/_system/governance/apimgt/applicationdata`

**Metadata**

**Properties**

**Entries**

5. Upload the policy file to the server as a registry resource .

6. In the management console, select the **Service Bus > Source View** menu.



7. The configurations of all APIs created in the API Manager instance opens. To engage the policy to a selected API, add it to your API definition. In this example, we add it to the login API.

```

<api xmlns="http://ws.apache.org/ns/synapse" name="_WSO2AMLoginAPI_"
context="/login">
    <resource methods="POST" url-mapping="/*">
        <inSequence>
            <send>
                <endpoint>
                    <address uri="https://localhost:9493/oauth2/token"/>
                </endpoint>
            </send>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
            <property name="id" value="A"/>
            <property name="policyKey"
value="gov:/apimgt/applicationodata/throttle.xml"/>
        </handler>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
        </handlers>
    </api>

```



**Note:** Be sure to specify the same path used in step 5 in the policy key of your API definition. Also, use the same tier name you selected when creating the API as the throttle id in the policy (example <throttle:ID throttle:type ="ROLE">Gold</throttle:ID>.)

You have successfully engaged a throttling policy to an API.

### Change the Default Mediation Flow of API Requests

The API Gateway has a default mediation flow for the API invocation requests that it receives. You can extend this default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a synapse mediation sequence. You design all sequences using a tool like WSO2 Developer Studio and then store the sequence in the Gateway's registry.

Let's see how to create a custom sequence using WSO2 Developer Studio and then deploy and use it in your APIs.

1. Log in to the API Publisher.
2. Click **Add** to create an API with the following information and then click **Implement**.

Field		Sample value
Name		YahooWeather
Context		/weather
Version		1.0
Resources	URL pattern	current/{country}/{zipcode}
	Request types	GET method to return the current weather conditions of a zip code that belongs to a particular country

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'Browse', 'Add' (which is highlighted with a red box), 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The main area is titled 'Design API' and contains two sections: 'General Details' and 'Resources'. In 'General Details', fields include 'Name' (YahooWeather), 'Context' (/weather), 'Version' (1.0), 'Visibility' (Public), 'Thumbnail Image' (No file chosen), 'Description' (empty), and 'Tags' (Add tags). In 'Resources', there's a URL pattern '/{context}/{version}/current/{country}/{zipcode}' with a GET method selected. Below it, there's a 'Resource Name' field with 'current' and a 'Add New Resource' button. At the bottom, there are 'Save', 'Implement' (which is highlighted with a red box), and 'Cancel' buttons.

3. The **Implement** tab opens. Provide the information given in the table below and click **Manage**.

Field	Sample value
Implementation method	Backend
Endpoint type	HTTP endpoint
Production endpoint	You can find the Yahoo weather API's endpoint from <a href="https://developer.yahoo.com/weather/">https://developer.yahoo.com/weather/</a> . Copy the part before the '?' sign to get this URL: <a href="https://query.yahooapis.com/v1/public/yql">https://query.yahooapis.com/v1/public/yql</a>

**YahooWeather : /weather/1.0**

Implementation Method  Backend Endpoint  Specify Inline

## Endpoints

Endpoint Type: **HTTP Endpoint**

Production Endpoint: **https://query.yahooapis.com/v1/** [Advanced Options](#) [Test](#)  
E.g., http://appserver/resource

Sandbox Endpoint: [Advanced Options](#) [Test](#)  
E.g., http://appserver/resource

[Show More Options](#)

**Save** **Deploy Prototype** **Manage** **Cancel**

4. Click **Manage** to go to the Manage tab, provide the following information and click **Save & Publish** once you are done.

Field	Sample value
Tier Availability	Gold
Keep the default values for the other attributes	

1 Design    2 Implement    3 Manage

## YahooWeather : /weather/1.0

### Configurations

Make this default version   No default version defined for the current API

Tier Availability: **Gold**

Transports:  HTTP  HTTPS

Sequences:  Check to select a custom sequence to be executed in the message flow

Response Caching: Disabled

Subscriptions: Available to current tenant

### Business Information

### Resources

**Add Scopes**

/current

**GET** /current/{country}/{zipcode} [+ Summary](#) Application & Application User Unlimited [+ Scope](#)

Save **Save & Publish** Cancel

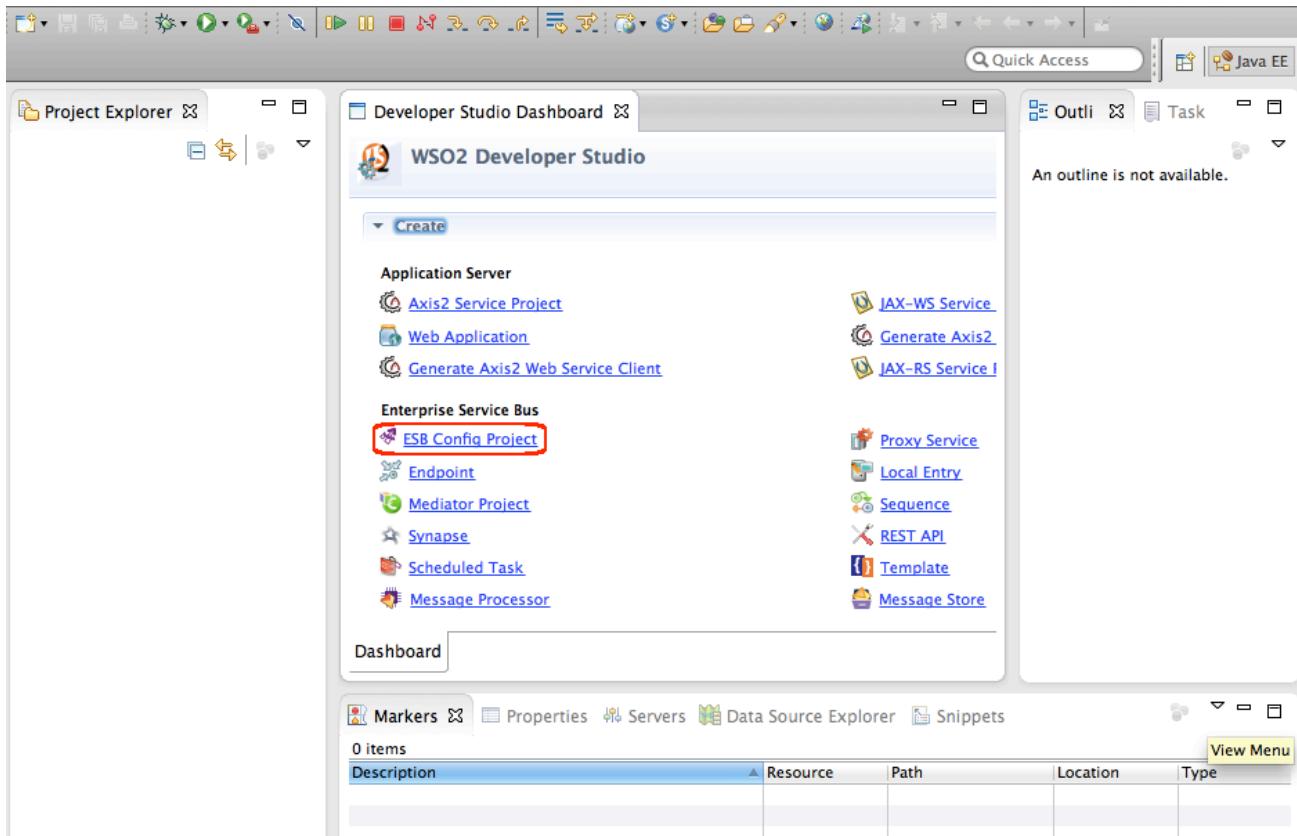
5. Download WSO2 Developer Studio (version 3.7.1 is used here) from <http://wso2.com/products/developer-studio/> and open it by double clicking the Eclipse.app file inside the downloaded folder.

**Tip:** To start Eclipse on a Mac for the first time, open a terminal and execute the following commands:

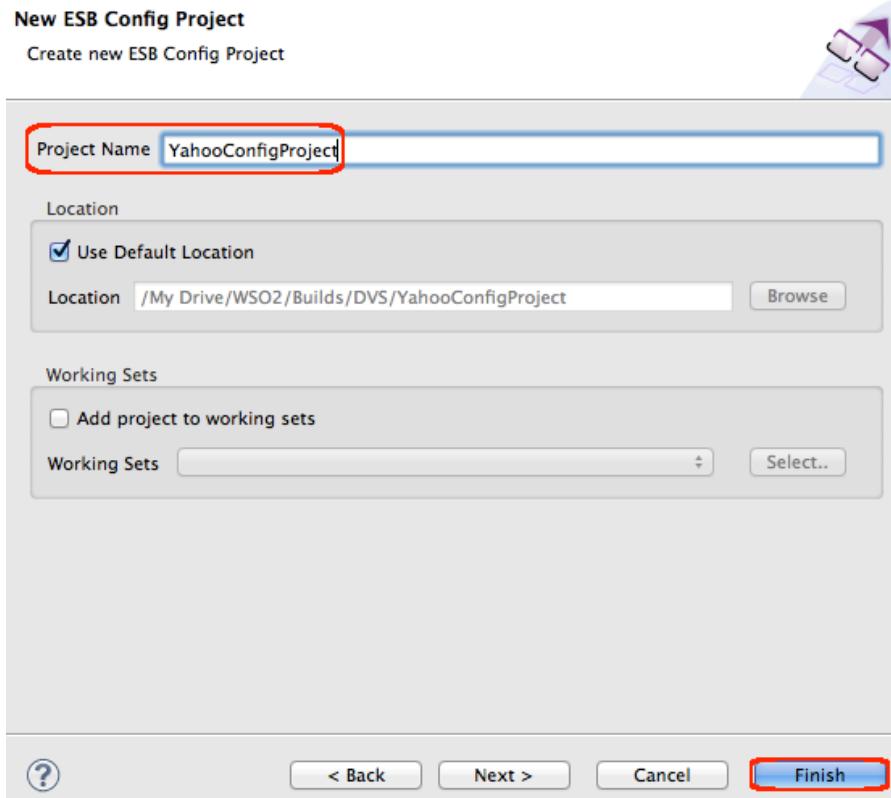
```
cd <DevStudio_Home>/Eclipse.app/Contents/MacOS/
chmod +x eclipse
./eclipse
```

Thereafter, you can start Eclipse by double-clicking the Eclipse icon in <DevStudio\_Home>.

6. Click the **Developer Studio** menu and choose **Open Dashboard**. When the dashboard opens, click the **ESB Config Project** link.



7. Create a new ESB project by the name YahooConfigProject.



8. Click the **Sequence** link on the Developer Studio Dashboard and create a new sequence by the name Yahoo WeatherSequence and the YahooConfigProject project.

**Create New Sequence**

Give a name for the sequence



Sequence Name: **YahooWeatherSequence**

Save Sequence in: **YahooConfigProject**

[Create new Project...](#)

**Advanced Configuration**

On Error Sequence:

Available Endpoints:

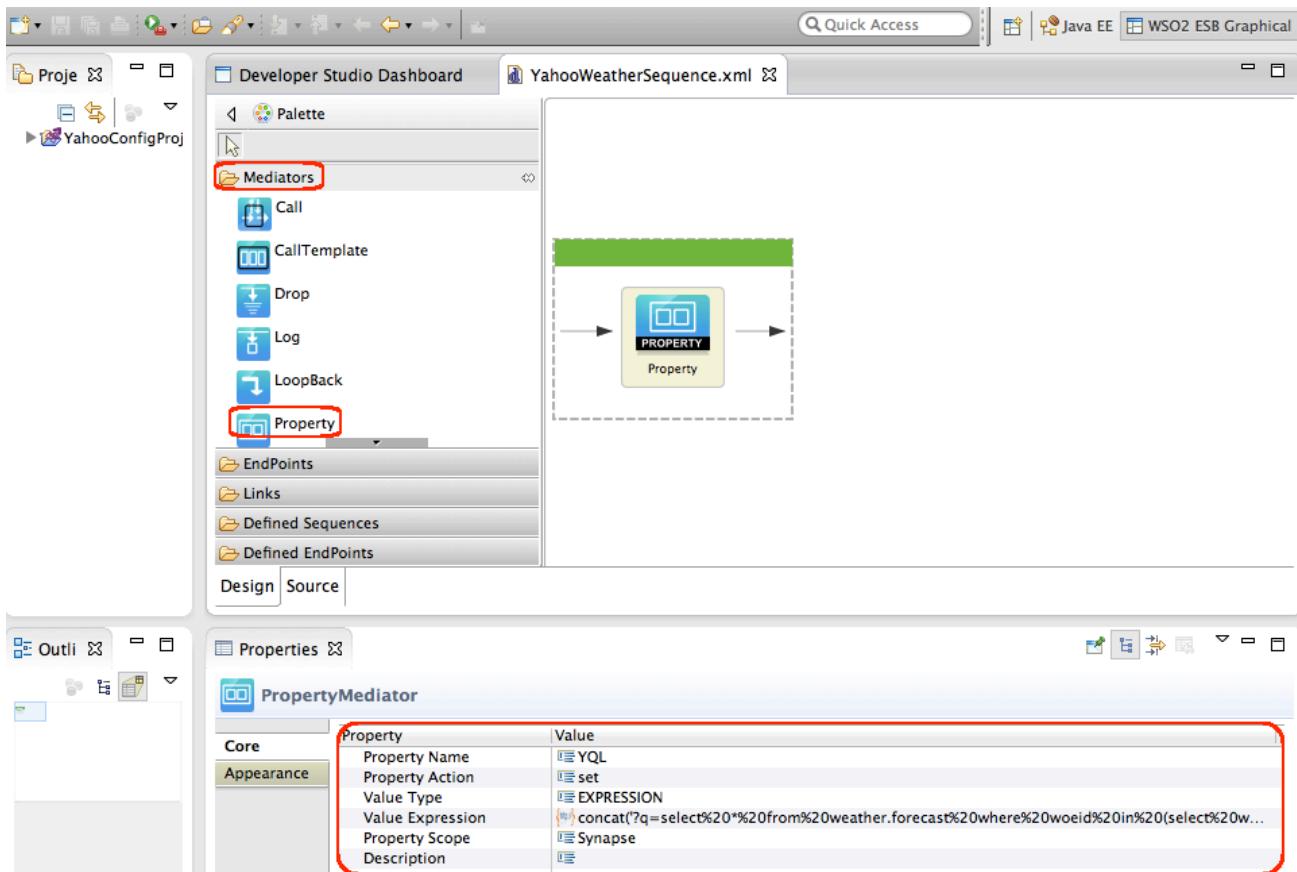
Make this as Dynamic Sequence

Registry: **Governance**

Registry Path:

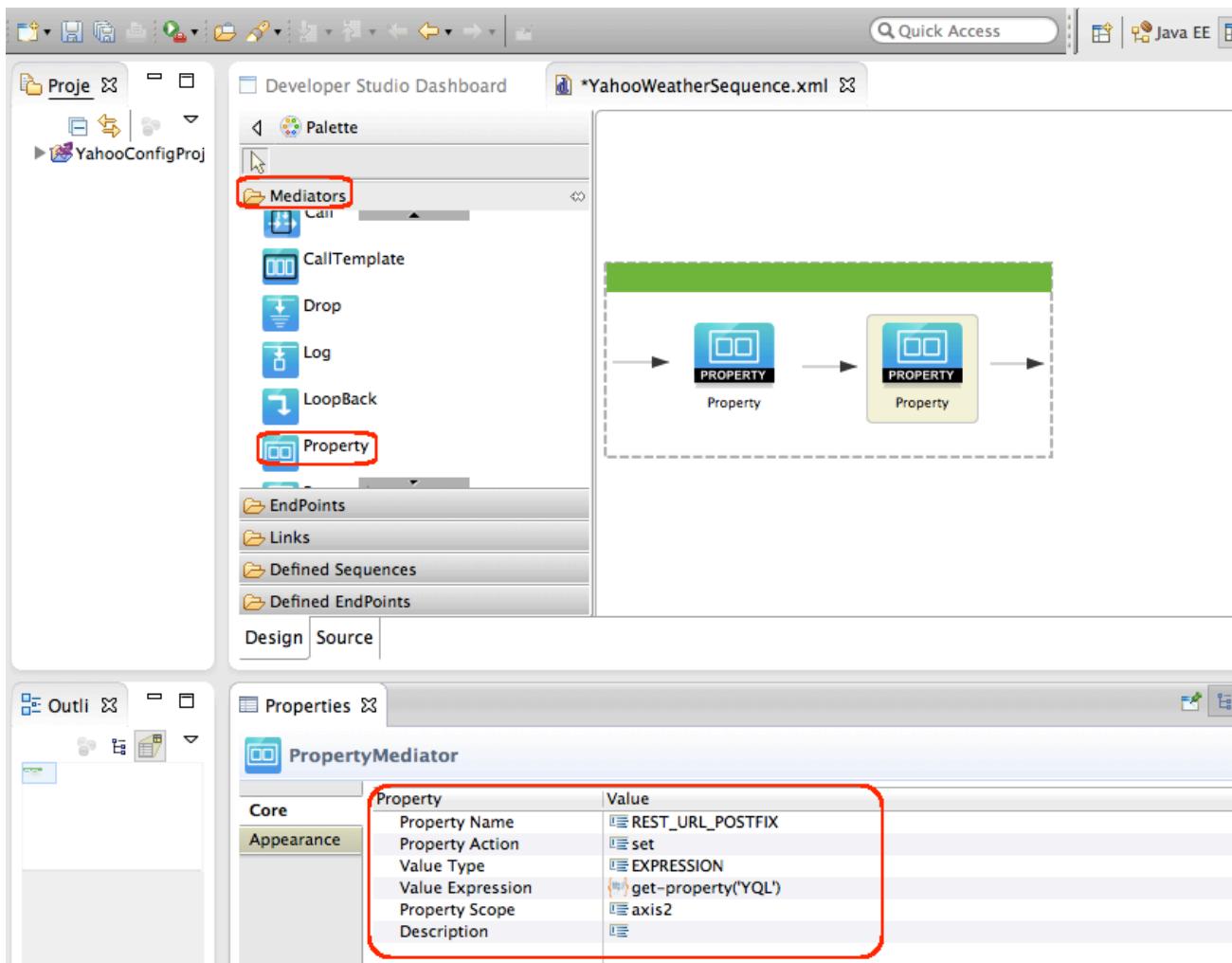
9. Your sequence now appears on the Developer Studio console. From under the Mediators section, drag and drop a property mediator to your sequence and give the following values to the property mediator.

Property Name	YQL
Value Type	Expression
Value Expression	For the XPath expression, we take a part of the query in the Yahoo API's endpoint ( <a href="https://deve">https://deve</a> <pre>concat(' ?q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20(se</pre>

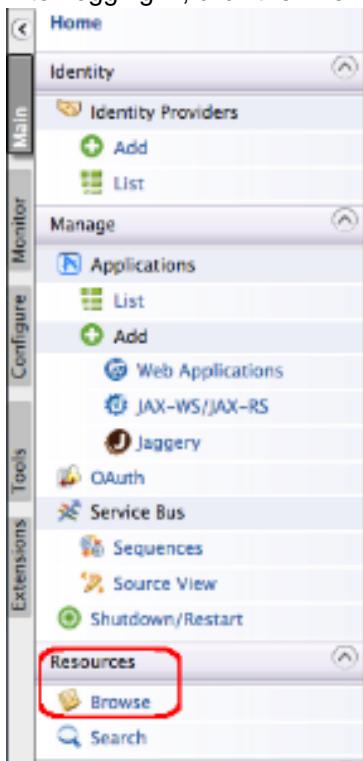


10. Similarly, add another property mediator with the following values. This is an HTTP transport property that appends its value to the address endpoint URL. Once you are done, save the sequence.

Property Name	REST_URL_POSTFIX
Value Type	Expression
Value Expression	Set the value of the property mediator that you created earlier as get-property('YQL')
Property Scope	Axis2



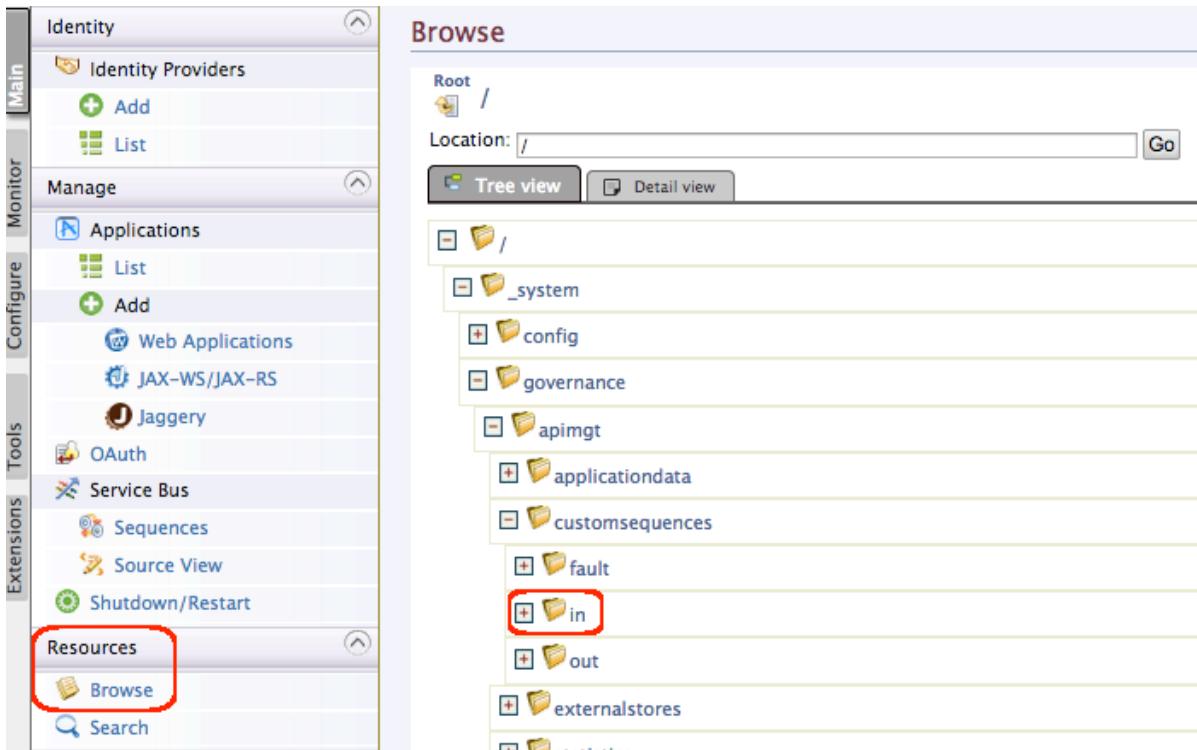
11. Log in to the API Gateway's management console (<https://localhost:9443/carbon>) as admin.
12. After logging in, click the **Browse** menu under the **Resources** menu.



13. When the API Gateway's registry opens, navigate to the registry path `/_system/governance/apimgt/customsequences/in`.

This is because you want the custom sequence to be invoked in the `In` direction or the request path. If you want it to be involved in the `Out` or `Fault` paths, select the respective location under `customsequences`.

 **Tip:** If you prefer not to use the registry to upload the sequence or want to engage a sequence to all APIs in the APIM at once, you can do so by saving the mediation sequence XML file in the file system. See [Adding Mediation Extensions](#) for details.



The screenshot shows the WSO2 API Manager Registry interface. The left sidebar has tabs for Main, Monitor, Configure, and Tools. Under Main, there are sections for Identity (Identity Providers, Add, List), Manage (Applications, List, Add, Web Applications, JAX-WS/JAX-RS, Jaggery, OAuth, Service Bus, Sequences, Source View, Shutdown/Restart), and Resources (Browse, Search). The 'Browse' link in the Resources section is highlighted with a red box. The main area is titled 'Browse' and shows a tree view of the registry structure under 'Root /'. The path '\_system/governance/apimgt/applicationdata/customsequences' is selected. The 'customsequences' folder is expanded, showing subfolders 'fault', 'in' (which is also highlighted with a red box), and 'out'.

14. Click **Add Resource** and upload the XML file of the sequence that you created earlier.

**Browse**

Root [/\\_system/governance/apimgt/customsequences/in](/_system/governance/apimgt/customsequences/in)

Location: /\_system/governance/apimgt/customsequences/in

**Metadata**

**Properties**

**Entries**

**Add Resource**

Method

**Upload Content From File**

File *	<input type="button" value="Choose File"/> YahooWeatherSequence.xml Give the path of a file to fetch content (xml,wsdl,jar etc..)
Name *	YahooWeatherSeque
Media type	application/xml
Description	<input type="text"/>

15. Log back in to the API Publisher, select the API that you created earlier and click the **Edit** link right next to its name to go to the edit wizard.

**WSO2 API PUBLISHER**

**APIs**

- 
- 
- 
- My APIs**
- 
- 
- Tier Permissions**
- 

APIs / All / YahooWeather-1.0

**YahooWeather - 1.0**

	Context	/weather
<input type="button" value="0 Users"/>	Production URL	<a href="https://query.yahooapis.com/v1/public/yql">https://query.yahooapis.com/v1/public/yql</a>
<input type="button" value="PUBLISHED"/>	Date Last Updated	2/25/2015, 11:28:59 AM
<input type="button" value="1.0"/>	Tier Availability	Gold
<input type="button" value="Docs"/>	Default API Version	None

16. Navigate to the API's **Manage** tab, click the check box next to sequences and select the sequence that you created for the In flow. Next, **Save & Publish** the API again.

 **Tip:** It might take a few minutes for the sequence to be uploaded into the API Publisher. If it isn't

there, please check again later.

**APIs**

- Browse
- Add
- All Statistics
- My APIs**
- Subscriptions
- Statistics
- Tier Permissions**
- Tier Permissions

**YahooWeather : /weather/1.0**

**Configurations**

Make this default version  ⓘ No default version defined for the current API

Tier Availability: **Gold** ⓘ

Transports:  HTTP  HTTPS

Sequences:  ⓘ Check to select a custom sequence to be executed in the message flow

In Flow	Out Flow	Fault Flow
YahooV	Non	N

Response Caching: Disabled ⓘ

Subscriptions: Available to current tenant ⓘ

**Business Information** >

**Resources**

**/current**

**GET** /current/{country}/{zipcode} + Summary Application & Application User Unlimited + Scope

Save **Save & Publish** Cancel

17. Open the API Store, subscribe to the API that you just published and generate the access tokens in order to invoke the API.

**WSO2 API STORE**

APIs Prototyped APIs My Applications My Subscriptions Forum Statistics Tools Themes

Search API Go

More APIs from 'admin'

PhoneVerification-1.0.0 ★★★★★

**YahooWeather - 1.0**

admin

Rating: Your rating: N/A ★★★★★

Version: 1.0

Status: PUBLISHED

Updated: 25/Feb/2015 15:24:23 PM IST

Applications DefaultApplication

Tiers Gold

Allows 20 request(s) per minute.

Subscribe

Overview Documentation API Console Throttling Info Forum

Production and Sandbox URLs:

18. Click the **API Console** tab of the API. It opens the integrated API Console using which you can invoke the API.

## YahooWeather - 1.0

**admin**



**Rating:** Your rating: N/A  
★★★★★

**Version:** 1.0

**Status:** PUBLISHED

**Updated:** 22/Jun/2015 16:47:08 PM IST

**Applications**

**Tiers**

Allows 20 request(s) per minute.

**Subscribe**

[Overview](#)   [Documentation](#)   **API Console**   [Throttling Info](#)   [Forum](#)

Try  On

**Set Request Header**   **Authorization : Bearer**

19. Give the following values for the parameters and invoke the API. You can also give any other value of your choice.

country	usa
zipcode	95004

**GET /current/{country}/{zipcode}**

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
country	<input type="text" value="usa"/>		path	string
zipcode	<input type="text" value="95004"/>		path	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			

**Try it out!**

20. Note the response that you get as a JSON object from Yahoo.

### Response Body

```
{
  "query": {
    "count": 1,
    "created": "2015-06-22T11:13:13Z",
    "lang": "en-US",
    "results": {
      "channel": {
        "title": "Yahoo! Weather - Aromas, CA",
        "link": "http://us.rd.yahoo.com/dailynews/rss/weather/Aromas__CA/*http://weather.yahoo.com/forecast/USCA0044_f.html",
        "description": "Yahoo! Weather for Aromas, CA",
        "language": "en-us",
        "lastBuildDate": "Mon, 22 Jun 2015 3:53 am PDT",
        "ttl": "60",
        "location": {
          "city": "Aromas",
          "country": "United States",
          "region": "CA"
        },
        "units": {
          "distance": "mi",
          "pressure": "in",
        }
      }
    }
  }
}
```

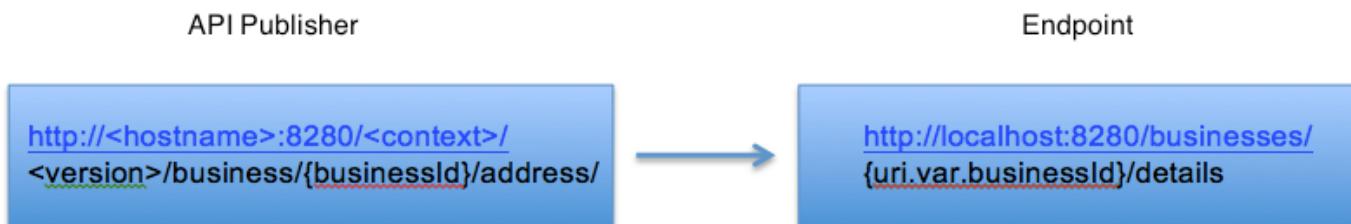
In this tutorial, you created a sequence to change the default mediation flow of API requests, deployed it in the API Gateway and invoked an API using the custom mediation flow.

#### Map the Parameters of your Backend URLs with the API Publisher URLs

This tutorial explains how to map your backend URLs to the pattern that you want in the API Publisher. Note the following:

1. The URL pattern of the APIs in the Publisher is `http://<hostname>:8280/<context>/<version>/<API resource>`.
2. You can define variables as part of the URI template of your API's resources. For example, in the URI template `/business/{businessId}/address/`, `businessId` is a variable.
3. The variables in the resources are read during mediation runtime using property values with the "`uri.var.`" prefix. For example, this HTTP endpoint gets the `businessId` that you specify in the resource `http://localhost:8280/businesses/{uri.var.businessId}/details`.
4. The URI template of the API's resource is automatically appended to the end of the HTTP endpoint at runtime. You can use the following mediator setting to remove the URL postfix from the backend endpoint: `<property name="REST_URL_POSTFIX" scope="axis2" action="remove" />`.

We do the following mapping in this tutorial:



**Before you begin**, note that a mock backend implementation is set up in this tutorial for the purpose of demonstrating the API invocation. If you have a local API Manager setup, save [this file](#) in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder to set up the mock backend.

- Log in to the API Publisher, click the **Add** link, give the information in the table below and then click **Implement**.

Field		Sample value
Name		TestAPI
Context		/test
Version		1.0.0
Visibility		Public
Resources	URL pattern	/business/{businessId}/address/
	Request types	GET

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with 'APIs' and 'Add' buttons. The 'Add' button is highlighted with a red box. The main area has a 'Design' tab highlighted with a red box. The 'General Details' section contains fields for Name, Context, Version, and Visibility. Below that is a 'Resources' section with a URL pattern and a method dropdown set to 'GET'.

- The **Implement** tab opens. Give the information in the table below.

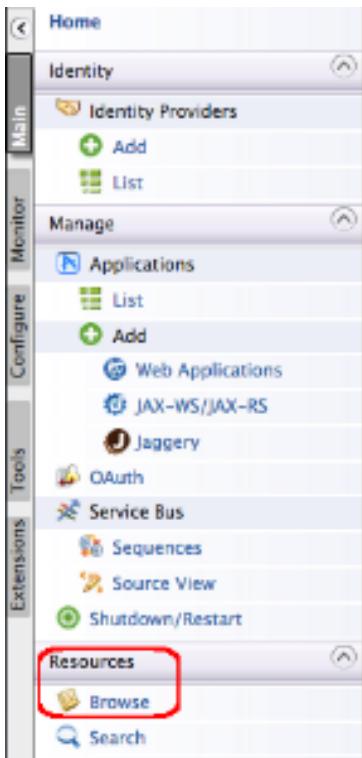
Field	Sample value
Implementation method	Backend
Endpoint type	HTTP endpoint
Production endpoint	<a href="http://localhost:8280/businesses/{uri.var.businessId}/details">http://localhost:8280/businesses/{uri.var.businessId}/details</a>

The screenshot shows the WSO2 API Manager's implementation interface for the 'TestAPI : /test/1.0.0' API. At the top, a navigation bar indicates the current step: '1 Design' (light blue), '2 Implement' (blue with a red border), and '3 Manage' (light gray). The main title is 'TestAPI : /test/1.0.0'. Below it, an 'Implementation Method' dropdown is set to 'Backend Endpoint' (radio button selected). The 'Production Endpoint' field contains the value '30/businesses/{businessId}/detail'. To its right are 'Advanced Options' and 'Test' buttons. A placeholder text 'E.g.: http://appserver/resource' is provided. Below this section, a 'Sandbox Endpoint' field is shown with an empty input field, 'Advanced Options' button, and 'Test' button. Another placeholder text 'E.g.: http://appserver/resource' is provided. At the bottom of the form are four buttons: 'Save' (blue), 'Deploy Prototype' (orange), 'Manage' (blue), and 'Cancel' (gray).

3. Click **Manage** to go to the Manage tab, select the Gold tier and publish the API. As the API's resource is appended to its endpoint by Synapse at runtime, let's write a custom sequence to remove this appended resource.
4. Copy the following to a text editor and save the file in XML format (e.g., TestSequence.xml).

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="TestSequence">
    <property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>
</sequence>
```

5. Log in to the API Gateway's management console. If you are using WSO2 Cloud, the Gateway URL is <https://gateway.api.cloud.wso2.com/carbon/admin/login.jsp>. If you are using a local setup, the URL is <https://localhost:9443/carbon>. You can see the username on the top right-hand corner of the API Publisher.
6. After logging in, click the **Browse** menu under the **Resources** menu.



7. When the API Gateway's registry opens, navigate to the registry path `/_system/governance/apimgt/customsequences/in`. This is because you want the custom sequence to be invoked in the `In` direction or the request path.

The screenshot shows the WSO2 API Manager registry browser. The left sidebar has the 'Main' tab selected. Under the 'Resources' section, the 'Browse' option is highlighted with a red box. The right panel shows a tree view of the registry structure. The path `/_system/governance/apimgt/customsequences/in` is selected, with the `in` folder highlighted with a red box.

8. Click **Add Resource** and upload the XML file of the sequence that you created earlier.

**Browse**

Root / \_system/governance/apimgt/customsequences/in

Location: / \_system/governance/apimgt/customsequences/in

**Metadata**

**Properties**

**Entries**

Add Resource

**Add Resource**

Method

**Upload Content From File**

File *	<input type="button" value="Choose File"/> YahooWeatherSequence.xml Give the path of a file to fetch content (xml,wsdl,jar etc..)
Name *	YahooWeatherSeque
Media type	application/xml
Description	<input type="text"/>

9. Log back to the API Publisher, click the **Edit** link associated with the API, navigate to the **Manage** tab, click the **Sequences** check-box and engage the sequence that you created to the API.

**WSO2 API PUBLISHER**

**APIs**

- Browse
- Add
- All Statistics
- My APIs**
- Subscriptions
- Statistics
- Tier Permissions**
- Tier Permissions

**TestAPI : /test/1.0.0**

**Configurations**

Make this default version  No default version defined for the current API

Tier Availability: **Gold**

Transports:  HTTP  HTTPS

Sequences:  Check to select a custom sequence to be executed in the message flow

In Flow	Out Flow	Fault Flow
TestSe	None	Nor

10. Save the API.

You have created an API. Let's subscribe to the API and invoke it.

11. Log in to the API Store and subscribe to the API.

The screenshot shows the WSO2 API Store interface. At the top, there's a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, and Themes. Below the navigation bar is a search bar labeled "Search API". The main content area displays "TestAPI - 1.0.0" with a rating of 5 stars. On the left, there's a sidebar titled "More APIs from 'admin'" showing "TestAPI1-1.0.0" and "TestAPI2-1.0.0", each with a 5-star rating and a gear icon. The right side shows detailed information for "TestAPI - 1.0.0" including Rating (N/A), Version (1.0.0), Status (PUBLISHED), and Updated (19/Mar/2015 11:46:04 AM IST). A "Subscribe" button is located at the bottom right of this section, which is highlighted with a red box.

#### Production and Sandbox URLs:

12. When prompted, choose to go to the **My Subscriptions** page and generate an access token to invoke the API.

The screenshot shows the WSO2 API Manager interface. At the top, there's a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions (which is highlighted with a red box), Forum, and Statistics. Below the navigation bar is a search bar labeled "Search API". The main content area displays "Subscriptions". Under "Applications With Subscriptions", there's a dropdown menu set to "DefaultApplication". The "Keys - Production" section contains fields for "Consumer Key" (gTL5LVKtCzHJxsPnhR1PRTEf0aYa), "Consumer Secret" (eEqbmykM7BLC5UZv5Eek9j2H38ka), and "Access Token" (acca1f3034f64126f87132c699294c, which is highlighted with a red box). There are also sections for "Allowed Domains" (set to "ALL") and "Update Domains". At the bottom, there are buttons for "cURL", "Validity Time: 3600 Seconds", and "Re-generate".

13. Click the API Console tab of your API.

## TestAPI - 1.0.0

 admin



Rating:	Your rating: N/A
Version:	1.0.0
Status:	PUBLISHED

Updated: 19/Mar/2015 14:27:52 PM IST

Applications  
Select Application...

Tiers  
Gold

Allows 20 request(s) per minute.

[Subscribe](#)

[Overview](#) [Documentation](#) [API Console](#) [Throttling Info](#) [Forum](#)

Try DefaultApplication On Production Environment.

[Set Request Header](#) Authorization : Bearer 4b5dc512832e8ec6cc6e3c5b3bb5f

### TestAPI

[Swagger Resource Listing \( /api-docs \)](#)

business :

[Checklist](#) [List Operations](#) [Expand Operations](#) [Down](#)

14. Note that the businessId is added in the UI as a parameter. Give a businessId and click **Try it out** to invoke the API.

**GET** /business/{businessId}/address/

Parameter	Value	Description	Parameter Type	Data Type
businessId	123		path	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200			

[Try it out!](#) [Hide Response](#)

15. Note the response that you get. According to the mock backend used in this tutorial, you get the response as "Received Request" Request."

#### Response Body

```
<response><value>Received Request</value></response>
```

In this tutorial, you mapped the URL pattern of the APIs in the Publisher with the endpoint URL pattern of a sample backend.

#### Convert a JSON Message to SOAP and SOAP to JSON

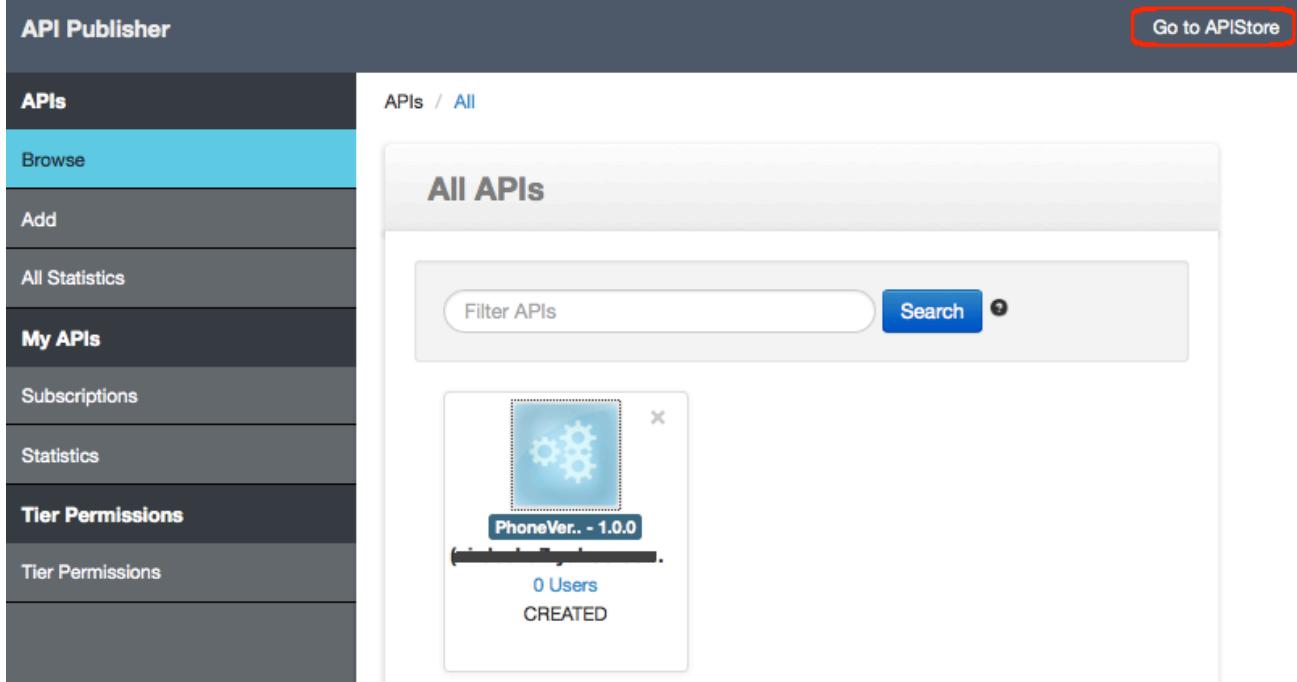
The API Gateway has a default mediation flow for the API invocation requests that it receives. You can extend this

default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a synapse mediation sequence. You can design sequences using a tool like WSO2 Developer Studio and then store the sequence in the Gateway's registry.

Let's see how to convert a message types using custom sequences. In this tutorial, we convert a JSON payload to SOAP before sending it to a SOAP backend. Then we receive the response in SOAP and convert it back to JSON.

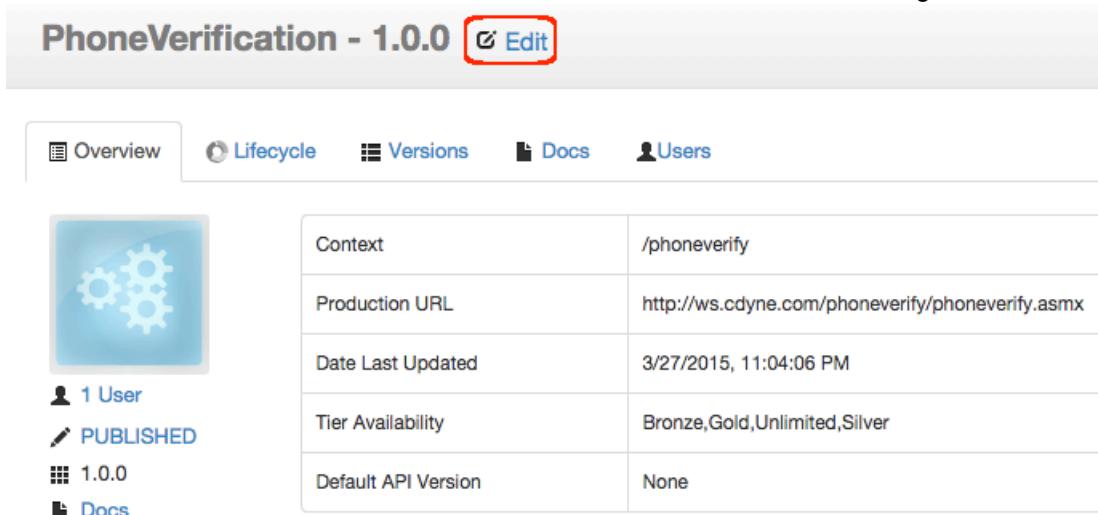
 This tutorial uses the PhoneVerification API that we created in [Create and Publish an API](#).

1. Log in to the API Publisher and click the PhoneVerification API.



The screenshot shows the WSO2 API Manager API Publisher interface. On the left, there is a sidebar with the following menu items: APIs, Browse, Add, All Statistics, My APIs (which is selected and highlighted in blue), Subscriptions, Statistics, Tier Permissions (which is also selected and highlighted in blue), and Tier Permissions. The main area is titled "All APIs" and contains a search bar with "Filter APIs" and "Search" buttons. Below the search bar is a card for the "PhoneVer.. - 1.0.0" API, which has a gear icon, 0 users, and the status "CREATED".

2. Click the Edit link next to the API's name to go to its edit mode.



The screenshot shows the "PhoneVerification - 1.0.0" API edit page. At the top, there is a title bar with the API name and an "Edit" button (which is highlighted with a red box). Below the title bar, there are tabs for Overview, Lifecycle (which is selected and highlighted in blue), Versions, Docs, and Users. On the left, there is a sidebar with icons for 1 User, PUBLISHED, 1.0.0, and Docs. The main content area displays the following details:

Context	/phoneverify
Production URL	<a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>
Date Last Updated	3/27/2015, 11:04:06 PM
Tier Availability	Bronze,Gold,Unlimited,Silver
Default API Version	None

3. Add the following resource to the API.

 **Tip:** The resource you create here invokes the [SOAP 1.2 Web service of the backend](#). Therefore, the recommended method is HTTP POST. As you do not include the payload in a query string, avoid giving any specific name in the URL pattern, which will be amended to the actual backend URL.

Field		Sample value
Resources	URL pattern	/*
	Request types	POST

## API Definition

URL Pattern   GET  POST  PUT  DELETE more

4. After the resource is added, expand it and add a parameter as follows. This parameter is used to pass the payload to the backend.

Parameter name	Description	Parameter Type	Data Type
body	Pass the phone number and license key	body	String

**POST** /\* + Summary

**Description :**  
+ Add Implementation Notes

**Response Content Type :** application/json

**Parameters :**

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
body	Pass the phone number and license key	body	string	True	<input type="button" value="Delete"/>

Parameter Name

Next, let's write a sequence to convert the JSON payload to a SOAP request. We do this because the backend accepts SOAP requests.

5. Navigate to the Implement page and change the endpoint of the API to <http://ws.cdyne.com/phoneverify/phonverify.asmx?WSDL>. Once the edits are done, click **Save**.

1 Design

2 Implement

3 Manage

**PhoneVerification(1.0.0) : /phoneverify/1.0.0****Warning!**

You are editing an API with active subscribers. Tier Availability changes will not be reflected on active subscriptions.

**Managed API**

Provide endpoints of the production API and sandbox API to be managed.

**Endpoints**Endpoint Type:<sup>\*</sup>

HTTP Endpoint

Production Endpoint:

phoneverify/phoneverify.asmx?wsdl

Advanced Options

Test

E.g.,: http://appserver/resource

Sandbox Endpoint:

Advanced Options

Test

E.g.,: http://appserver/resource

[Show More Options](#)[Save](#)[Next : Manage >](#)

6. Download WSO2 Developer Studio (version 3.7.1 is used here) from <http://wso2.com/products/developer-studio/> and open it by double clicking the Eclipse.app file inside the downloaded folder.

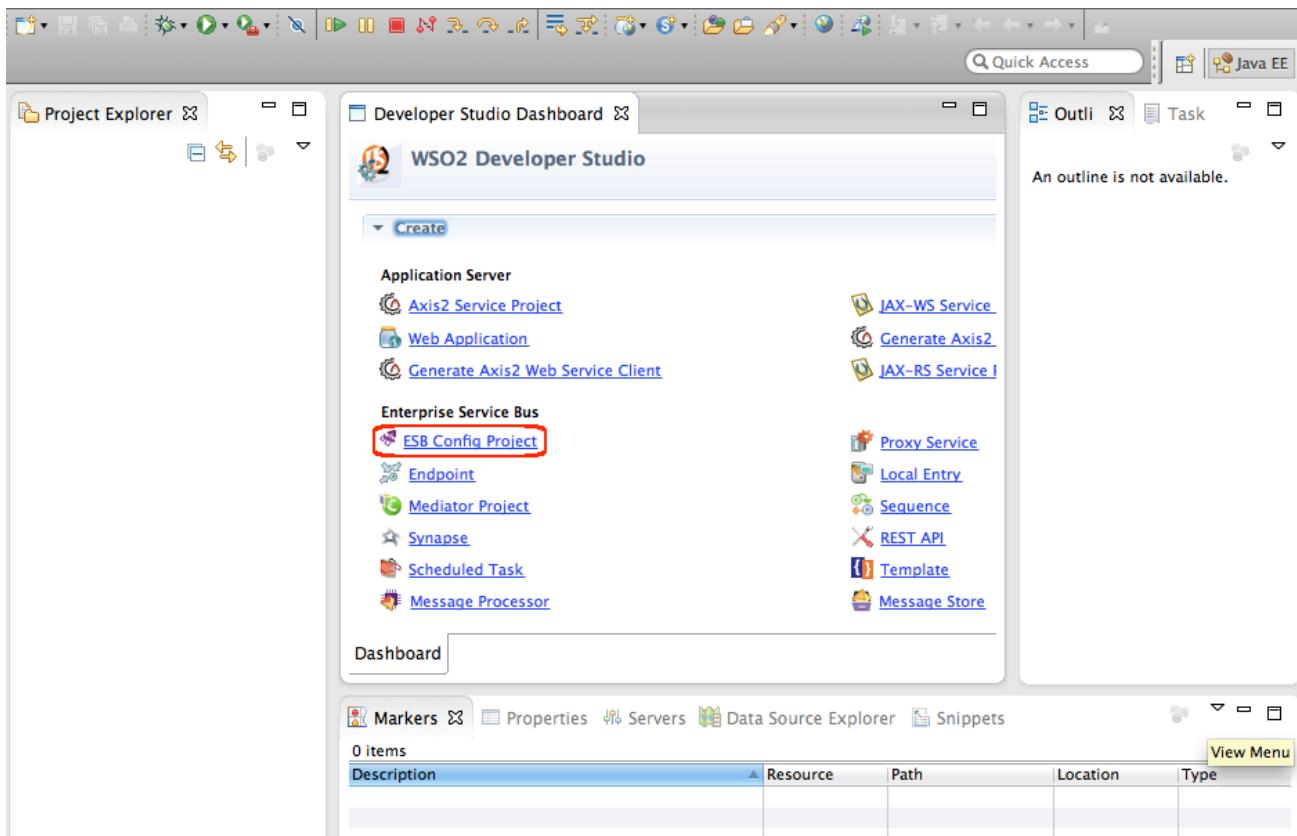


**Tip:** To start Eclipse on a Mac for the first time, open a terminal and execute the following commands:

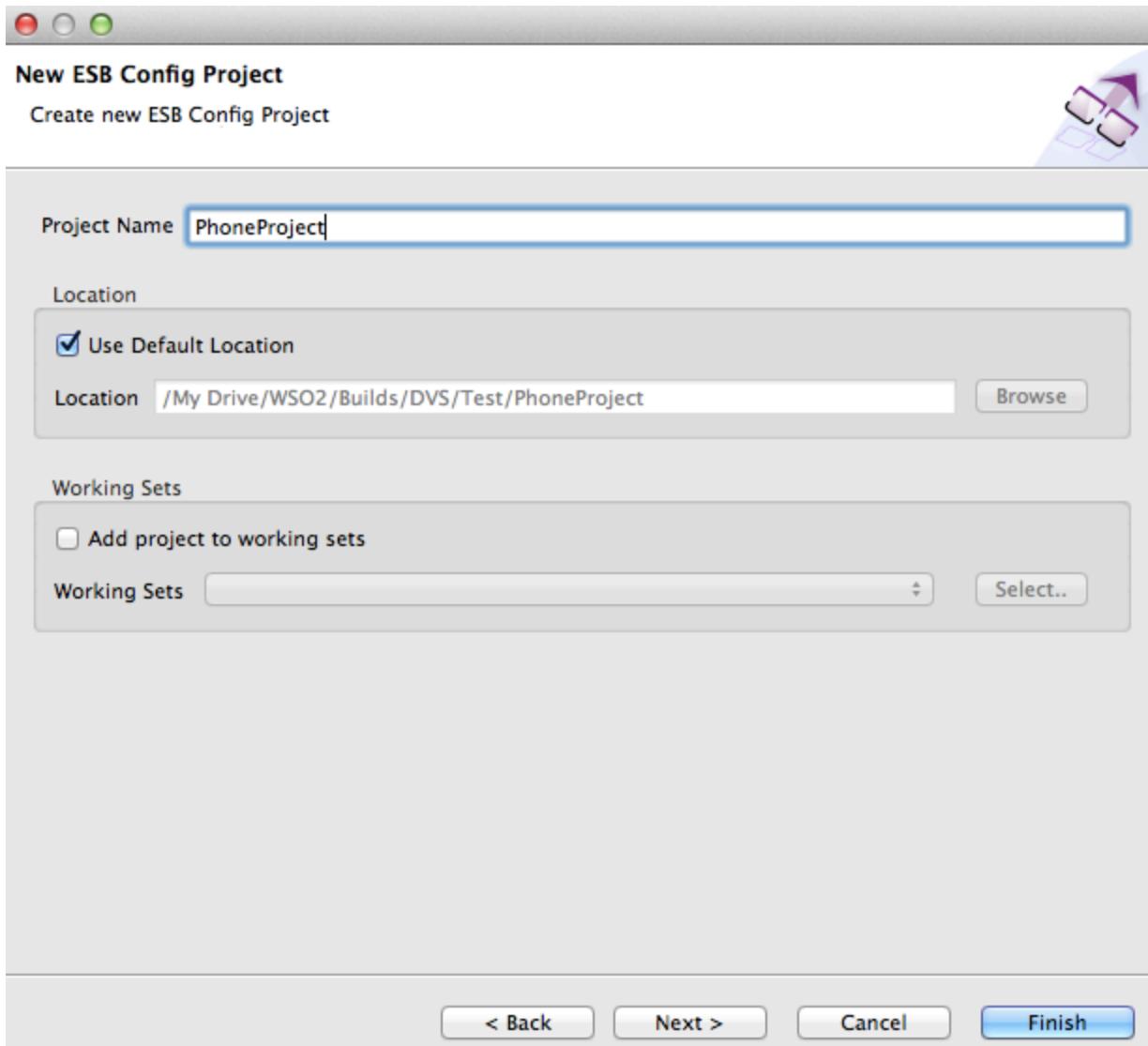
```
cd <DevStudio_Home>/Eclipse.app/Contents/MacOS/
chmod +x eclipse
./eclipse
```

Thereafter, you can start Eclipse by double-clicking the Eclipse icon in <DevStudio\_Home>.

7. Click the **Developer Studio** menu and choose **Open Dashboard**. When the dashboard opens, click the **ESB Config Project link**.



8. Create a new ESB project by the name PhoneProject.



9. Click the **Sequence** link on the Developer Studio Dashboard and create a new sequence by the name **JSON to SOAP in the Phone Project.**

**Create New Sequence**

Give a name for the sequence



Sequence Name:

Save Sequence in:   [Create new Project...](#)

**Advanced Configuration**

On Error Sequence:

Available Endpoints:

Make this as Dynamic Sequence

Registry:

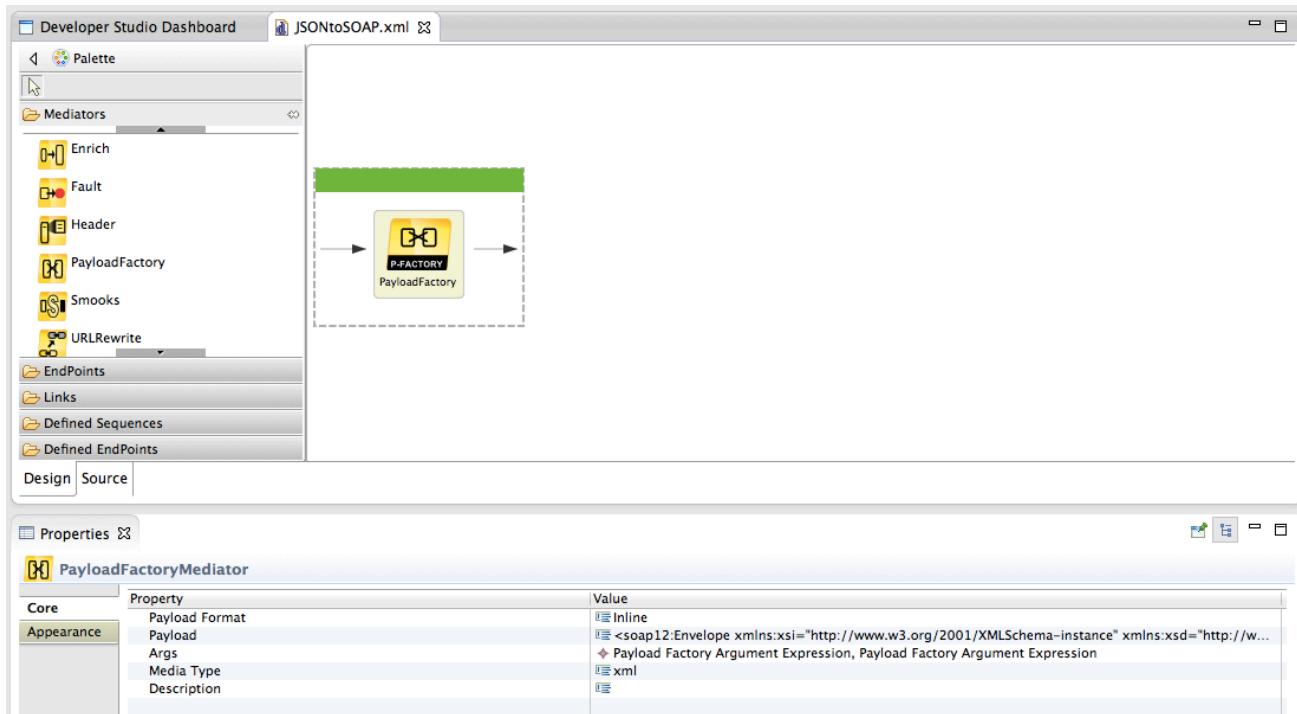
Registry Path:

10. Your sequence now appears on the Developer Studio console. From under the **Mediators** section, drag and drop a **PayloadFactory** mediator to your sequence and give the following values to the mediator.

 **Tip:** The **PayloadFactory** mediator transforms the content of your message. The `<args>` elements define arguments that retrieve values at runtime by evaluating the provided expression against the SOAP body. You can configure the format of the request/response and map it to the arguments.

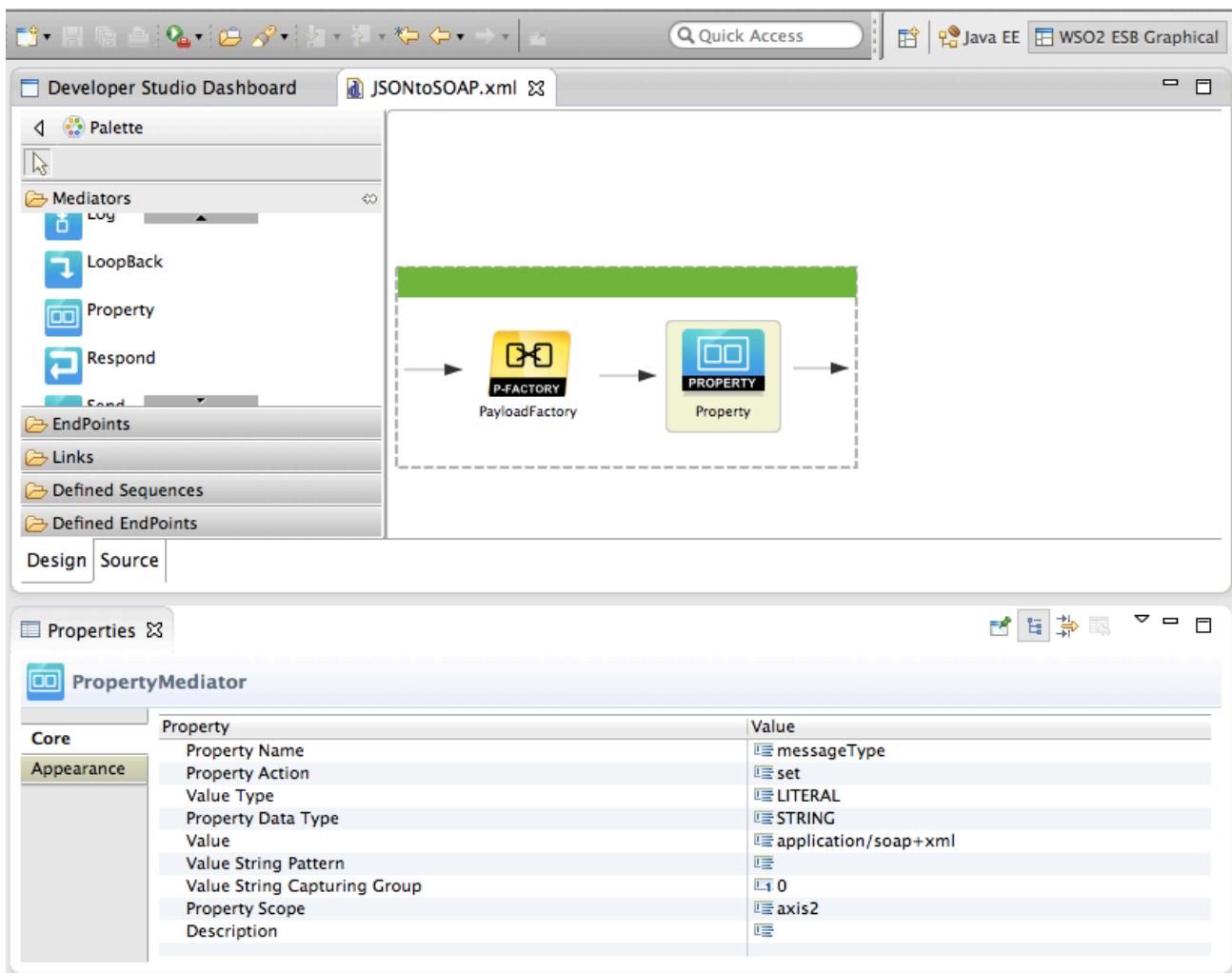
For example, in the following configuration, the values for the format parameters `PhoneNumber` and `LicenseKey` will be assigned with values that are taken from the `<args>` elements (arguments,) in that particular order.

For details on how you got this configuration, see [PayloadFactory Mediator](#) in the WSO2 ESB documentation.



Payload	<pre> &lt;soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xmlns:xsd="http://www.w3.org/2001/XMLSchema"   xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"&gt;   &lt;soap12:Body&gt;     &lt;CheckPhoneNumber       xmlns="http://ws.cdyne.com/PhoneVerify/query"&gt;       &lt;PhoneNumber&gt;\$1&lt;/PhoneNumber&gt;       &lt;LicenseKey&gt;\$2&lt;/LicenseKey&gt;     &lt;/CheckPhoneNumber&gt;   &lt;/soap12:Body&gt; &lt;/soap12:Envelope&gt; </pre>									
Args	<p>Give the following arguments:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Value</th> <th>Evaluator</th> </tr> </thead> <tbody> <tr> <td>expression</td> <td>//request/PhoneNumber</td> <td>xml</td> </tr> <tr> <td>expression</td> <td>//request/LicenseKey</td> <td>xml</td> </tr> </tbody> </table>	Type	Value	Evaluator	expression	//request/PhoneNumber	xml	expression	//request/LicenseKey	xml
Type	Value	Evaluator								
expression	//request/PhoneNumber	xml								
expression	//request/LicenseKey	xml								

11. Similarly, add a **property** mediator to the same sequence and give the following values to the property mediator.



Property Name	messageType
Value Type	Literal
Value	application/soap+xml
Property Scope	axis2

12. Save the sequence, which is in XML format (e.g., `JSONtoSOAP.xml`). This will be the `In` sequence for your API.
13. Go back to the Developer Studio Dashboard, click the **Sequence** link and create another sequence by the name `SOAPtoJSON` in the `Phone Project`.

**Create New Sequence**

Give a name for the sequence



Sequence Name:

Save Sequence in:   [Create new Project...](#)

Advanced Configuration

On Error Sequence:

Available Endpoints:

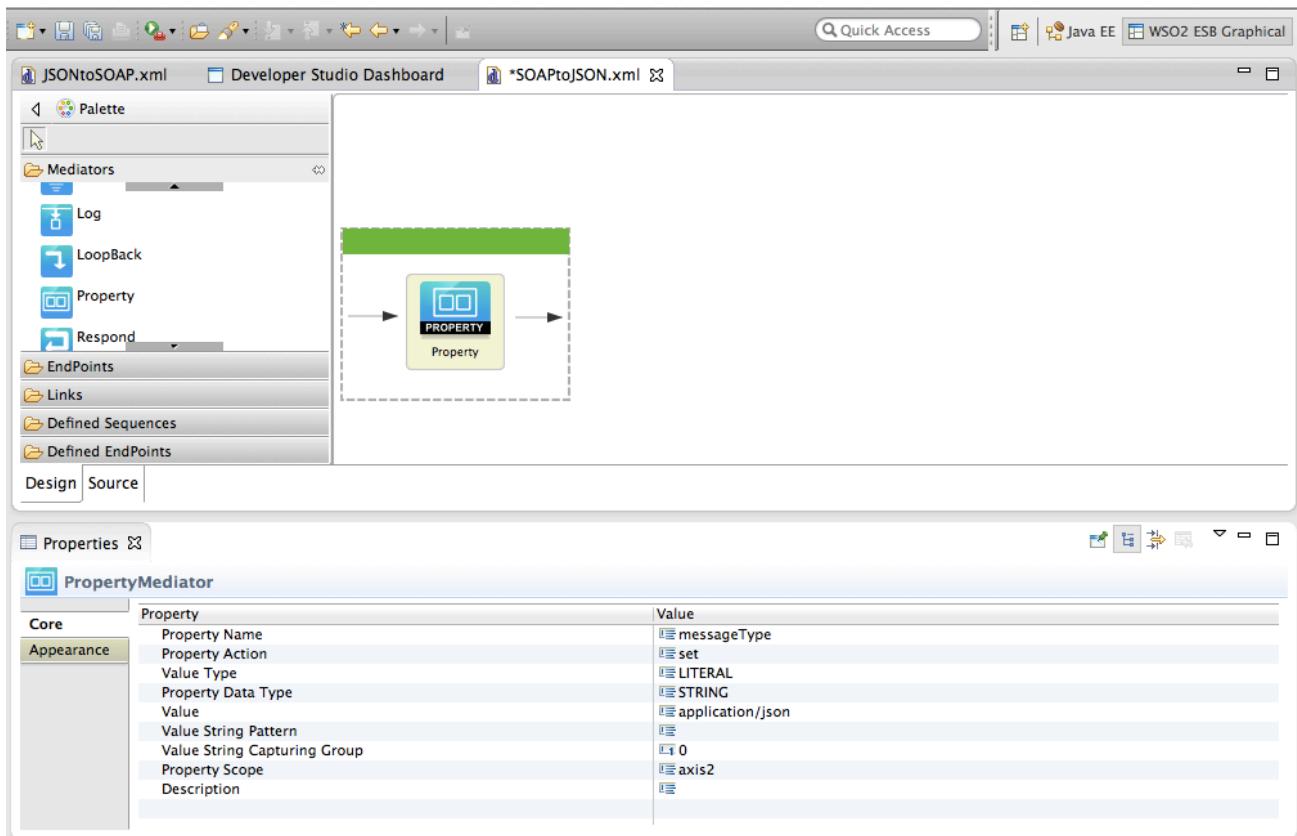
Make this as Dynamic Sequence

Registry:

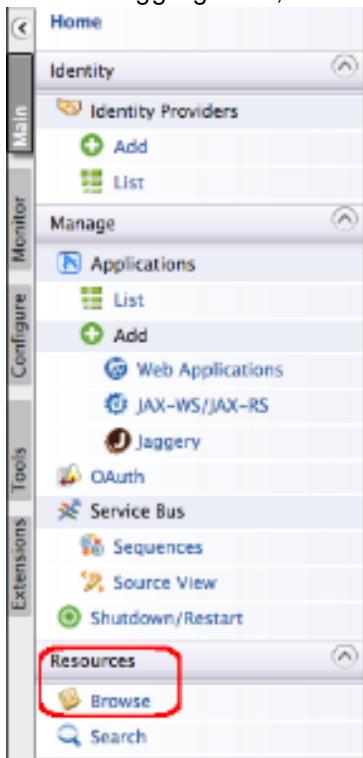
Registry Path:

14. Add a **property** mediator to the second sequence and give the following values to the property mediator.

Property Name	messageType
Value Type	Literal
Value	application/json
Property Scope	axis2



15. Save the sequence, which is in XML format (e.g., `SOAPtoJSON.xml`). This will be the `Out` sequence for your API.
16. Log in to the API Gateway's management console. If you are using WSO2 Cloud, the Gateway URL is <http://gateway.api.cloud.wso2.com/carbon/admin/login.jsp>. If you are using a local setup, the URL is <https://localhost:9443/carbon>. You can see the username on the top right-hand corner of the API Publisher.
17. After logging in, click the **Browse** menu under the **Resources** menu.



18. When the API Gateway's registry opens, navigate to the registry path `/_system/governance/apimgt/customsequences/in`. This is because you want the custom sequence to be invoked in the In direction or the request path.

The screenshot shows the WSO2 API Manager Registry interface. The left sidebar has tabs for Main, Monitor, Configure, Tools, and Extensions. Under Configure, 'Main' is selected. The 'Resources' section contains 'Browse' (which is highlighted with a red box) and 'Search'. The main area is titled 'Browse' and shows a tree view of the registry structure. The path 'Root /' is at the top. Below it, the tree view shows: 
 

- \_system
  - config
  - governance
    - apimgt
      - applicationdata
      - customsequences
        - fault
        - in** (highlighted with a red box)
        - out
      - externalstores

19. Click **Add Resource** and upload the XML file of the sequence that you created earlier.

**Browse**

Root /\_system/governance/apimgt/customsequences/in

Location: /\_system/governance/apimgt/customsequences/in

**Metadata**

**Properties**

**Entries**

Add Resource  
 Add Collection  
 Create Link

**Add Resource**

Method	Upload content from file
File *	<input type="button" value="Choose File"/> JSONtoSOAP.xml Give the path of a file to fetch content (xml,wsdl,jar etc..)
Name *	JSONToSOAP.xml
Media type	application/xml
Description	<input type="text"/>
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

Next, let's write another sequence to convert the SOAP response that the backend sends to JSON.

20. Similarly, navigate to the registry path `/_system/governance/apimgt/customsequences/out` and upload the `SOAPtoJSON.xml` sequence file. This will invoke the second sequence in the Out direction or the `response path`.

**Browse**

Root [/\\_system/governance/apimgt/customsequences/out](/_system/governance/apimgt/customsequences/out)

Location: /\_system/governance/apimgt/customsequences/out

**Metadata**

**Properties**

**Entries**

Add Resource  
 Add Collection  
 Create Link

**Add Resource**

Method

**Upload Content From File**

File *	<input type="button" value="Choose File"/> SOAPtoJSON.xml Give the path of a file to fetch content (xml,wsdl,jar etc..)
Name *	SOAPtoJSON.xml
Media type	application/xml
Description	<input type="text"/>

21. Log back to the API Publisher, click the **Edit** link associated with the API, navigate to the **Manage** tab, click the **Sequences** check-box and engage the In and out sequences that you created earlier.

1 Design    2 Implement    3 Manage

## PhoneVerification : /phoneverify/1.0.0

### Configurations

Make this default version  [?](#)  
No default version defined for the current API

Tier Availability\*: **4 selected** [?](#)

Transports\*:  HTTP  HTTPS

Sequences:  Check to select a custom sequence to be executed in the message flow

In Flow	Out Flow	Fault Flow
JSON	SOAP1	Nc

22. **S a v e** the API.  
You have created an API, a resource to access the SOAP backend and engaged sequences to the request and response paths to convert the message format from JSON to SOAP and back to JSON. Let's subscribe to the API and invoke it.

23. Log in to the API Store and subscribe to the API and create an access token if you have not done so already.

WSO2 API STORE

APIs Prototyped APIs My Applications My Subscriptions Forum Statistics Themes admin

Search API Go to Public API Store

More APIs from 'admin'

### PhoneVerification - 1.0.0

admin

	Rating: Your rating: N/A 
Version:	1.0.0
Status:	PUBLISHED
Updated:	01/Jun/2015 13:41:14 PM IST

Applications: DefaultApplication

Tiers: Unlimited

Allows unlimited requests

Subscribe

Overview Documentation API Console Throttling Info Forum

24. Go to the API Console tab and expand the POST method.

The screenshot shows the WSO2 API Manager dashboard. At the top, there are links for 'API STORE', 'APIs', 'Prototyped APIs', 'My Applications', 'My Subscriptions', 'Forum', 'Statistics', 'Themes', and 'TestUser'. Below the header is a search bar labeled 'Search API' with a magnifying glass icon. A sidebar on the left says 'More APIs from 'admin''.

The main content area displays the 'PhoneVerification - 1.0.0' API. It includes a profile picture for 'admin', a rating of 'N/A' (no stars), a version of '1.0.0', a status of 'PUBLISHED', and an updated date of '01/Jun/2015 14:33:38 PM IST'. To the right, there are dropdown menus for 'Applications' (set to 'Select Application...'), 'Tiers' (set to 'Unlimited'), and 'Allows unlimited requests'. A 'Subscribe' button is also present.

The navigation tabs at the top of the API details page are 'Overview', 'Documentation', 'API Console' (which is highlighted with a red box), and 'Throttling Info'. Below these tabs, there are dropdowns for 'Try' (set to 'DefaultApplication') and 'On' (set to 'Production'). There is also a dropdown for 'Environment'. Underneath, there is a 'Set Request Header' section with an 'Authorization' field containing 'Bearer b04048e086da73f5e88ca43b889aecd'.

The 'PhoneVerification' API console shows a green 'POST' button and a red box highlights the URL '/'. To the right, there is a link to 'Swagger (/swagger.json)' and buttons for 'Show/Hide', 'List Operations', and 'Expand Operations'.

25. Give the payload in the `body` parameter in JSON format and click **Try it out**. Here's a sample JSON payload:
- ```
{"request":{"PhoneNumber":"18006785432","LicenseKey":"0"}}
```

The screenshot shows the 'Parameters' section of the API console. It has a 'POST' button and a URL '/'. Below is a table for parameters:

| Parameter | Value                                                                     | Description                           | Parameter Type | Data Type |
|-----------|---------------------------------------------------------------------------|---------------------------------------|----------------|-----------|
| body      | <pre>{"request": {"PhoneNumber": "18006785432", "LicenseKey": "0"}}</pre> | Pass the phone number and license key | body           | string    |

Below the table, it says 'Parameter content type:' with a dropdown set to 'application/json'.

The 'Response Messages' section shows a table with columns for 'HTTP Status Code', 'Reason', 'Response Model', and 'Headers'. A row for '200' has a 'Try it out!' button (highlighted with a red box) and a 'Hide Response' link.

26. Note that you get a JSON response to the JSON request whereas the backend accepts SOAP messages. The request and response are converted by the sequences that you engaged at the API Gateway.

## Response Body

```
{
  "CheckPhoneNumberResponse": {
    "CheckPhoneNumberResult": {
      "Company": "Toll Free",
      "Valid": true,
      "Use": "Assigned to a code holder for normal use.",
      "State": "TF",
      "RC": null,
      "OCN": null,
      "OriginalNumber": 18006785432,
      "CleanNumber": 8006785432,
      "SwitchName": null,
      "SwitchType": null,
      "Country": "United States",
      "CLLI": null,
      "PrefixType": "Landline",
      "LATA": null,
      "sms": "Landline",
      "Email": null,
      "AssignDate": null,
      "TelecomCity": null,
    }
  }
}
```

In this tutorial, you converted a message from JSON to SOAP and back to JSON using `In` and `Out` sequences.

## Publish through Multiple API Gateways

You can configure multiple API Gateway environments that publish to a single API Store. It helps you distribute the API Gateway load to multiple nodes and also gives you some logical separation (e.g., production vs sandbox) between the APIs in the API Store. Once you publish an API through multiple Gateway environments, the APIs in the API Store will have different server hosts and ports.

The steps below explain how to configure and publish to multiple Gateways. In this guide, we set up 3 API Manager instances in the same server. In a typical production environment, the Gateways will ideally be in separate servers.

- **Instance 1:** Acts as the node that provides the API Publisher, Store and the Key Manager functionality.
- **Instance 2:** Acts as a production Gateway node.
- **Instance 3:** Acts as a sandbox Gateway node.

1. Copy the API Manager product pack in 3 separate folders.  
Let's add offsets to the default ports of the two Gateway instances. A port offset ensures that there are no port conflicts when more than one WSO2 products run on the same server.
2. Open the `<APIM_HOME>/repository/conf/carbon.xml` file in the **second** API Manager instance and add an offset of 1 to its default port. This increments its default server port, which is 9443, by 1.

```
<Offset>1</Offset>
```

3. Open the `<APIM_HOME>/repository/conf/carbon.xml` file in the **third** API Manager instance and add an offset of 2 to its default port. This increments its default server port, which is 9443, by 2.

```
<Offset>2</Offset>
```

The two Gateway instances need to communicate with the Key Manager in the first API Manager instance.

4. Open the <APIM\_HOME>/repository/conf/api-manager.xml files in the **first and the second** Gateway instances and change the following:

```
<AuthManager>
    <ServerURL>https://<IP of the first instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
</AuthManager>
...
<APIKeyValidator>
    <ServerURL>https://<IP of the first instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
    <RevokeAPIURL>https://<IP of the first instance>:8243/revoke</RevokeAPIURL>
</APIKeyValidator>
```

You are done configuring the two API Gateway instances. Let's point to them from the first instance.

- 5.
6. Open the <APIM\_HOME>/repository/conf/api-manager.xml file in the **first** API Manager instance, add two API Gateway environments under the <Environments> element and delete the <environment> that comes by default. For example:

```
<Environments>
    <Environment type="production">
        <Name>Production Gateway</Name>
        <Description>Production Gateway Environment</Description>
        <ServerURL>https://localhost:9444/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>

        <GatewayEndpoint>http://localhost:8281,https://localhost:8244</GatewayEndpoint>
    </Environment>
    <Environment type="sandbox">
        <Name>Sandbox Gateway</Name>
        <Description>Sandbox Gateway Environment</Description>
        <ServerURL>https://localhost:9445/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>

        <GatewayEndpoint>http://localhost:8282,https://localhost:8245</GatewayEndpoint>
    </Environment>
</Environments>
```



**Tip:** The Gateway environment names must be unique.

7. Start all instances.
8. Log in to the API Publisher (first instance) and choose to edit an API.

**PhoneVerification - 1.0.0** 

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#)

	Context	/phoneverify
 1 User	Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
 PUBLISHED	Date Last Updated	3/27/2015, 11:04:06 PM
 1.0.0	Tier Availability	Bronze,Gold,Unlimited,Silver
 Docs	Default API Version	None

9. Navigate to the **Manage** tab, expand the **Gateway Environments** section and note that the two Gateway environments listed there.

 1 Design 2 Implement 3 Manage

### Manage API: PhoneVerification : /phoneverify/1.0.0

#### Configurations

Make this default version  ⓘ No default version defined for the current API

Tier Availability: \*  4 selected

Transports: \*  HTTPS  HTTP ⓘ

Sequences:  Check to select a custom sequence to be executed in the message flow

Response Caching:  Disabled ⓘ

#### Gateway Environments

Environment Name	Type	Description
<input checked="" type="checkbox"/> Production Gateway	production	Production Gateway Environment
<input checked="" type="checkbox"/> Sandbox Gateway	sandbox	Sandbox Gateway Environment

In a typical production setup, you will publish the API to the sandbox Gateway first and then publish it to the production Gateway. In this case, let's publish to both.

10. Select both Gateways and **Save and Publish** the API.

11. Log in to the API Store (of the first instance) and select click the API to open it.

12. In the API's Overview tab, note that it has two sets of URLs for the two Gateway instances:

## PhoneVerification - 1.0.0

admin

	Version:	1.0.0
	Status:	PUBLISHED
	Updated:	09/Jun/2015 13:48:54 PM IST

**Overview** [Documentation](#) [API Console](#) [Throttling Info](#)

**SANDBOX Endpoints**

**Sandbox Gateway URLs:**

- `https://localhost:8245/phoneverify/1.0.0`
- `http://localhost:8282/phoneverify/1.0.0`

**PRODUCTION Endpoints**

**Production Gateway URLs:**

- `https://localhost:8244/phoneverify/1.0.0`
- `http://localhost:8281/phoneverify/1.0.0`

You have published an API to the API Stores through multiple Gateway environments.

## Enforce Throttling and Resource Access Policies

**Throttling** allows you to limit the number of hits to an API during a given period of time, typically to protect your APIs from security attacks and your backend services from overuse, regulate traffic according to infrastructure limitations and to regulate usage for monetization. For information on different levels of throttling in WSO2 Cloud, see [Throttling tiers](#).

**⚠** This tutorial uses the PhoneVerification API, which has one resource, GET and POST methods to access it and a throttling policy enforced.

**Before you begin**, follow the

[Create and Publish an API](#) to create and publish the PhoneVerification API and then the

[Invoke an API using cURL](#) to subscribe to the API using the Bronze throttling tier.

After you created, published and subscribed to the API, let's see how the API Gateway enforces throttling and resource access policies to the API.

1. Log in to the API Store and select the PhoneVerification API.
2. Subscribe to the API using the **Bronze tier** if you haven't done so already.

The screenshot shows the WSO2 API Store interface. On the left, there's a sidebar with a list of APIs: PhoneVerification-2.0.0, TestAPI1-1.0.0, and WeatherAPI-1.0.0. The main area displays the PhoneVerification - 1.0.0 API details. It includes fields for Rating (N/A), Version (1.0.0), Status (PUBLISHED), and Updated (20/May/2015 12:15:56 PM IST). On the right, there are dropdown menus for Applications (DefaultApplication) and Tiers (Bronze). A tooltip for the Tiers dropdown states "Allows 1 request(s) per minute." A blue "Subscribe" button is located at the bottom right of the API details section.

3. Choose to go to the **My Subscriptions** page and generate an access token. If you already have an access token for the application, you have to regenerate it after 1 hour.

Screenshot of the WSO2 API Manager interface showing the 'My Subscriptions' section. The 'My Subscriptions' tab is highlighted with a red box. The 'DefaultApplication' dropdown is also highlighted with a red box. A 'C Regenerate' button is highlighted with a red box.

**Subscriptions**

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

**Applications With Subscriptions**

DefaultApplication

Show Keys

**Keys - Production**

Consumer Key : khTYzxdAo12VakSPG6oJujHolUca

Consumer Secret : WWBdXT05c0vELSmY0POz\_xNYPgca

Access Token: ffbe86dd1ca2b36b17284efc4273ee

cURL ▾ Validity Time: 3600 Seconds C Regenerate

Allowed Domains ALL

The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.

Update Domains

**PhoneVerification - 1.0.0**

admin

Rating: Your rating: N/A

Version: 1.0.0

Status: PUBLISHED

Updated: 20/May/2015 12:15:56 PM IST

Applications Select Application...

Tiers Bronze

Allows 1 request(s) per minute.

Subscribe

Overview Documentation API Console Throttling Info

Try DefaultApplication On Production Environment.

Set Request Header Authorization : Bearer 948a27eed6787a5166ef673f49daac8

**PhoneVerification**

GET /CheckPhoneNumber

POST /CheckPhoneNumber

5. Give values to the parameters and click **Try it out** to invoke the API.

**GET /CheckPhoneNumber**

### Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

### Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

**Try it out!**

6. Note the response that appears in the API Console. As we used a valid phone number in this example, the response returns as valid.

### Response Body

```
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query"> <Company>Toll Free</Company>
<Valid>true</Valid>
<Use>Assigned to a code holder for normal use.</Use>
<State>TF</State>
<RC />
<OCN />
<OriginalNumber>18006785432</OriginalNumber>
<CleanNumber>8006785432</CleanNumber>
<SwitchName />
<SwitchType />
<Country>United States</Country>
<CLLI />
<PrefixType>Landline</PrefixType>
<LATA />
<sms>Landline</sms>
<Email />
<AssignDate />
<TelecomCity />
<TelecomCountry />
```

7. Within a minute after the first API invocation, make another attempt to invoke the API.  
 8. Note that you get a throttling error saying that you exceeded your quota. This is because you subscribed to the API on the Bronze throttling tier and the Bronze tier only allows you to make one call to the API per

m i n u t e .

**Response Body**

```
<amt:fault xmlns:amt="http://wso2.org/apimanager/throttling">
<amt:code>900800</amt:code>
<amt:message>Message Throttled Out</amt:message>
<amt:description>You have exceeded your quota</amt:description>
</amt:fault>
```

Let's try to invoke the API using an unavailable resource name.

9. Go to the API's **Overview** page in the API Store and get the API's URL.

**PhoneVerification - 1.0.0** admin**Rating:** Your rating: N/A**Version:** 1.0.0**Status:** PUBLISHED**Updated:** 20/May/2015 12:15:56 PM IST**Overview****Documentation****API Console****Throttling Info****Production and Sandbox URLs:**

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

**Share:****Social Sites****Embed****Email**

10. Install [cURL](#) or any other REST client.

11. Go to the command-line invoke the API using the following cURL command.

```
curl -k -H "Authorization :Bearer <access token in step 3>" '<API's URL in step 9>/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

Note that the PhoneVerification API's resource name is **CheckPhoneNumber**, but we use an undefined resource name as **CheckPhoneNum**. Here's an example:

```
curl -k -H "Authorization :Bearer 633d6db88e6ee42457e60ad1b736210"
'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

12. Note that the call gets blocked by the API Gateway with a 'no matching resource' message. It doesn't reach your backend services as you are trying to access a REST resource that is not defined for the API.

```
$ curl -k -H "Authorization :Bearer 8eaf3ade3da7698a2fcff143e144103b" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
<ams:fault xmlns:ams="http://wso2.org/apimanager/security"><ams:code>900906</ams:code><ams:message>No matching resource found in the API for the given request</ams:message><ams:description>Access failure for API: /phoneverify/1.0.0, version: 1.0.0 with key: 8eaf3ade3da7698a2fcff143e144103b. Check the API documentation and add a proper REST resource path to the invocation URL</ams:description></ams:fault>
```

You have seen how the API Gateway enforces throttling and resource access policies for APIs.

#### Include Additional Headers in the API Console

The Swagger API Console is a JavaScript client that runs in the API Store and makes JavaScript calls from the Store to the API Gateway. Since the API Store and Gateway run on two different ports, we have enabled cross-origin resource sharing (CORS) between the two. You must specify any additional headers that you want to add to the API Console under the CORS configuration.

Open the CORS configuration in <APIM\_HOME>/repository/conf/api-manager.xml file, enable CORS if it is not enabled already and specify the additional headers (Content-Type and SOAPAction, in this case) under the <Access-Control-Allow-Headers> element:

#### CORS configurations in api-manager.xml

```
<CORSConfiguration>
    <Enabled>true</Enabled>
    <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>

    <Access-Control-Allow-Methods>GET,PUT,POST,DELETE,OPTIONS</Access-Control-Allow-Methods>
    <Access-Control-Allow-Headers>tenant, enterprise, authorization,
    Access-Control-Allow-Origin, Content-Type, SOAPAction</Access-Control-Allow-Headers>
</CORSConfiguration>
```

Next, let's see how to add the two headers as parameters to the API Console.

1. Log in to the API Publisher and click the API that you want to invoke (e.g., PhoneVerification).
2. Click the **Edit** link next to the API's name, navigate down to the **Resources** section and click on the **POST method** to expand it.

## Resources

URL Pattern	<input type="text" value="/{context}/{version}/"/>	Url Pattern Ex: path/to/resource
	<input type="checkbox"/> GET	<input type="checkbox"/> POST
	<input type="checkbox"/> PUT	<input type="checkbox"/> DELETE
	<input type="checkbox"/> OPTIONS	
Resource Name	<input type="text" value="Resource"/>	
<input type="button" value="Add New Resource"/>		

### /checkphonenumer

<b>GET</b>	<a href="#">/CheckPhoneNumber</a> + Summary	
<b>POST</b>	<a href="#">/CheckPhoneNumber</a> + Summary	

3. Create the following headers under the POST method using the **Add Parameter** button.

Parameter name	Values
SOAPAction	Description: Set to <a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a> Parameter Type: Header Data Type: String Required: False
Content-Type	Description: Set to text/xml Parameter Type: Header Data Type: String Required: False

<b>POST</b>	<a href="#">/CheckPhoneNumber</a> + Summary																					
<b>Implementation Notes :</b> <a href="#">+ Add Implementation Notes</a>																						
<b>Response Content Type :</b> <a href="#">text/xml</a>																						
<b>Parameters :</b> <table border="1"> <thead> <tr> <th>Parameter Name</th> <th>Description</th> <th>Parameter Type</th> <th>Data Type</th> <th>Required</th> </tr> </thead> <tbody> <tr> <td>body</td> <td><a href="#">Request Body</a></td> <td>body</td> <td>string</td> <td>true</td> </tr> <tr> <td>SOAPAction</td> <td>Set to <a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a></td> <td>header</td> <td>String</td> <td>false</td> </tr> <tr> <td>Content-Type</td> <td>Set to text/xml</td> <td>header</td> <td>String</td> <td>false</td> </tr> </tbody> </table>			Parameter Name	Description	Parameter Type	Data Type	Required	body	<a href="#">Request Body</a>	body	string	true	SOAPAction	Set to <a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a>	header	String	false	Content-Type	Set to text/xml	header	String	false
Parameter Name	Description	Parameter Type	Data Type	Required																		
body	<a href="#">Request Body</a>	body	string	true																		
SOAPAction	Set to <a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a>	header	String	false																		
Content-Type	Set to text/xml	header	String	false																		
<input type="text" value="Parameter Name"/>		<input type="button" value="Add Parameter"/>																				

4. Once you are done, click **Save**.  
 5. Log in to the API Store, subscribe to the API and generate an access token. If it's an API that you are already subscribed to, you might have to renew the access token from the **My Subscriptions** page.

The screenshot shows the 'My Subscriptions' tab selected in the top navigation bar. A search bar is at the top left, and a magnifying glass icon is at the top right. Below the header, the word 'Subscriptions' is displayed in a large, bold font. A sub-header 'Applications With Subscriptions' is followed by a dropdown menu set to 'DefaultApplication'. The main content area is titled 'Keys - Production'. It displays three sets of credentials: 'Consumer Key' (OeOn7gfCVXcZ\_DYnAyuRxlvpriEa), 'Consumer Secret' (8PhC0iPY5VTSrq\_7LRUrQ3tjPla), and 'Access Token' (a37b44f1863cbe60ab6282f89dc31635). Each credential has a copy icon to its right. Below these fields are buttons for 'CURL' (dropdown), 'Validity Time' (set to 3600 seconds), and a prominent blue 'Re-generate' button. To the right, there is a section titled 'Allowed Domains' with a dropdown set to 'ALL'. A note below it says 'The domains from which requests are accepted' and includes a 'Update Domains' button.

6. Click on the API again to open it and then click its **API Console** tab.

## PhoneVerification - 1.0.0

 admin



Rating:	Your rating: N/A 
Version:	1.0.0
Status:	PUBLISHED
Updated:	17/Feb/2015 16:44:34 PM IST

Applications

Select Application...

Tiers

Bronze

Allows 1 request(s) per minute.

[Subscribe](#)
[Overview](#) [Documentation](#) [API Console](#) [Throttling Info](#) [Forum](#)

Try  On  Enviromant.

[Set Request Header](#) Authorization : Bearer a37b44f1863cbe60ab6282f89dc31635

Verify a phone number

[Swagger Resource Li](#)

checkphonenumber :

[Show/Hide](#) [List Operations](#) [Expand Op](#)

[GET](#) /CheckPhoneNumber

[POST](#) /CheckPhoneNumber

7. Expand the POST method, fill the parameter values and invoke the API. For example,

Parameter	Value
Body	<p>This is the example SOAP request that we copied from the SOAP UI of the previous tutorial:</p> <pre>&lt;soapenv:Envelope   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"   xmlns:quer="http://ws.cdyne.com/PhoneVerify/query"&gt;   &lt;soapenv:Header/&gt;   &lt;soapenv:Body&gt;     &lt;quer:CheckPhoneNumber&gt;       &lt;!--Optional:--&gt;       &lt;quer:PhoneNumber&gt;123456&lt;/quer:PhoneNumber&gt;       &lt;!--Optional:--&gt;       &lt;quer:LicenseKey&gt;0&lt;/quer:LicenseKey&gt;     &lt;/quer:CheckPhoneNumber&gt;   &lt;/soapenv:Body&gt; &lt;/soapenv:Envelope&gt;</pre>
SOAPAction	<a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a>
Content-Type	text/xml

POST /CheckPhoneNumber

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
body	<pre>&lt;soapenv:Envelope   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"   xmlns:quer="http://ws.cdyne.com/PhoneVerify/query"&gt;</pre>	Request Body	body	string
Parameter content type:	application/json			
SOAPAction	http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber	Set to http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber	header	String
Content-Type	text/xml	Set to text/xml	header	String

**Try it out!**

8. Note the result that appears on the console. As you gave an invalid phone number here, the result is invalid.

You have added SOAP parameters to the API Console and invoked a SOAP service using the API Console.

## API Consumer

See the following topics:

- [Invoke an API using a SOAP Client](#)
- [Invoke an API using the Integrated API Console](#)
- [Subscribe to an API](#)
- [Use the Community Features](#)

### Invoke an API using a SOAP Client

You can use any SOAP client to **invoke an API**. We use the SOAP UI in this example.



See the following topics for a description of the concepts that you need to know when invoking an API:

- [Applications](#)
- [Throttling](#)
- [Access tokens](#)

Create and Publish an APIThe examples here use the PhoneVerification API, which is created in section .

Let's invoke the PhoneVerification API using a SOAP client.

1. Log in to the API Store and click an API that you want to invoke (e.g., PhoneVerification).
2. The API's **Overview** page opens. Select an application, the **Bronze tier** and subscribe to the API.

**PhoneVerification - 1.0.0**

**admin**

**Rating:** Your rating: N/A  
★★★★★

**Version:** 1.0.0

**Status:** PUBLISHED

**Updated:** 14/Aug/2014 14:19:47 PM IST

**Applications:** DefaultApplication

**Tiers:** Bronze

Allows 1 request(s) per minute.

**Subscribe**

**Overview** **Documentation** **API Console** **Throttling Info** **Forum** **Subscribe**

3. Go to the **My Subscriptions** page and generate a production key to the default application using which you subscribed to the API.

**Subscriptions**

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times

**Applications With Subscriptions**

**DefaultApplication**

**Access Token**  
U9znDo4OSYPfzoW16S2puHmKahga

**Re-generate** Token Validity: 3600 Seconds

**Consumer Key**  
LQuF3UNG5AlIJYoUCfKN8Gz\_bWEa

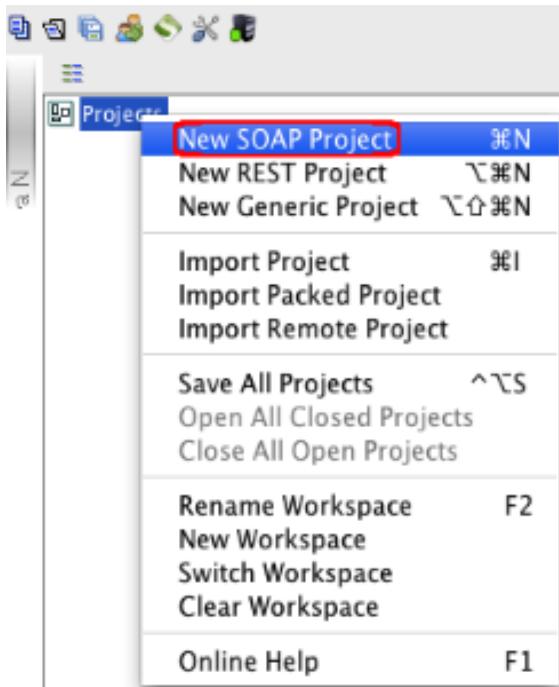
**Consumer Secret**  
Vn2hCdAOhRbbGspmnCKWlfcn08oa

**Allowed Domains**  
ALL

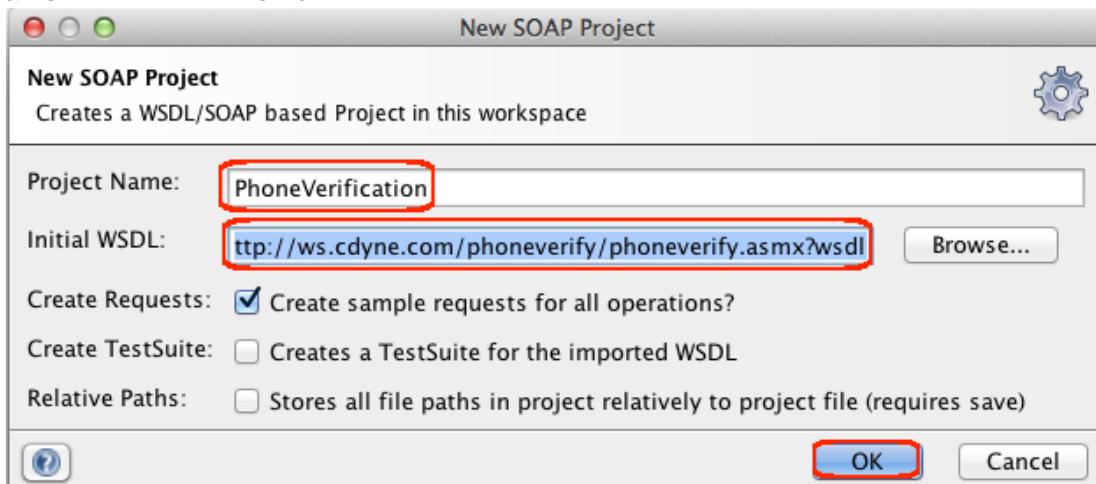
Leave empty or filling with wildcards

**Update**  
**Domains**

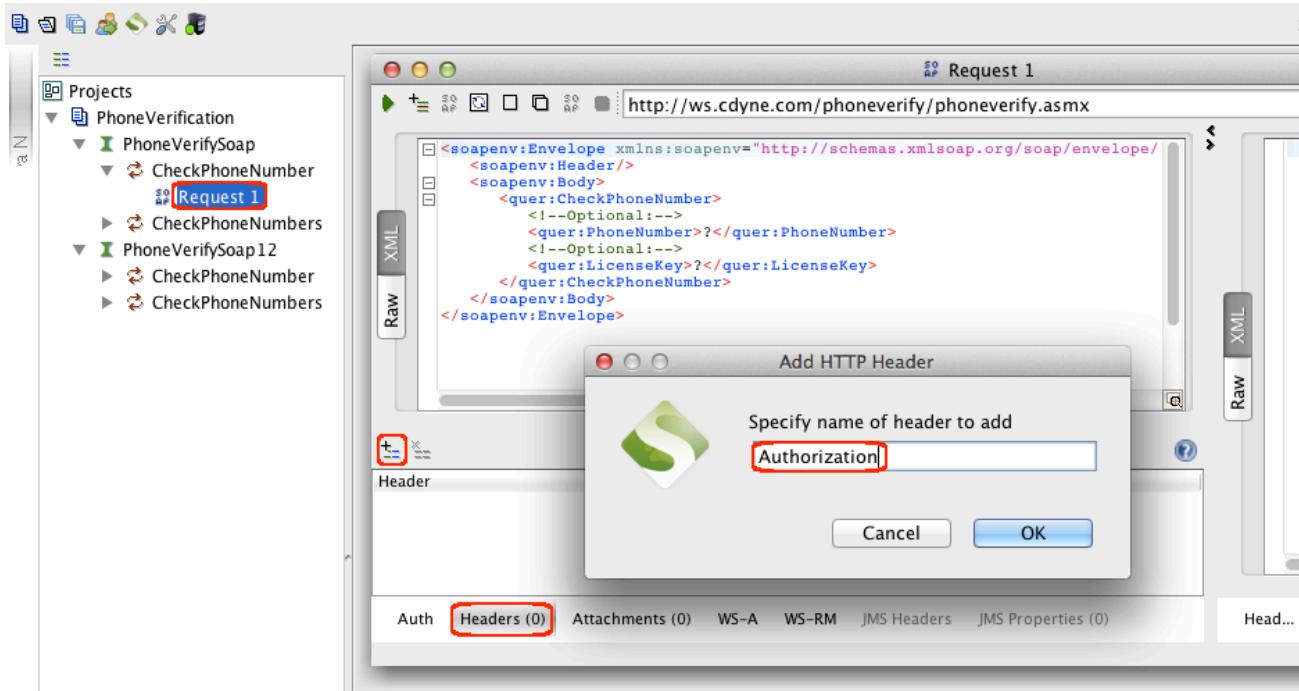
4. Copy the access token to the clipboard as you need it later to invoke the API.  
 5. Download the SOAP UI installation that suits your operating system from <http://www.soapui.org/> and open its console.  
 6. In the SOAP UI, right click on the **Projects** menu and create a new SOAP project.



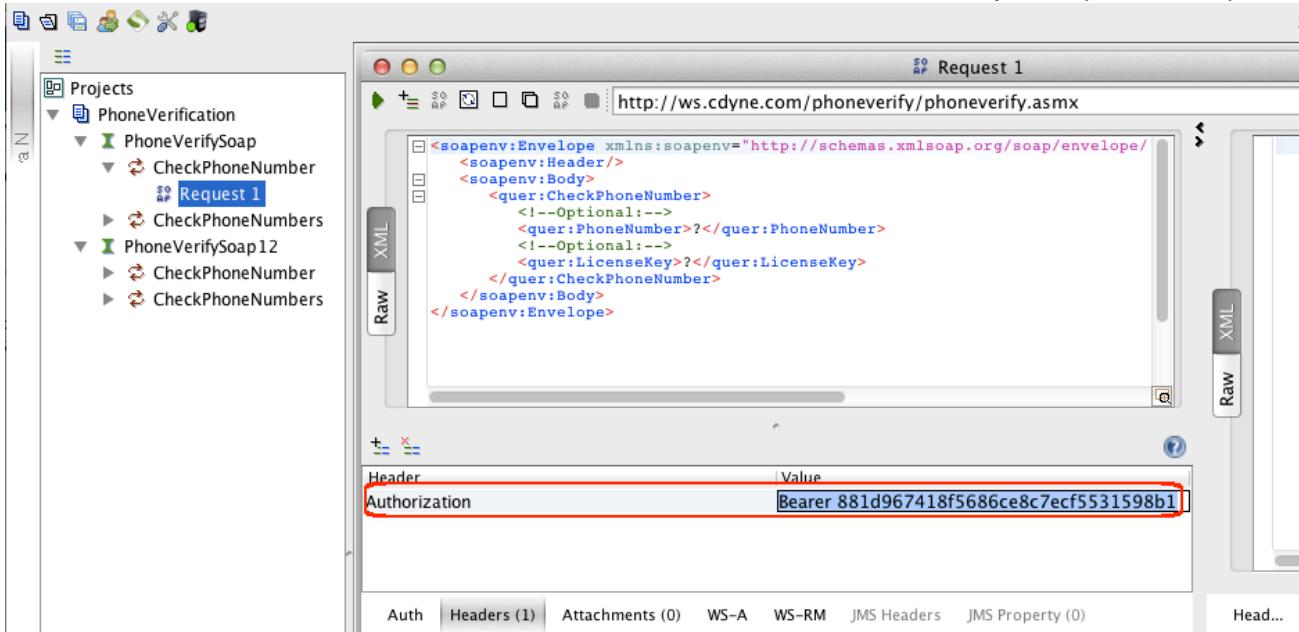
7. Give your API's WSDL and click **OK**. In this case, the WSDL is <http://ws.cdyne.com/phoneverify/phoneverify.wsdl>.



8. The WSDL defines two operations. Let's work with CheckPhoneNumber. Double click on Request 1. Then, add an authorization header to your request by clicking the add sign on the **Header** tab of the console.

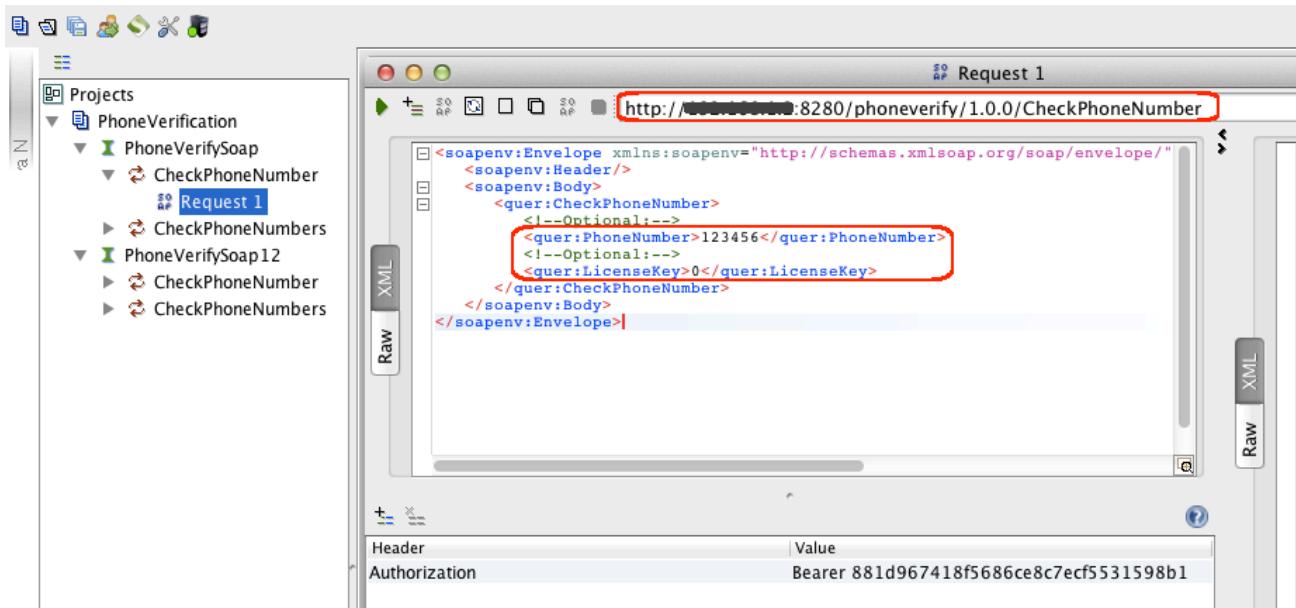


9. Give the value of the Authorization header as 'Bearer <the access token you copied in step 5>.

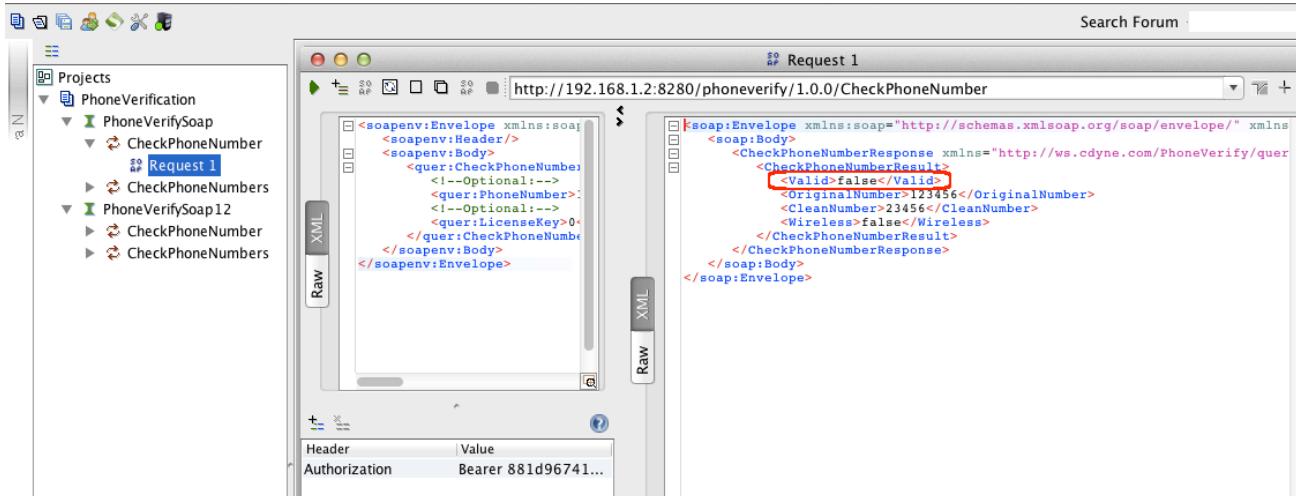


10. Add the following values and submit the request:

- Change the endpoint with the production URL of the API. You can copy the production URL from the API's **Overview** tab in the API Store. Append the resources to the end of the URL, if any. The resource is /CheckPhoneNumber for the PhoneVerification API that we use here.
- In the SOAP request, change the parameters, which are PhoneNumber and LicenseKey. Let's give any dummy phone number and 0 as the license key



11. Note the result on the right-hand side panel. As you gave a dummy phone number in this example, you get the result as invalid.



You have invoked an API using a SOAP client.

### Invoke an API using the Integrated API Console

WSO2 API Manager has an integrated Swagger UI, which is part of the Swagger project.

[Swagger](#) is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe a services in the same way that interfaces describe lower-level programming code.

The [Swagger UI](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation and more discoverability. The Swagger UI has JSON code and its UI facilitates easier code indentation, keyword highlighting and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI.

Also, see the [Swagger 2.0 specification](#).

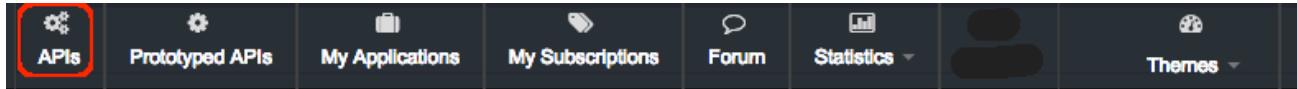
Let's see how to use the API Console in the Store to invoke an API.

! See the following topics for a description of the concepts that you need to know when invoking an API:

- Applications
- Throttling
- Access tokens
- Cross-origin resource sharing if you have the **API Store and Gateway in different ports** or you want to invoke an API with **inline endpoints**.

Create and Publish an API. The examples here use the PhoneVerification API, which is created in section

1. Log in to the API Store and click an API (e.g., PhoneVerification).
2. Subscribe to the API (e.g., PhoneVerification 1.0.0) using the **default application** and an available



Search API   Search Info

### PhoneVerification - 1.0.0



Status: PUBLISHED

Updated: 18/May/2015 13:49:17 PM IST

Bronze

Allows 1 request(s) per minute.

Subscribe

Overview

Documentation

API Console

Throttling Info

3. Go to the **My Subscriptions** page and generate a production key to the default application using which I subscribed to the PhoneVerification API.



2.

APIs   Prototyped APIs   My Applications   **My Subscriptions**   Forum   Statistics   Tools   Themes   admin

Search API  

## Subscriptions

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

### Applications With Subscriptions

DefaultApplication

Show Keys

#### Keys - Production

Consumer Key : G7jkZWkrKMQlof4XvcCWH9D561Ea

Consumer Secret : hppj\_c\_7D1MUK00PamoQUrjfj8la

Access Token : 1ca4534e9b04d6b33efcc462ba882a0

cURL   Validity Time: 3600 Seconds

Allowed Domains  
ALL

The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.

#### Keys - Sandbox

Sandbox keys are not yet generated for this application.

Click the **APIs** menu in the API Store and then click on the API that you want to invoke. When the API opens, it's API Console

Screenshot of the WSO2 API Manager interface showing the 'PhoneVerification - 1.0.0' API details and the 'API Console' tab selected.

The top navigation bar includes links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Themes, and admin.

The search bar contains the text "Search API".

The main title is "PhoneVerification - 1.0.0".

User information: admin

API Details:

- Rating:** Your rating: N/A
- Version:** 1.0.0
- Status:** PUBLISHED
- Updated:** 08/Dec/2014 16:50:11 PM IST

Subscription Options:

- Applications: Select Application...
- Tiers: Unlimited
- Allows unlimited requests
- Subscribe button

Tab navigation: Overview, Documentation, API Console (selected), Throttling Info.

Try It Out configuration:

- Try: DefaultApplication
- On: Production
- Environment: Envioromant.

Set Request Header: Authorization : Bearer b690ca26e6375b44b5bde78a44fbef2

The API Console section displays the "PhoneVerification" resource with two operations:

- GET /CheckPhoneNumber**
- POST /CheckPhoneNumber**

Swagger Resource Listing ( /api-docs ) is available.

Step 5 instructions:

5. Expand the GET method, provide the required parameters and click Try it Out. For example,

PhoneNumber	E.g., 18006785432
LicenseKey	Give 0 for testing purpose
Authorization	The API console is automatically populated by the access token that you generated in step after subscribing to the API. The token is prefixed by the string "Bearer" as per the OAuth bearer token profile. OAuth security is enforced on all published APIs. If the application key is invalid, you get a Unauthorized response in return.

<Resource, if any><backend service requirements included

**i**

If you **cannot invoke the API's HTTPS endpoint** (causes the **SSLPeerUnverifiedException**), it could be because the security certificate issued by the server is not trusted by your browser. To resolve this issue, access the HTTPS endpoint directly from your browser and accept the security certificate.

```
curl -k -H "Authorization: Bearer <access token>" -v '<API URL>'
```

**GET /CheckPhoneNumber**

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

**Response Messages**

curl -k -H "Authorization :Bearer 8e64c4201d1c311c76a9c540856d1043"  
 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=1800678543  
 &LicenseKey=0'  
 200

**Try it out!**

**POST /CheckPhoneNumber**

**BASE URL: /phoneverify/1.0.0 , API VERSION: 1.0.0**

6. Note the response for the API invocation. As we used a valid phone number in this example, the response is

## Response Body

```
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query"> <Company>Toll Free</Company>
  <Valid>true</Valid>
    <Use>Assigned to a code holder for normal use.</Use>
    <State>TF</State>
    <RC />
    <OCN />
    <OriginalNumber>18006785432</OriginalNumber>
    <CleanNumber>8006785432</CleanNumber>
    <SwitchName />
    <SwitchType />
    <Country>United States</Country>
    <CLLI />
    <PrefixType>Landline</PrefixType>
    <LATA />
    <sms>Landline</sms>
    <Email />
    <AssignDate />
    <TelecomCity />
    <TelecomCountry />
```

## Response Code

200

**curl -k -H "Authorization :Bearer e9c8e79669041b4f73ab922f82692fa" --data "PhoneNumber=18006785432&LicenseKey=0"**

**Response Headers**

```
curl -k -H "Authorization :Bearer e9c8e79669041b4f73ab922f82692fa" --data "PhoneNumber=18006785432&LicenseKey=0"
  https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber
  Pragma: no-cache
  Content-Type: text/xml; charset=utf-8
  Cache-Control: no-cache
```



**My Subscriptions** menu in the API Store, select the application used for the subscription, find the API under the **Subscribed APIs** section and click the delete icon associated with it.

The screenshot shows the 'Subscriptions' section of the API Store. A dropdown menu is open for the application 'TestApp'. The 'Subscribed APIs' option is selected and highlighted with a red box. Below it, a list of subscribed APIs is shown, with one entry for 'PhoneVerif.. - 1.0.0' also highlighted with a red box. To the right of this entry is a small red square containing a white 'X' icon, which is also highlighted with a red box, indicating it is the target for deletion.

If you unsubscribe from an API and then resubscribe with a different tier, it takes approximately 15 minutes for the tier change to be reflected. This is because the older tier remains in the cache until it is refreshed periodically by the system.

## Use the Community Features

The API Store provides several useful features to build and nurture an active community of users for your APIs. This is required to advertise APIs, learn user requirements and market trends.

Let's see what community features are available in the API Store:

- Use the search facility
- Rate and comment
- Share on social media/e-mail
- Embed an API widget
- Participate in the forum

### Use the search facility

You can search for APIs in the API Publisher or Store in the following ways:

Clause	Syntax
By the API's name	As this is the default option, simply enter the API's name and search.
By API the API provider	<b>provider:xxxx</b> . For example, provider:admin Provider is the user who created the API.
By the API version	<b>version:xxxx</b> . For example, version:1.0.0 A version is given to an API at the time it is created.
By the context	<b>context:xxxx</b> . For example, context:/phoneverify Context is the URL context of the API that is specified as /<context_name> at the time the API is created.

By the API's status	<b>status:xxxx</b> . For example, status:PUBLISHED A state is any stage of an API's lifecycle. The default lifecycle stages include created, prototyped, published, deprecated, retired and blocked.
By description	<b>description:xxxx</b> A description can be given to an API at the time it is created or later. There can be APIs without descriptions as this parameter is optional.
By the subcontext	<b>subcontext:xxxx</b> . For example, subcontext:/checkphonenumbers. A subcontext is the URL pattern of any resource of the API. API resources are created at the time the API is created or later when it is modified. For example, if you create a resource by the name checkphonenumbers, then /checkphonenumbers becomes one subcontext of the API.
By the content of the API documentation	<b>doc:xxxx</b> You can create API documentation in-line (using the API Publisher UI itself), by uploading a file or referring to an external URL. This search enables you to give a sentence or word phrase that is inside the in-line documentation and find the API that the documentation is added for.

### **Rate and comment**

Rates and comments give useful insights to potential API consumers on the quality and usefulness of an API. You can rate and comment on APIs per each version.

1. Log in to the API Store and click on a published API.
2. The API's **Overview** page opens. Note the rating and commenting options there:

**PhoneVerification - 1.0.0**

admin



**Rating:**  Your rating: 5/5  
★★★★★

---

**Version:** 1.0.0

---

**Status:** PUBLISHED

---

**Updated:** 04/Sep/2014 10:02:30 AM IST

**Overview** **Documentation** **API Console** **Throttling Info** **Forum**

**Production and Sandbox URLs:**

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

**Share:**

Social Sites Embed Email

**Comments:**

Characters left: 450

3. Add a rating and a comment. Note that the comments appear sorted by the time they were entered, alongside the author's name.

#### **Share on social media/e-mail**

1. Log in to the API Store and click on a published API.
2. On the API's **Overview** page, you get the social media options using which you can share and advertize APIs.

## PhoneVerification - 1.0.0

 admin



Rating:	Your rating: N/A 
Version:	1.0.0
Status:	PUBLISHED
Updated:	14/Aug/2014 14:19:47 PM IST

**Overview**

Documentation

API Console

Throttling Info

Forum

### Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

### Share:

[Social Sites](#) [Embed](#) [!\[\]\(46c3d349a7f4b3104fb518bc3fb4df07\_img.jpg\) Email](#)



### Embed an API widget

A widget is an embeddable version of the API in HTML that you can share on your Website or other Web pages. This is similar to how Youtube videos can be embedded in a Web page.

1. Log in to the API Store and click on a published API.
2. Note the Embed tab under the API's sharing options.

## PhoneVerification - 1.0.0

admin



Rating:	Your rating: N/A ★★★★★
Version:	1.0.0
Status:	PUBLISHED
Updated:	14/Aug/2014 14:19:47 PM IST

Overview

Documentation

API Console

Throttling Info

Forum

### Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

### Share:

Social Sites

Embed

Email

```
<iframe width="450" height="120" src="https://localhost:9443/store/apis/widget?name=PhoneVerification&version=1.0.0&provider=admin" frameborder="0" allowfullscreen></iframe>
```

### Participate in the forum

1. Log in to the API Store.
2. Click the **Forum** tab or menu to go to the forum where you can initiate conversations and share your opinions with other users.

## PhoneVerification - 1.0.0

 admin



Rating:	Your rating: N/A
	
Version:	1.0.0
Status:	PUBLISHED

Applications  
Select Application...

Tiers  
Unlimited

Allows unlimited requests

[Subscribe](#)

[Overview](#) [Documentation](#) [API Console](#) [Throttling Info](#) [Forum](#)

[Create New Topic](#)

Search Forum



Topic

Replies

[When do we get the next version of this API?](#)



0

By : admin

## Security

See the following topics:

- [Block Subscription to an API](#)
- [Pass a Custom Authorization Token to the Backend](#)

### Block Subscription to an API

An API creator **blocks subscription** to an API as a way of disabling access to it and managing its usage and monetization. A blocking can be temporary or permanent. There is an unblocking facility to allow API invocations back.

You block APIs by subscriptions. That is, a given user is blocked access to a given API subscribed to using a given application. If a user is subscribed to two APIs using the same application and you block access to only one of the APIs, s/he can still continue to invoke the other APIs that s/he subscribed to using the same application. Also, s/he can continue to access the same API subscribed to using different applications.

Blocking can be done in two levels:

- **Block production and sandbox access:** API access is blocked with both production and sandbox keys
- **Block production access only:** Allows sandbox access only. Useful when you wants to fix and test an issue in an API. Rather than blocking all access, you can block production access only, allowing the developer to fix and test.

When [API Gateway caching](#) is enabled (it is enabled by default), even after blocking a subscription, consumers might still be able to access APIs until the cache expires, which happens approximately every 15 minutes.



See the following topics for a description of the concepts that you need to know when you block

subscriptions to an API:

- Applications
- Throttling
- Access tokens

1. Log in to the API Publisher.
2. Create two APIs by the names TestAPI1 and TestAPI2 and publish them to the API Store.

The screenshot shows the WSO2 API Publisher interface. The left sidebar has 'APIs' selected and 'Browse' highlighted. The main area shows four published APIs: PhoneVer.. - 1.0.0, PhoneVer.. - 1.1.0, TestAPI1 - 1.0.0, and TestAPI2 - 1.0.0. The TestAPI1 and TestAPI2 cards are highlighted with a red box.

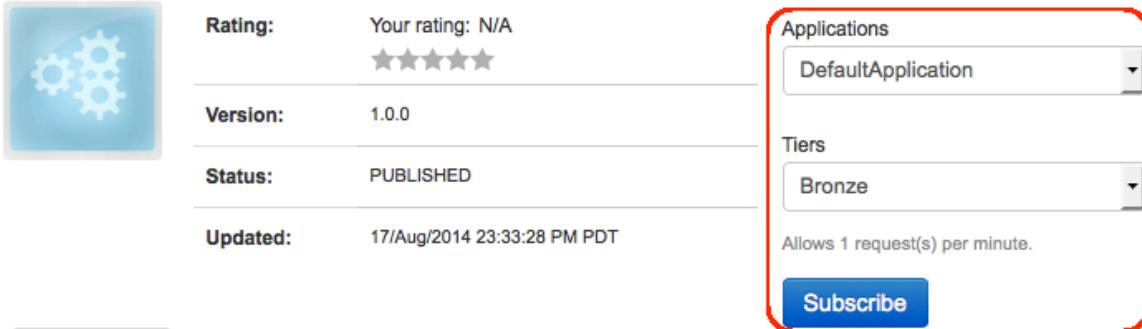
3. Log in to the API Store. Click the **APIs** menu and note that the two APIs are visible in the APIs page.

The screenshot shows the WSO2 API Store interface. The top navigation bar has 'API Store' selected and 'APIs' highlighted. The 'Recently Added' section shows three APIs: PhoneVerification-1.1.0, TestAPI1-1.0.0, and TestAPI2-1.0.0. The TestAPI1 and TestAPI2 cards are highlighted with a red box.

4. Subscribe to both APIs using the same application. You can use the default application or create your own.

## TestAPI1 - 1.0.0

 nirdesha7.yahoo.com@nirdesha



Rating: Your rating: N/A  
★★★★★

Version: 1.0.0

Status: PUBLISHED

Updated: 17/Aug/2014 23:33:28 PM PDT

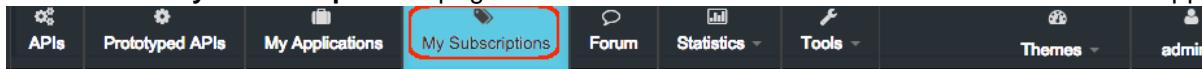
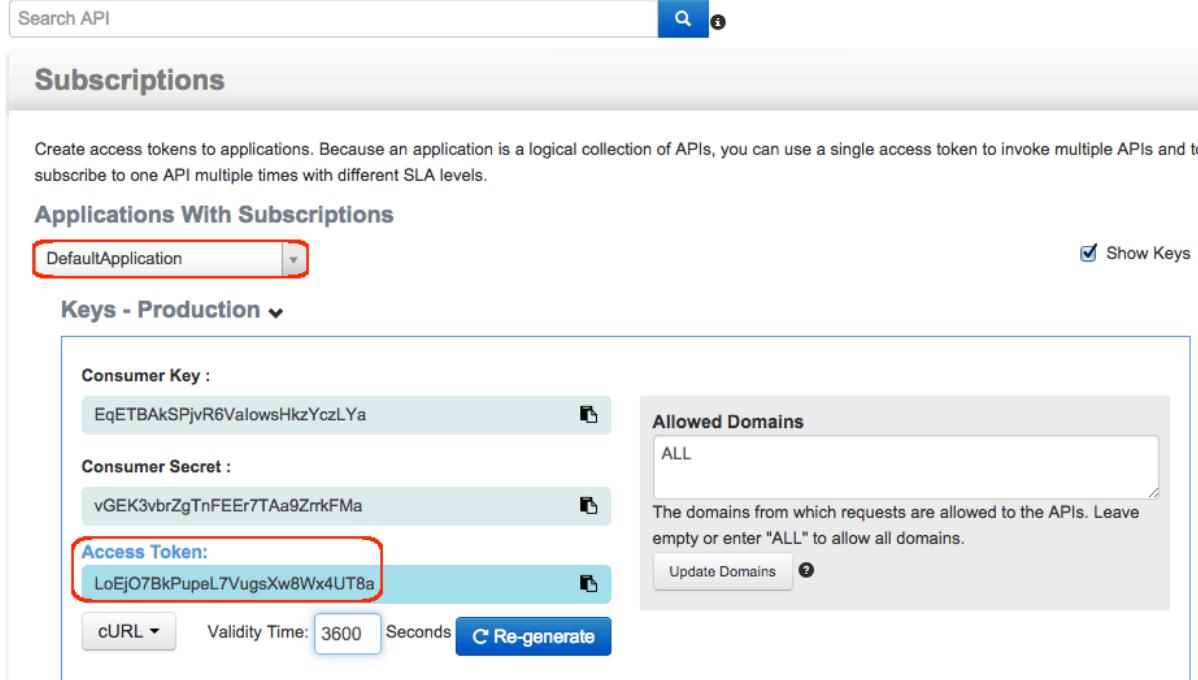
Applications: DefaultApplication

Tiers: Bronze

Allows 1 request(s) per minute.

**Subscribe**

5. Go to the **My Subscriptions** page and create an access token to the default application.

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

**Applications With Subscriptions**

DefaultApplication  Show Keys

**Keys - Production**

Consumer Key : EqETBAkSPjvR6ValowsHkzYczLYa

Consumer Secret : vGEK3vbrZgTnFEEr7TAa9ZrrkFMa

Access Token : LoEjO7BkPugeL7VugsXw8Wx4UT8a

Allowed Domains: ALL

The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.

cURL ▾ Validity Time: 3600 Seconds **C Re-generate**

6. Invoke both APIs using the access token you got in the previous step. We use cURL here. The command is,

```
curl -k -H "Authorization: Bearer <access token>" '<API URL>'
```

Be sure to replace the placeholders as follows:

- **<access token>**: Give the token generated in step 5
- **<API URL>**: Go to the API's **Overview** tab in the API Store and copy the production URL and append the payload to it.

Here's an example:

```
curl -k -H "Authorization :Bearer 633d6db88e6ee42457e60ad1b736210"
'https://localhost:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
```

```
$ curl -k -H "Authorization :Bearer 633d6db88e6ee42457e60ad1b736210"
'https://localhost:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
<Company>Toll Free</Company>
<Valid>true</Valid>
<Use>Assigned to a code holder for normal use.</Use>
<State>TF</State>
<RC />
<OCN />
<OriginalNumber>18006785432</OriginalNumber>
<CleanNumber>8006785432</CleanNumber>
<SwitchName />
<SwitchType />
<Country>United States</Country>
<CLLI />
<PrefixType>Landline</PrefixType>
<LATA />
<sms>Landline</sms>
<Email />
<AssignDate />
<TelecomCity />
<TelecomCounty />
<TelecomState>TF</TelecomState>
<TelecomZip />
<TimeZone />
<Lat />
<Long />
<Wireless>false</Wireless>
```

You have subscribed to two APIs and invoked them successfully. Let's block one subscription and see the outcome.

7. Log back to the API Publisher.
8. Click the **Subscriptions** menu to open the **Subscriptions** page. It shows all APIs/applications that each user is subscribed to.

The screenshot shows the 'Subscriptions' page in the WSO2 API Manager. On the left, there's a sidebar with 'APIs' navigation items like 'Browse', 'Add', 'All Statistics', 'My APIs' (which is selected and highlighted in blue), 'Subscriptions' (which is also highlighted in red), 'Statistics', and 'Tier Permissions'. The main content area has a header 'Subscriptions' and a table with four columns: 'User', 'Application', 'Subscribed APIs', and 'Actions'. There are two rows of data:

User	Application	Subscribed APIs	Actions
admin	DefaultApplication	PhoneVerification-1.0.0 PhoneVerification2-1.0.0 <b>TestAPI1-1.0.0</b> TestAPI2-1.0.0	<input type="radio"/> Production Only <input checked="" type="radio"/> Production & Sandbox <input type="radio"/> Unblock <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block
admin	NewApp	PhoneVerification-1.0.0 PhoneVerification2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block

9. Block subscription for TestAPI1 using the DefaultApplication. Select the production and sandbox option and click the **Block** link.

This screenshot shows the same 'Subscriptions' page after the 'Block' link for the TestAPI1-1.0.0 subscription under the DefaultApplication row has been clicked. The 'Block' link is now highlighted with a red box, indicating it has been selected. The other rows and their actions remain the same as in the previous screenshot.

User	Application	Subscribed APIs	Actions
admin	DefaultApplication	PhoneVerification-1.0.0 PhoneVerification2-1.0.0 <b>TestAPI1-1.0.0</b> TestAPI2-1.0.0	<input type="radio"/> Production Only <input checked="" type="radio"/> Production & Sandbox <input type="radio"/> Unblock <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block
admin	NewApp	PhoneVerification-1.0.0 PhoneVerification2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block

10. Note that the **Block** link immediately turns to **Unblock**, allowing you to activate the subscription back at any time.  
 11. Log back to the API Store.  
 12. Invoke the two APIs (TestAPI1 and TestAPI2) again.



You might have to **regenerate the access token** for DefaultApplication if the access token

expiration time (1 hour by default) has passed since the last time you generated it. You can refresh the access token by going to the **My Subscriptions** page in the Store.

13. Note that you can invoke TestAPI2 again but when you invoke TestAPI1, it gives a message that the requested API is temporarily blocked. Neither the API creator nor any subscriber can invoke the API until the block is removed.

```
$ curl -k -H "Authorization :Bearer 633d6db88e6ee42457e60ad1b736210"
'https://localhost:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
<ams:fault xmlns:ams="http://wso2.org/apimanager/security"><ams:code>900907</ams:code><ams:message>
The requested API is temporarily blocked.</ams:message><ams:description>Access failure for API: /test1/1.0.0, version: 1.0.0 with key: 633d6db88e6ee42457e60ad1b736210</ams:description></ams:fault>
```

14. Go to the **My Subscriptions** page in the API Store, select the application that you used to subscribe to the API and note that your subscription is blocked.

The screenshot shows the WSO2 API Store interface. At the top, there's a navigation bar with tabs: APIs, Prototyped APIs, My Applications, My Subscriptions (which is highlighted with a red box), Forum, and Statistics. Below the navigation bar is a search bar labeled 'Search API' and a magnifying glass icon. The main content area is divided into two sections: 'Recently Added' on the left and 'Subscriptions' on the right. The 'Recently Added' section displays three API entries with icons and ratings. The 'Subscriptions' section contains instructions about creating access tokens and managing SLA levels. Under 'Applications With Subscriptions', a dropdown menu shows 'DefaultApplication' (also highlighted with a red box). Below the dropdown are links for 'Keys - Production' and 'Keys - Sandbox'. In the 'Subscribed APIs' section, a box highlights a card for 'TestAPI1 - 1.0.0' which is marked as 'Blocked'.

15. Go back to the API Publisher's **Subscriptions** page and **unblock** the subscription.
16. Invoke TestAPI1 again and note that you can invoke it as usual.

You have subscribed to two APIs, blocked subscription to one and tested that you cannot invoke the blocked API.

## Pass a Custom Authorization Token to the Backend

When you send an API request to the backend, you pass a token in the Authorization header of the request. The API Gateway uses this token to authorize access, and then drops it from the outgoing message. In this tutorial, we explain how to pass a custom authorization token that is different to the authorization token generated by the API Cloud to the backend.

In this example, we have a sample JAX-RS backend that always expects 1234 as the authorization token. It is deployed in the Cloud. In your API request, you pass the token that is generated by the API Cloud in the Authorization header, and 1234 in a Custom header. The mediation extension you write extracts the value of the Custom header, and sets it as the Authorization header before sending it to the backend.

Given below is a summary of what happens:

```
Client (headers: Authorization, custom) -> Gateway (drop: Authorization, convert: custom->Authorization) -> Backend
```

Let's start by creating the mediation sequence.

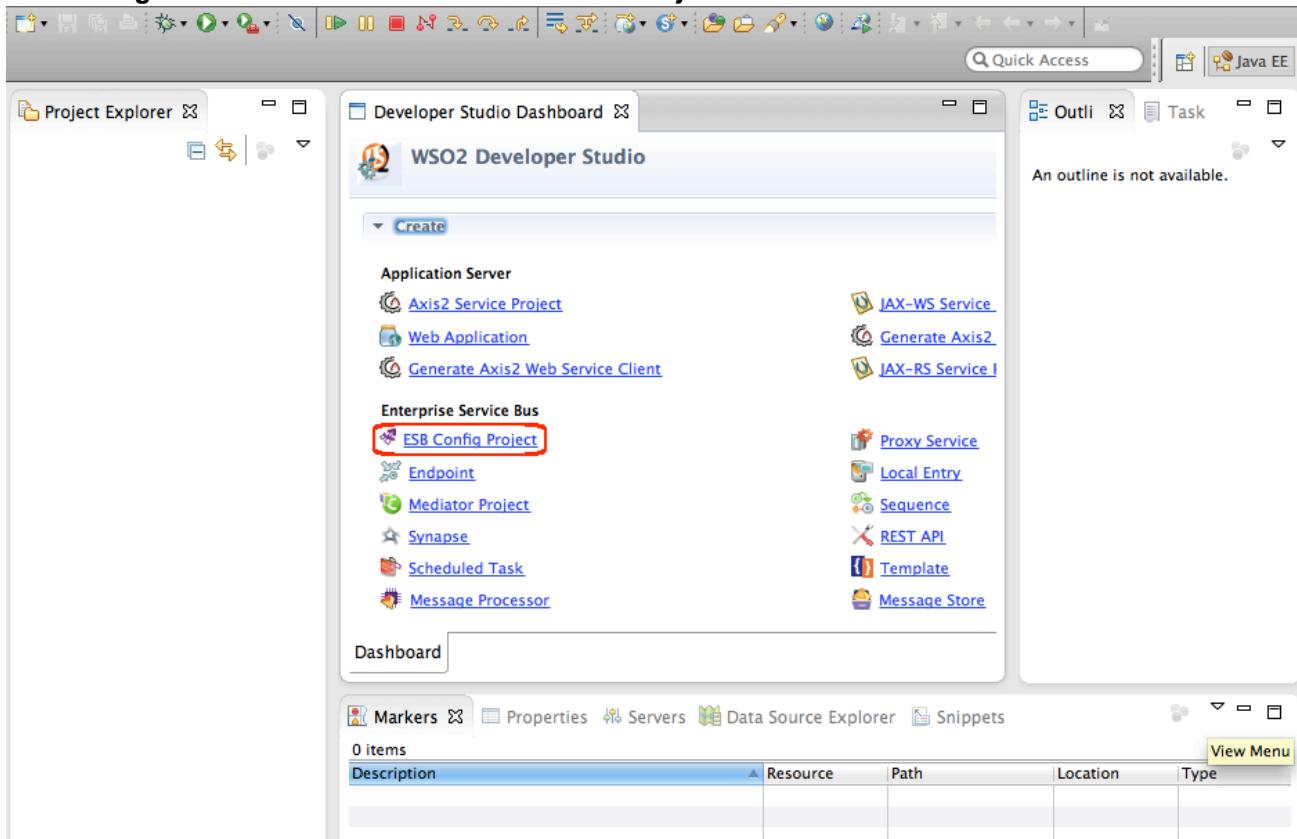
1. Download WSO2 Developer Studio (version 3.7.1 is used here) from <http://wso2.com/products/developer-studio/> and open it by double clicking the Eclipse.app file inside the downloaded folder.

 **Tip:** To start Eclipse on a Mac for the first time, open a terminal and execute the following commands:

```
cd <DevStudio_Home>/Eclipse.app/Contents/MacOS/
chmod +x eclipse
./eclipse
```

Thereafter, you can start Eclipse by double-clicking the Eclipse icon in <DevStudio\_Home>.

2. Click the **Developer Studio** menu and choose **Open Dashboard**. When the dashboard opens, click the **ESB Config** link.



3. Create a new ESB project by the name TestProject.

New ESB Config Project

Create new ESB Config Project

Project Name

Location

Use Default Location

Location

Working Sets

Add project to working sets

Working Sets

4. Click the **Sequence** link on the Developer Studio Dashboard and create a new sequence by the name Token Exchange.

**Create New Sequence**

Give a name for the sequence



Sequence Name	<input type="text" value="TokenExchange"/>
Save Sequence in:	<input type="text" value="TestProject"/> <input type="button" value="Browse..."/>
<a href="#">Create new Project...</a>	
<b>Advanced Configuration</b>	
On Error Sequence	<input type="text"/>
Available Endpoints	<input type="text"/>
<input type="checkbox"/> Make this as Dynamic Sequence	
Registry	<input type="text" value="Governance"/> <input type="button" value="Browse..."/>
Registry Path	<input type="text"/> <input type="button" value="Browse..."/>

&lt; Back

Next &gt;

Cancel

Finish

5. Your sequence now appears on the Developer Studio console. From under the **Mediators** section, drag and drop a **Property** mediator to your sequence and give the following values to the mediator.



**Tip:** The **Property Mediator** has no direct impact on a message, but rather on the message context flowing through Synapse. For more information, see [Property Mediator](#) in the WSO2 ESB documentation.

<b>Property Name</b>	Custom
<b>Value Type</b>	EXPRESSION
<b>Value Expression</b>	get-property('transport', 'Custom')

The screenshot shows the WSO2 API Manager studio interface. On the left, the Palette contains a list of mediators: LoopBack, Property (which is highlighted with a red box), Respond, and Send. The main workspace shows a sequence with a green header and a single Property mediator (labeled 'PROPERTY') inside a dashed boundary. Below the workspace is the Properties panel, which is currently displaying the configuration for a 'PropertyMediator'. The 'Appearance' tab is selected. The properties listed are:

Property	Value
Property Name	Custom
Property Action	set
Value Type	EXPRESSION
Value Expression	<code>{get-property('transport', 'Custom')}</code>
Property Scope	Synapse
Description	

- Similarly, add another **Property** mediator to your sequence and give the following values to the mediator.

<b>Property Name</b>	Authorization
<b>Value Type</b>	EXPRESSION
<b>Value Expression</b>	<code>get-property('Custom')</code>
<b>Property Scope</b>	transport

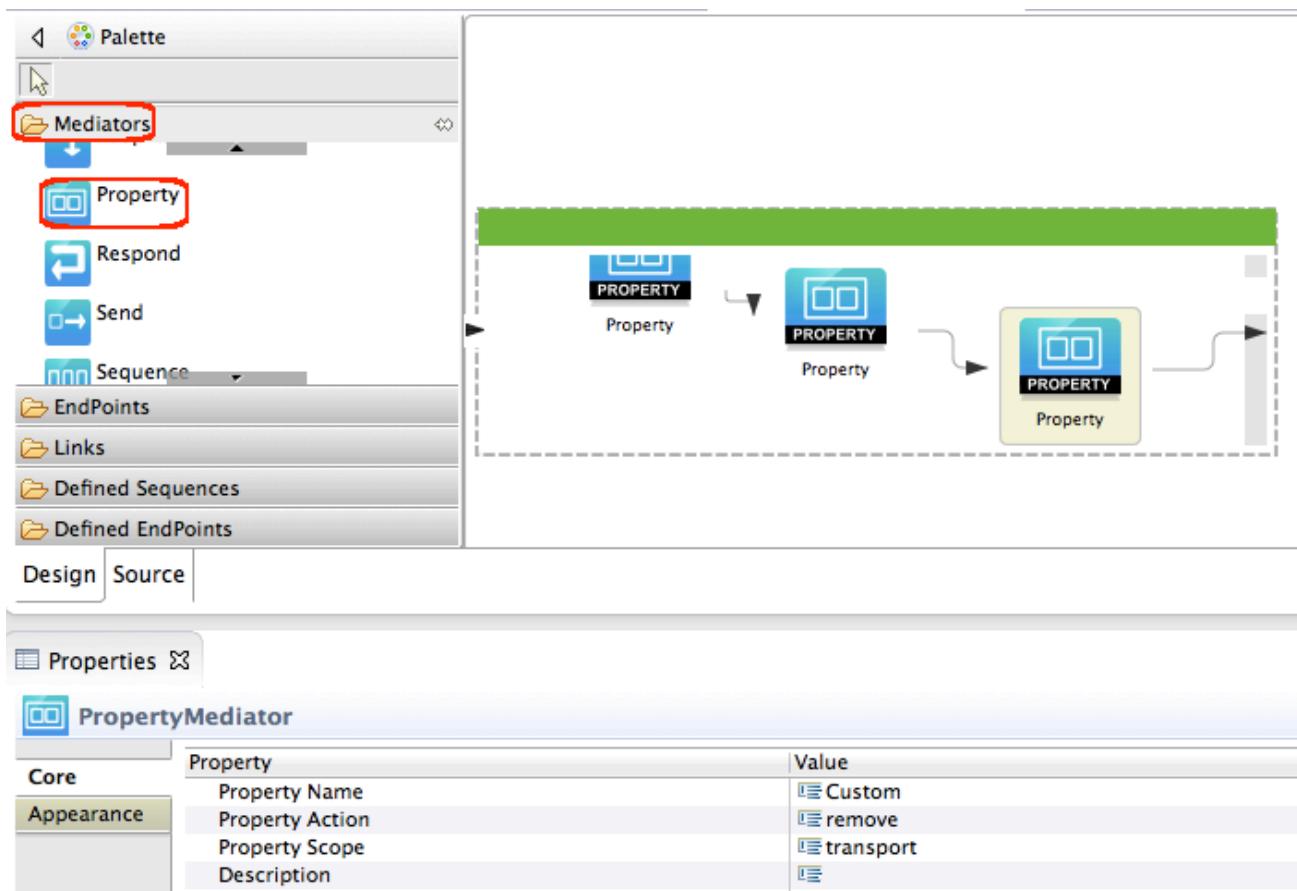
The screenshot shows the WSO2 API Manager studio interface. On the left, the 'Mediators' section of the palette is open, with the 'Property' mediator icon highlighted and circled in red. Other mediators listed include LoopBack, Respond, and Send. To the right, a sequence diagram is displayed within a green boundary. It shows two 'PROPERTY' mediator components connected by an arrow, with a third arrow pointing out from the right side of the boundary.

**Properties**

Property	Value
Property Name	Authorization
Property Action	set
Value Type	EXPRESSION
Value Expression	{get-property('Custom')}
Property Scope	transport
Description	

7. Add a third **Property** mediator to your sequence and give the following values to the mediator.

<b>Property Name</b>	Custom
<b>Property Action</b>	remove
<b>Property Scope</b>	transport

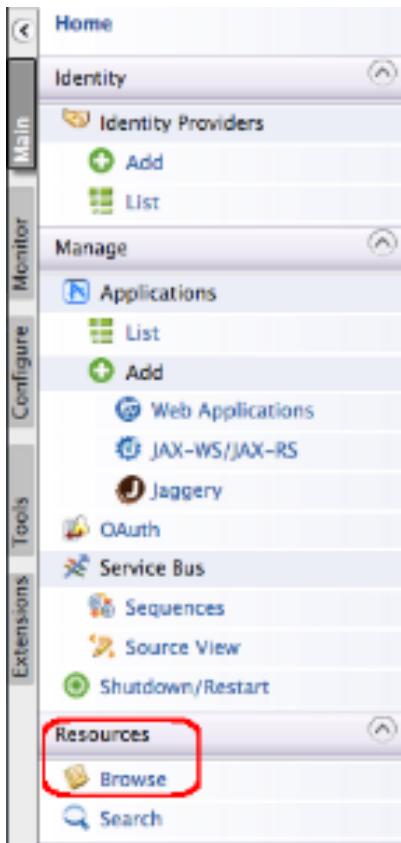


8. Save the sequence.

Let's upload this sequence to the registry.

9. Log in to the API Gateway using the URL <https://gateway.api.cloud.wso2.com/carbon/admin/login.jsp>. If you are not logged in already, use the same credentials that you used to log in to the API Cloud.

10. After logging in, click the **Browse** menu under the **Resources** menu.



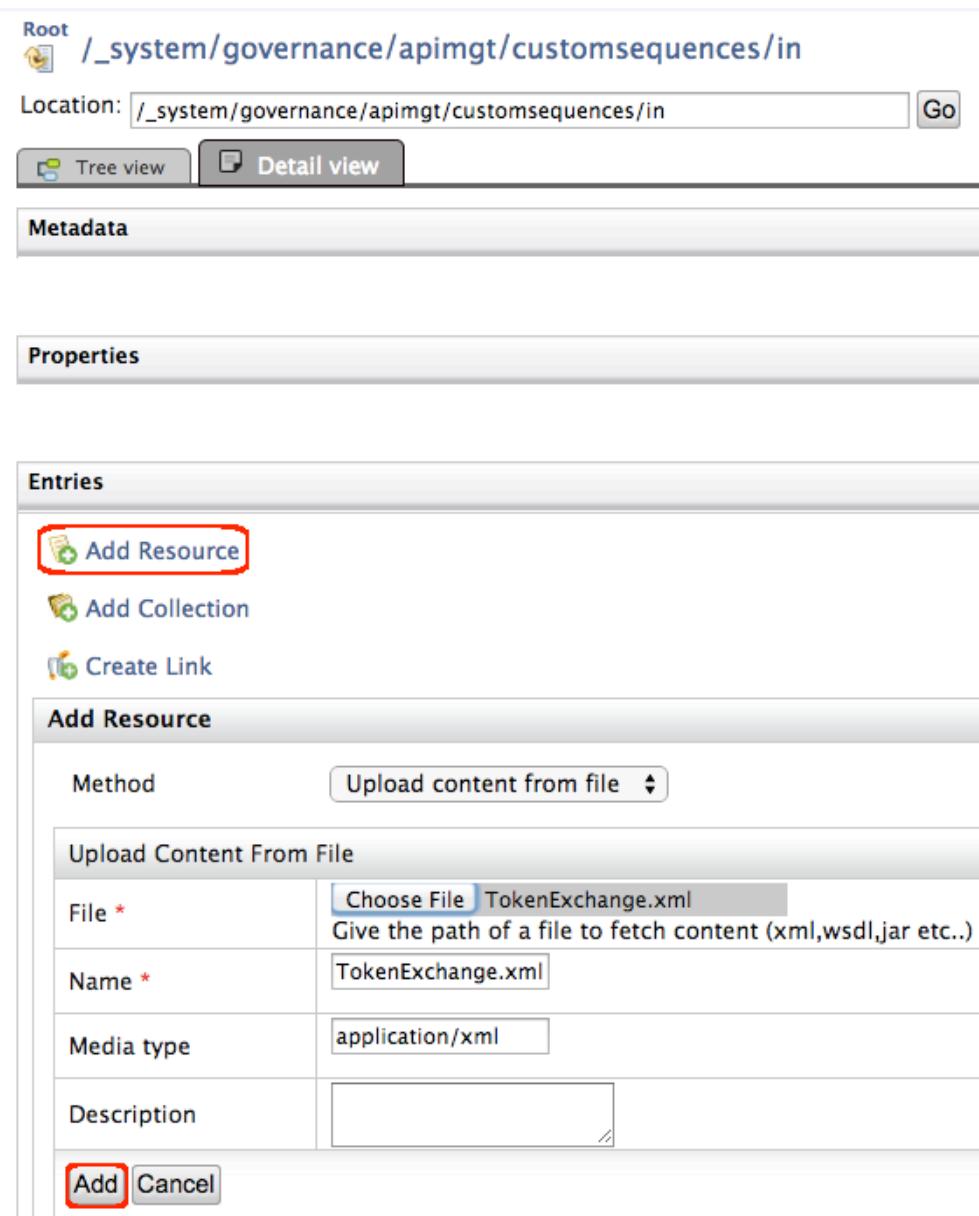
11. When the API Gateway's registry opens, navigate to the registry path `/_system/governance/apimgt/customsequences/in`. This is because you want the custom sequence to be invoked in the In direction or the request path.

The screenshot shows the 'Browse' page of the WSO2 API Manager Registry. The navigation tree on the right side shows the following structure:

- Root /
- \_system
- config
- governance
- apimgt
- applicationdata
- customsequences
  - + fault
  - + in (highlighted with a red box)
  - + out
- externalstores
- repository

12. Click **Add Resource** and upload the XML file of the sequence that you created earlier.

 **Tip:** In this example, the configuration of the TokenExchange sequence is saved in the <DVS\_HOME>/TestProject/src/main/synapse-config/sequences/TokenExchange.xml file.



The screenshot shows the WSO2 API Manager governance interface. The URL in the address bar is `/_system/governance/apimgt/customsequences/in`. Below the address bar are two buttons: "Tree view" and "Detail view". The "Detail view" button is highlighted. There are two tabs: "Metadata" and "Properties", with "Properties" currently selected. Under the "Entries" section, there is a red box around the "Add Resource" button. Below it are "Add Collection" and "Create Link" buttons. A modal window titled "Add Resource" is open. It has a "Method" dropdown set to "Upload content from file". The "Upload Content From File" section contains fields for "File \*", "Name \*", "Media type", and "Description". The "File \*" field has a "Choose File" button labeled "TokenExchange.xml" and a placeholder "Give the path of a file to fetch content (xml,wsdl,jar etc..)". The "Name \*" field is filled with "TokenExchange.xml". The "Media type" field is set to "application/xml". The "Description" field is empty. At the bottom of the modal are "Add" and "Cancel" buttons, with "Add" also having a red box around it.

- After uploading the sequence to the registry, click the sequence to open it, click the **Edit as text** link and confirm that the configuration of the sequence is as follows:

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="TokenExchange">
    <property name="Custom" expression="get-property('transport', 'Custom')"
scope="default" type="STRING"/>
    <property name="Authorization" expression="get-property('Custom')"
scope="transport" type="STRING" description="" />
    <property name="Custom" scope="transport" action="remove" />
</sequence>
```

Let's create a new API and engage the sequence you created to it.

- Log in to the API Publisher, click the **Add** link, give the information in the table below.

Field	Sample value
Name	TestAPI1
Context	/test1
Version	1.0.0
Visibility	Public

1 Design

2 Implement

3 Manage

## Design API

### General Details

[Edit Swagger Definition](#)
[Import Swagger Definition](#)
Name:<sup>\*</sup>

TestAPI1

?

Context:<sup>\*</sup>

/test1

?

Version:<sup>\*</sup>

1.0.0

E.g.: v1.0.0 ,v1.0 , 1.0.0, 1.0

Visibility:

Public

?

Thumbnail Image:

No file chosen

- Size (max): 1MB
- Dimensions (max): 100 x 100 pixels

Description:

Tags:

?

15. Leave the **Resources** section blank, and click **Implement**. It asks whether you like to add a wildcard resource (\*). Click **Yes** and then click **Implement** again to move to the **Implement** tab.

**Resources**

URL Pattern  Url Pattern Ex: path/to/resource

GET  POST  PUT  DELETE  OPTIONS

Resource Name

[+ Add New Resource](#)

**/default**

Method	Implementation	Action
GET	/* + Summary	
POST	/* + Summary	
PUT	/* + Summary	
DELETE	/* + Summary	
OPTIONS	/* + Summary	

[Save](#) [Implement](#) [Cancel](#)

16. The **Implement** tab opens. Give the information in the table below and click **Manage**.

Field	Sample value
Implementation method	Backend
Endpoint type	HTTP endpoint
Production endpoint	<a href="https://appserver.cloud.wso2.com/t/clouddemo/webapps/authheadersamp">https://appserver.cloud.wso2.com/t/clouddemo/webapps/authheadersamp</a>

The screenshot shows the 'Implement' tab of the WSO2 API Manager interface for the API 'TestAPI1 : /t/companyn1/test1/1.0.0'. At the top, there is a navigation bar with three steps: 'Design', 'Implement' (which is highlighted with a red border), and 'Manage'. Below the navigation bar, the page title is 'TestAPI1 : /t/companyn1/test1/1.0.0'. Underneath the title, there is a section for 'Implementation Method' with two options: 'Backend Endpoint' (selected) and 'Specify Inline'. The main content area is titled 'Endpoints'. It includes fields for 'Endpoint Type:' (set to 'HTTP Endpoint'), 'Production Endpoint' (containing the value '.0.0/services/customer-service/1' with 'Advanced Options' and 'Test' buttons), and 'Sandbox Endpoint' (with a placeholder and 'Advanced Options' and 'Test' buttons). Below these fields is a link 'Show More Options'. At the bottom of the page are four buttons: 'Save' (blue), 'Deploy Prototype' (orange), 'Manage' (highlighted with a red border), and 'Cancel'.

17. In the Manage tab, select the Gold tier. Also click the **Sequences** check box and select the TokenExchange sequence that you created earlier for the In flow.

1 Design

2 Implement

3 Manage

## TestAPI1 : /t/companyn1/test1/1.0.0

### Configurations

Make this default version



No default version defined for the current API

Tier Availability:\*

Gold



Transports:\*

 HTTP HTTPS

Sequences:

Check to select a custom sequence to be executed in the message flow

In Flow	Out Flow	Fault Flow
Toke	None	N

Response Caching:

Disabled



Subscriptions:

Available to current tenant



### Business Information >

18. Click **Save and Publish** to publish the API to the API Store.  
Let's subscribe to the API and invoke it.
19. Go to the API Store by clicking the **Go to API Store** link at the top right-hand corner of the screen. If you are not logged in already, use the same credentials that you used to log in to the API Cloud.
20. Click TestAPI1, and subscribe to the API using an available application and the Gold tier. If there are no applications available by default, create one.

The screenshot shows the WSO2 API Manager interface. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, and Themes. Below the navigation bar, there is a search bar labeled "Search API" with a magnifying glass icon. On the left, there is a sidebar titled "More APIs from 'admin'" showing a thumbnail for "PhoneVerification-1.0.0" with a 5-star rating. The main content area is titled "TestAPI1 - 1.0.0". It displays the following details:

- User: admin
- Rating: Your rating: N/A
- Version: 1.0.0
- Status: PUBLISHED
- Updated: 27/Jul/2015 11:30:16 AM IST

On the right side, there are two dropdown menus: "Applications" (set to "DefaultApplication") and "Tiers" (set to "Gold"). A note below the tiers says "Allows 20 request(s) per minute." A red box highlights the "Tiers" dropdown and the "Subscribe" button below it. At the bottom, there are tabs for Overview, Documentation, API Console, Throttling Info, and Forum.

21. When prompted, choose to go to the **My Subscriptions** page, and generate an access token to invoke the

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple times with different SLA levels.

### Applications With Subscriptions

DefaultApplication

#### Keys - Production ▾

**Consumer Key :** gTL5LVKtCzHJxsPnhR1PRTEf0aYa

**Consumer Secret :** eEQbmykM7BLC5UZv5Eek9j2H38ka

**Access Token:** acca1f3034f64126f87132c699294c

curl ▾      Validity Time: 3600 Seconds      C Re-generate

**Allowed Domains**

ALL

The domains from which requests are made enter "ALL" to allow all domains.

Update Domains

22. Install any REST client in your machine. We use **cURL** here.
23. Go to the command-line, and invoke the API using the following cURL command. In this command, you pass the token that the backend expects, i.e., 1234, in the **Custom** header with the authorization token that the system generates in the **Authorization** header.

```
curl -H "Authorization: Bearer <access token>" -H "Custom: Bearer 1234" <API URL>
```

Note the following:

- **<access token>** is the token that you got in step 20.
- **<API URL>** appears on the API's **Overview** page in the API Store. Copy the HTTP endpoint. If you select the HTTPS endpoint, be sure to run the cURL command with the -k option.

Here's an example:

```
curl -H "Authorization: Bearer DNp94L7mAT0e30j0QuUx0rmSIt0a" -H "Custom: Bearer 1234" 'http://gateway.api.cloud.wso2.com:8280/t/companyn1/test1/1.0.0'
```

24. Note the response that you get in the command line. According to the sample backend used in this tutorial, you get the response as "Request Received."

```
$ curl -H "Authorization: Bearer DNp94L7mAT0e30j0QuUx0rmSIt0a" -H "Custom: Bearer 1234" 'http://gateway.api.cloud.wso2.com:8280/t/companyn1/test1/1.0.0'
<Response><code>200</code><message>Request Received</message><description>Request Received successfully</description></Response>
```

In this tutorial, you passed a custom token that the backend expects along with the system-generated Authorization token, and invoked an API successfully by swapping the system's token with your custom token.

## Configuring the API Manager

This section covers the following

- Customizing the API Store
- Configuring Multiple Tenants
- Adding Internationalization and Localization
- Configuring Single Sign-on with SAML2
- Changing the Default Transport
- Configuring Caching
- Working with Databases
- Managing Users and Roles
- Configuring User Stores
- Directing the Root Context to the API Store
- Adding Links to Navigate Between the Store and Publisher
- Maintaining Separate Production and Sandbox Gateways
- Configuring Transports
- Transforming API Message Payload
- Sharing Applications and Subscriptions

### Customizing the API Store

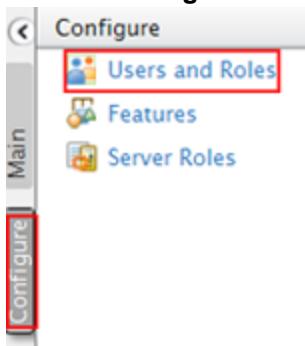
You can customize the API Store in the following ways:

- Enabling or disable self signup
- Changing the theme
- Changing language settings
- Setting single login for all apps
- Categorising and grouping APIs

#### *Enabling or disable self signup*

In a multi-tenanted API Manager setup, self signup to the API Store is disabled by default to all tenants except the super tenant. A tenant admin can enable it as follows:

1. Log in to the management console (<https://<HostName>:9443/carbon>) as admin (or tenant admin).
2. Click the **Configure -> Users and Roles** menu.



3. In the **User Management** page that opens, click **Roles**.



4. Add a role by the name subscriber (or any other name you prefer) and the following permissions:

- Login
- Manage > API > Subscribe

## Add Role

### Step 1 : Enter role details

Enter role details

Domain	PRIMARY
Role Name*	subscriber
<input type="button" value="Next &gt;"/> <input type="button" value="Finish"/> <input type="button" value="Cancel"/>	

5. Go to the **Resources -> Browse** menu.

6. Load the `/_system/governance/apimgt/applicationdata/sign-up-config` resource in the browser UI.

The screenshot shows the WSO2 API Manager interface. On the left, there is a vertical sidebar with tabs: Main, Monitor, Configure, Tools, and Extensions. Under the 'Main' tab, the 'Resources' section is selected, and it contains 'Browse' and 'Search' buttons, both of which are highlighted with red boxes. The main area is titled 'Browse' and shows a tree view of resources under 'Root /'. The path '/\_system/governance/apimgt/applicationdata/sign-up-config' is expanded, and the file 'sign-up-config.xml' is highlighted with a red box.

7. Do the following changes in the signup configuration and save.

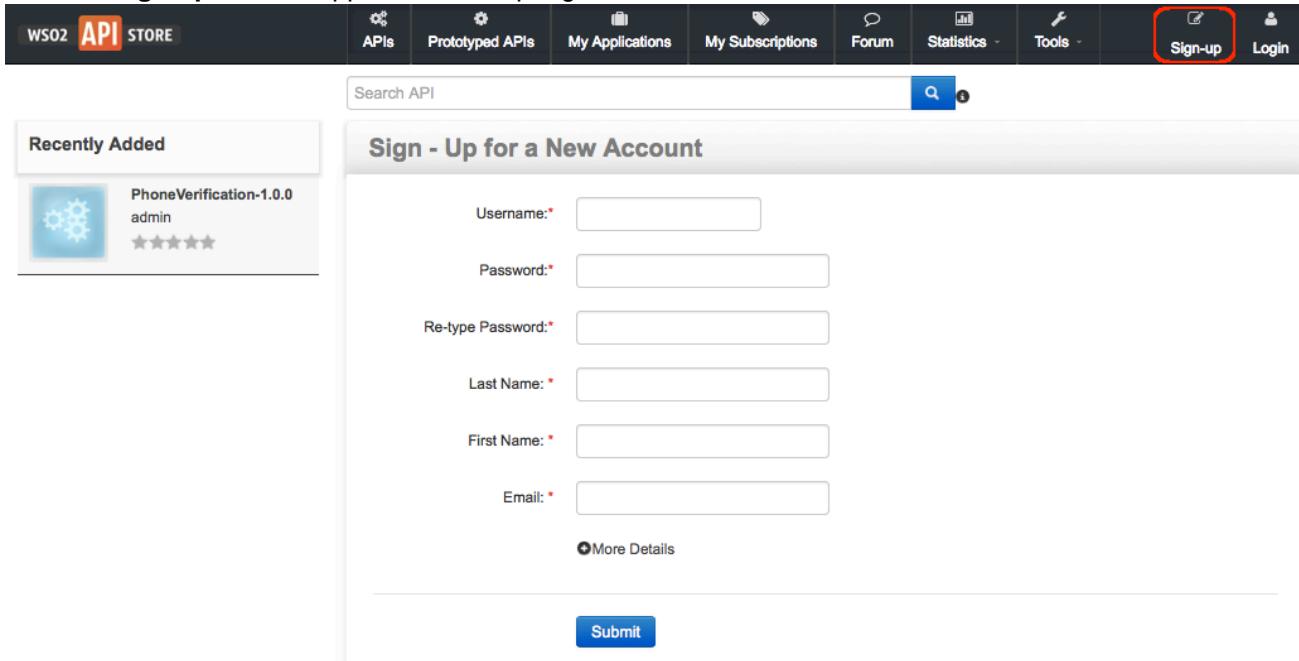
- Set `<EnableSignup>` to `true`.
- Set `<RoleName>` to `subscriber` and `<IsExternalRole>` to `true`. Note that you must have the `subscriber` role created at this point.
- Set `<AdminUserName>` and `<AdminPassword>` to the credentials of the super admin, or if you are in a **multitenant setup** and you are not the super admin, to the tenant admin's credentials. **Note** that the super admin's credentials are `admin/admin` by default. If you changed the default super admin's credentials, using `admin/admin` will cause errors.

```

<SelfSignUp>
    <EnableSignup>true</EnableSignup>
    <!-- user storage to store users -->
    <SignUpDomain>PRIMARY</SignUpDomain>
    <!-- Tenant admin information. (for clustered setup credentials for
AuthManager) -->
    <AdminUserName>xxxx</AdminUserName>
    <AdminPassword>xxxx</AdminPassword>
    <!-- List of roles for the tenant user -->
    <SignUpRoles>
        <SignUpRole>
            <RoleName>subscriber</RoleName>
            <IsExternalRole>true</IsExternalRole>
        </SignUpRole>
    </SignUpRoles>
</SelfSignUp>

```

8. Restart the server and open the API Store (<https://<HostName>:9443/store>).
9. Note the **Sign-up** link that appears in the top, right-hand corner of the window.



The screenshot shows the WSO2 API Store homepage. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Sign-up (which is highlighted with a red box), and Login. Below the navigation bar is a search bar labeled "Search API". On the left, there is a sidebar titled "Recently Added" featuring a thumbnail of a gear icon, the name "PhoneVerification-1.0.0", the owner "admin", and a 5-star rating. The main content area is titled "Sign - Up for a New Account". It contains fields for Username, Password, Re-type Password, Last Name, First Name, and Email, each with a red asterisk indicating it is required. There is also a "More Details" link and a "Submit" button at the bottom.

10. To disable the self signup capability, navigate to `/_system/governance/apimgt/applicationdata/signup-config.xml` in the registry again and set the `<SelfSignUp><EnableSignup>` element to false.



**Tip:** To engage your own signup process, see [Adding a User Signup Workflow](#).

#### Changing the theme

See [Adding a new API Store Theme](#).

#### Changing language settings

To change the language of the API Store, see [Adding Internationalization and Localization](#).

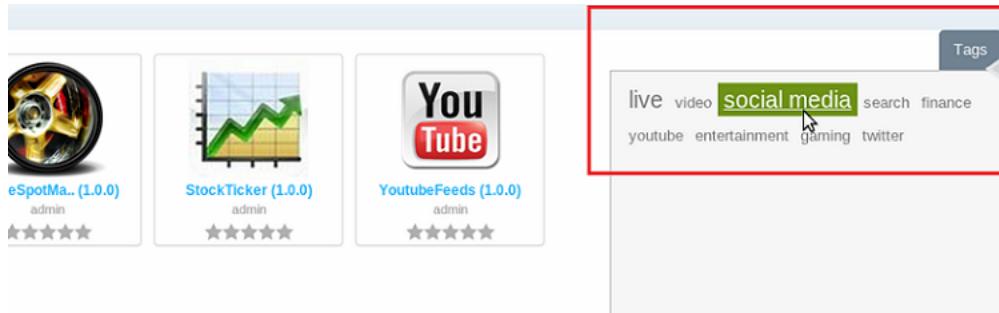
#### Setting single login for all apps

Single sign-on (SSO) allows users who are logged in to one application to automatically log in to multiple other

applications using the same credentials. They do not have to repeatedly authenticate themselves. To configure, see [Configuring Single Sign-on with SAML2](#).

### Categorising and grouping APIs

API providers add tags to APIs when designing them using the API Publisher. Tags allow API providers to categorise APIs that have similar attributes. Once a tagged API gets published to the API Store, its tags appear as clickable links to the API consumers, who can use them to quickly jump to a category of interest.



If you want to see the APIs grouped according to different topics in the API Store, do the following:

1. Go to <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf directory, open the site.json file and set the tagWiseMode attribute as true.
2. Go to the API Publisher and add tags to APIs with the suffix "-group" to APIs (e.g., Workflow APIs-group, Integration APIs-group, Quote APIs-group.)
3. Restart the server.
4. Log in to the API Store and note the APIs listed under their groups. You can click on a group to check what the APIs are inside it.

Integration APIs	Invoice APIs	Quote APIs	Workflow APIs
Lore ipsum dolor sit amet, consectetur adipiscing elit.	Lore ipsum dolor sit amet, consectetur adipiscing elit.	Lore ipsum dolor sit amet, consectetur adipiscing elit.	Lore ipsum dolor sit amet, consectetur adipiscing elit.

### Customizing the API group

If you want to change the descriptions and the thumbnail images that come by default, do the following:

1. Log in to the Management Console and click the **Resources -> Browse** menu to open the registry.



2. Create a collection named `tags` under the registry location `/_system/governance/apimgt/applicationdata`.

A screenshot of the WSO2 API Manager Registry interface. The URL in the address bar is `/_system/governance/apimgt/applicationdata`. The main area shows tabs for 'Metadata' and 'Properties'. Below these is a 'Entries' section with buttons for 'Add Resource', 'Add Collection' (highlighted with a red box), and 'Create Link'. A modal dialog titled 'Add Collection' is open. It has fields for 'Name \*' (set to 'tags'), 'Media Type' (set to 'not specified'), and 'Description' (empty). At the bottom of the dialog are 'Add' and 'Cancel' buttons, with 'Add' highlighted with a red box. To the right of the dialog, there is a large empty space.

3. Give read permission to the `system/wso2.anonymous.role` role.

The screenshot shows the 'Entries' and 'Permissions' sections of the WSO2 API Manager Management Console. In the 'Permissions' section, a red box highlights the 'Add Permission' button and the dropdown menu where 'system/wso2.anonymous.role' is selected. The 'Action' dropdown is set to 'Read' and the 'Allow' radio button is selected.

4. Add each tag as collections under the tags collection (e.g., Workflow APIs-group, Integration APIs-group, Quote APIs-group.)
5. Navigate to each tag collection and upload the following:
  - **description.txt** with the description of the tag
  - **thumbnail.png** for the thumbnail image
6. Back in the API Store, note the changes you did in the registry.

## Configuring Multiple Tenants

The goal of multitenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single sever/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multitenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

You can register tenant domains using the Management Console of WSO2 products.

This section covers the following topics:

- [Multi Tenant Architecture](#)
- [Managing Tenants](#)
- [Tenant-Aware Load Balancing using WSO2 ELB](#)

### Multi Tenant Architecture

The multi tenant architecture of WSO2 products allows you to deploy Web applications, Web services, ESB mediators, mashups etc. in an environment that supports the following:

- **Tenant isolation:** Each tenant has its own domain, which the other tenants cannot access.
- **Data isolation:** Each tenant can manage its data securely in an isolated manner.
- **Execution isolation:** Each tenant can carry out business processes and workflows independent of the other tenants. No action of a tenant is triggered or inhibited by another tenant.
- **Performance Isolation:** No tenant has an impact on the performance of another tenant.

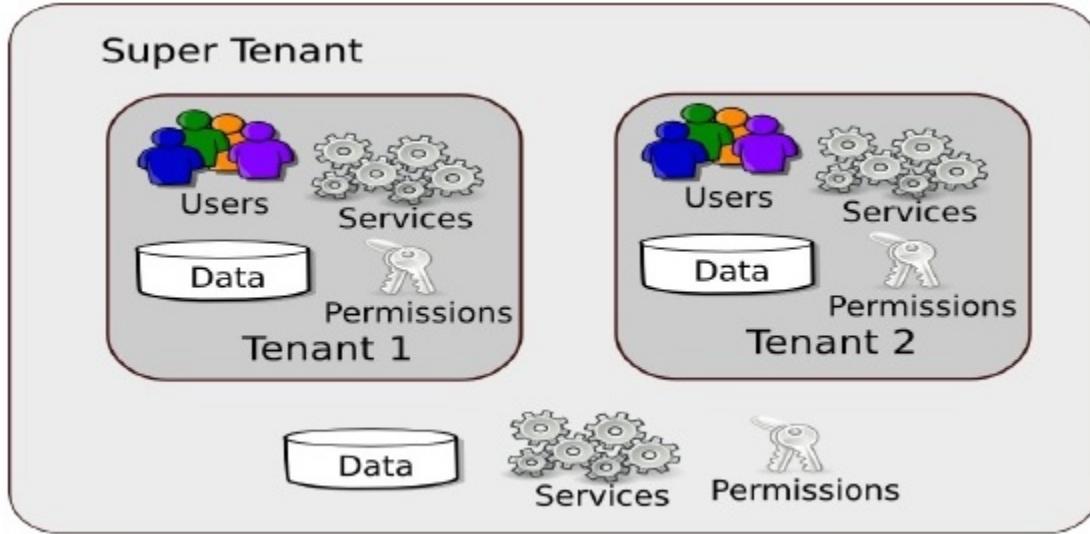
### Architecture

A tenant is an isolated domain. The users within this domain can manage their own data and perform their own transactions without being affected by actions carried out in other domains.

These domains are allocated server space from the complete server space of a WSO2 product instance which is referred to as the *super tenant*.

The super tenant as well as each individual tenant has its own configuration and context module.

Each tenant has its own security domain. A domain has a set of users, and permissions for those users to access resources. Thus, a tenant is restricted by the users and permissions of the domain assigned to it. The artifact repositories of the tenants are separated from each other.



An individual tenant can carry out the following activities within the boundaries of its own configuration and context module:

- Deploying artifacts
- Applying security
- User management
- Data management
- Request throttling
- Response caching

WSO2 Carbon provides a number of Admin services which have special privileges to manage the server. These admin services are deployed in the super tenant. Other tenants can make use of these admin services to manage their deployment. The admin services operate in a tenant aware fashion. Thus, privileges and restrictions that apply to any client using an admin service are taken into account.

### **Resource sharing**

WSO2 Carbon supports the following methods for sharing resources among tenants:

- **Private Jet mode:** This method allows the load of a tenant ID to be deployed in a single tenant mode. A single tenant is allocated an entire service cluster. The purpose of this approach is to allow special privileges (such as priority processing and improved performance) to a tenant.
- **Separation at hardware level:** This method allows different tenants to share a common set of resources, but each tenant has to run its own operating system. This approach helps to achieve a high level of isolation, but it also incurs a high overhead cost.
- **Separation at JVM level:** This method allows tenants to share the same operating system. This is done by enabling each tenant to run a separate JVM instance in the operating system.
- **Native multitenancy:** This method involves allowing all the tenants to share a single JVM instance. This method minimises the overhead cost.

### **Lazy loading**

Lazy loading is a design pattern used specifically in cloud deployments to prolong the initialization of an object or artifact until it is requested by a tenant or an internal process.

Tenants

Lazy loading of tenants is a feature that is built into all WSO2 products. This feature ensures that all the tenants are not loaded at the time the server starts in an environment with multiple tenants. Instead, they are loaded only when a request is made to a particular tenant. If a tenant is not utilized for a certain period of time (30 minutes by default), it will be unloaded from the memory.

You can change the default time period allowed for tenant inactiveness by adding `-Dtenant.idle.time=<time_in_minutes>` java property to the startup scrip of the product (`./wso2server.sh` file for Linux and `wso2server.bat` for Windows) as shown below.

```
JAVA_OPTS \
-Dtenant.idle.time=30 \
```

## Artifacts

Lazy loading of artifacts is a feature that is used by some WSO2 products, which can be enabled in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. The deployer that handles lazy loading of artifacts is called the Ghost Deployer. A flag to enable or disable the Ghost Deployer is shown below. This is set to `false` by default because the Ghost Deployer works only with the HTTP/S transports. Therefore, if other transports are used, the Ghost Deployer does not have to be enabled.

```
<GhostDeployment>
<Enabled>false</Enabled>
<PartialUpdate>false</PartialUpdate>
</GhostDeployment>
```

When a stand-alone WSO2 product instance is started with lazy loading enabled, its services, applications and other artifacts are not deployed immediately. They are first loaded in the Ghost form and the actual artifact is deployed only when a request for the artifact is made. In addition, if an artifact has not been utilized for a certain period of time, it will be unloaded from the memory.

When lazy loading of artifacts is enabled for PaaS deployments, lazy loading applies both for tenants as well as a tenant artifacts. As a result, lazy loading is applicable on both levels for a tenant in a cloud environment. Therefore, the associated performance improvements and resource utilization efficiencies are optimal.

## Restrictions

The following restrictions are imposed to ensure that each individual tenant has the required level of isolation and maintains fine grained security control over its own services without affecting the other tenants.

- Only the super tenant can modify its own configuration. In addition, it can add, view and delete tenants.
- When a tenant logs into the system, it can only access artifacts deployed under its own configuration. One tenant cannot manipulate the code of another tenant.
- The super admin or tenant admin can add user stores to their own domain. Dynamic configurations are possible only for secondary user stores and the primary user store is not configurable at run time. This is because primary user stores are available for all tenants and allowing changes to the configuration at run time can lead to instability of the system. Therefore, the primary user store is treated as a static property in the implementation and it should be configured prior to run time.
- A tenant's code cannot invoke sensitive server side functionality. This is achieved via Java security.
- Tenants share the transports provided by the system. They are not allowed to create their own transports.

## Request dispatching

This section describes how the multi tenancy architecture described above works in a request dispatching scenario.

When a Carbon server receives a request, the message is first received by the handlers and dispatchers defined for the server configuration (i.e. super tenant). The server configuration may include handlers that implement cross tenant policies and Service Level Agreement (SLA) management. For example, a priority based dispatcher can be applied at this stage to offer differentiated qualities of service to different clients. Once the relevant handlers and dispatchers are applied, the request is sent to the tenant to which it is addressed. Then the message dispatchers and handlers specific to that tenant will be applied.

The following example further illustrates how message dispatching is carried out in a multi tenant server.

For example, two tenants named foo.com and bar.com may deploy a service named MyService. When this service is hosted on the two tenants, they would have the following URLs.

`http://example.com/t/foo.com/services/MyService`  
`http://example.com/t/bar.com/services/MyService`

The name of the tenant in the URL allows the tenant to be identified when the Carbon server receives a message which is addressed to a specific client. Alternatively, you may configure a CNAME record in DNS (Domain Name System) as an alias for this information.

If a request is addressed to the MyService service hosted by foo.com, the message handlers and dispatchers of the super tenant will be applied and the tenant foo.com will be identified by the tenant name in the URL. Then the request will be sent to foo.com where it will be processed.

## Scaling

The multi tenancy architecture described above mainly refers to a scenario where a single instance of a Carbon server acts as a single multi tenant node. In a situation where a very high load of requests are handled, you may need multiple multi tenant nodes. In order to operate with multiple multi tenant nodes, you need load balancing. The load balancer you use also needs to be tenant-aware.

## Managing Tenants

You can add a new tenant in the management console and then view it by following the procedure below. In order to add a new tenant, you should be logged in as a super user.

1. Click **Add New Tenant** in the **Configure** tab of your product's management console.



2. Enter the tenant information in **Register A New Organization** screen as follows, and click **Save**.

Parameter Name	Description
<b>Domain</b>	The domain name for the organization, which should be unique (e.g., abc.com). This is used as a unique identifier for your domain. You can use it to log into the admin console to be redirected to your specific tenant. The domain is also used in URLs to distinguish one tenant from another.
<b>Select Usage Plan for Tenant</b>	The usage plan defines limitations (such as number of users, bandwidth etc.) for the tenant.
<b>First Name/Last Name</b>	The name of the tenant admin.
<b>Admin Username</b>	The login username of the tenant admin. The username always ends with the domain name (e.g., admin@abc.com)
<b>Admin Password</b>	The password used to log in using the admin username specified.
<b>Admin Password (Repeat)</b>	Repeat the password to confirm.

Email	The email address of the admin.
-------	---------------------------------

3. After saving, the newly added tenant appears in the **Tenants List** page as shown below. Click **View Tenants** in the **Configure** tab of the management console to see information of all the tenants that currently exist in the system. If you want to view only tenants of a specific domain, enter the domain name in the **Enter the Tenant Domain** parameter and click **Find**.

Tenants List		Domain	parameter	and	click	Find.
<input type="text" value="Enter the Tenant Domain"/>						<input type="button" value="Find"/>
<hr/>						
Domain	Email	Created Date	Active	Edit		
wso2.com	frankie.avalon@gmail.com	2014/11/17 12:03:06	<input checked="" type="checkbox"/>	<input type="button" value="Edit"/>		
abc.com	dean.martin@gmail.com	2014/11/17 13:43:46	<input checked="" type="checkbox"/>	<input type="button" value="Edit"/>		

When you create multiple tenants in an API Manager deployment, the API Stores of each tenant are displayed in a multi-tenanted view for all users to browse and permitted users to subscribe to as shown below:

1. Access the API Store URL (by default, `https://localhost:9443/store`) using a Web browser. You see the storefronts of all the registered tenant domains listed there. For example,

The screenshot shows the WSO2 API STORE interface. At the top left, it says "WSO2 API STORE". Below that, a heading reads "API Stores available on this server". There are three separate store fronts displayed in boxes:

- domain1.com**: Represented by a shopping bag icon with puzzle pieces inside. Below the icon is the text "Visit Store".
- domain2.com**: Represented by a shopping bag icon with puzzle pieces inside. Below the icon is the text "Visit Store".
- carbon.super**: Represented by a shopping bag icon with puzzle pieces inside. Below the icon is the text "Visit Store".

This is called the public store. Each icon here is linked to the API Store of a registered tenant, including the super tenant, which is `carbon.super`. That is, the super tenant is also considered a tenant.

2. Click the **Visit Store** link associated with a given store to open it.
3. Anonymous users can browse all stores and all public APIs that are published to them. However, in order to subscribe to an API, the user must log in.

For example, if you are a user in the `domain1.com` tenant domain,

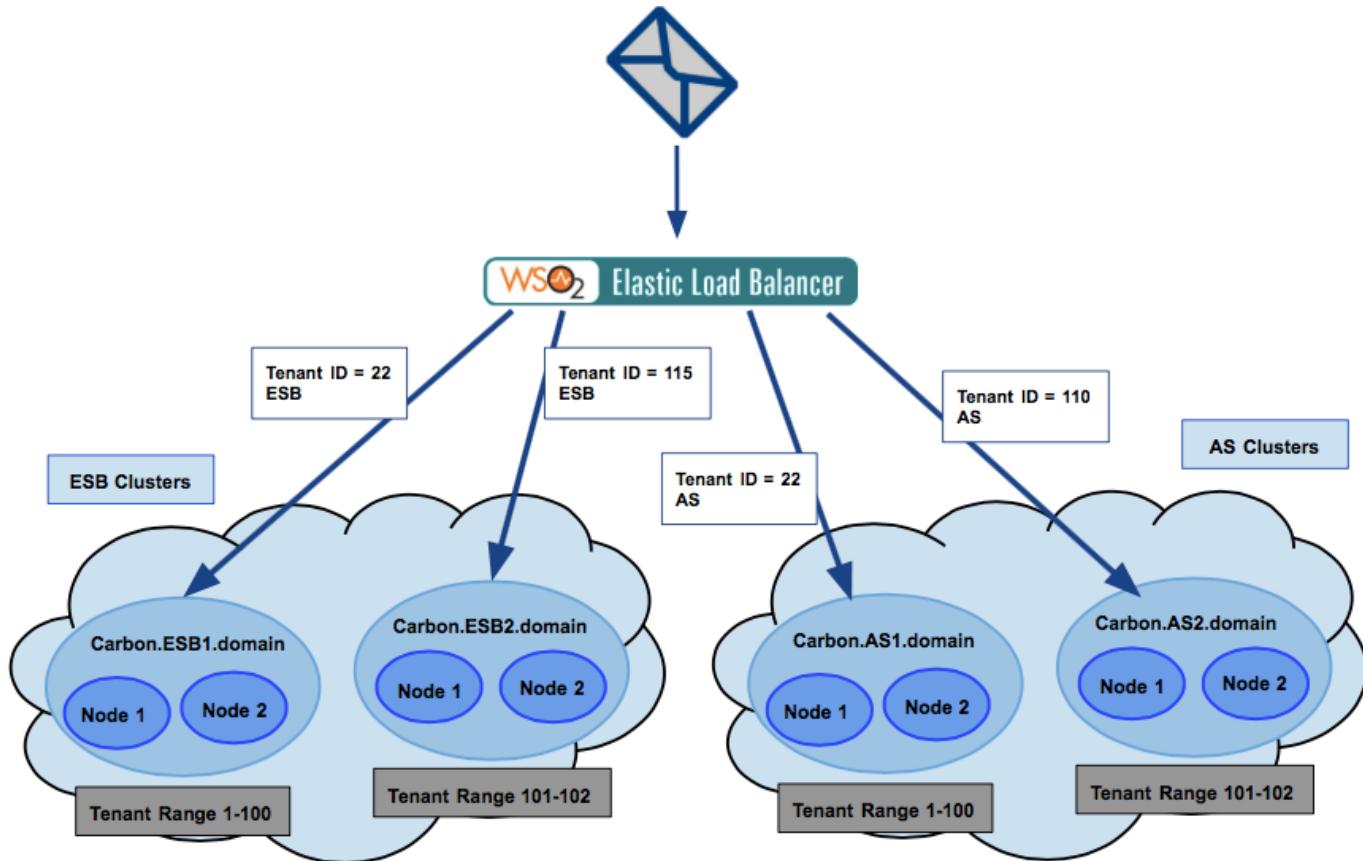
- You can access the public store (`https://localhost:9443/store`), go to the `domain1.com` store, log in to it and subscribe to its APIs.
- You can also browse the other tenant stores listed in the public store. But, within other tenant stores, you can only subscribe to the APIs to which your tenant domain is permitted to subscribe to. At the time an API is created, the API creator can specify which tenants are allowed to subscribe to the API. For information, see [API Subscriptions](#).

## Tenant-Aware Load Balancing using WSO2 ELB

WSO2 Elastic Load Balancer is currently retired.

Tenant partitioning is required in a clustered deployment to be able to scale to large numbers of tenants. There can be multiple clusters for a single service and each cluster would have a subset of tenants as illustrated in the diagram below. In such situations, the load balancers need to be tenant aware in order to route the requests to the required tenant clusters. They also need to be service aware since it is the service clusters which are partitioned according to the clients.

The following example further illustrates how this is achieved in WSO2 Elastic Load Balancer (ELB).



A request sent to a load balancer has the following host header to identify the cluster domain:

<https://appserver.cloud-test.wso2.com/carbon.as1.domain/carbon/admin/login.jsp>

In this URL:

- appserver.cloud-test.wso2.com is the service domain which allows the load balancer to identify the service.
- carbon.as1.domain.com is the tenant domain which allows the load balancer to identify the tenant.

Services are configured with their cluster domains and tenant ranges in the `ELB_HOME/repository/conf/loadbalancer.conf` file. These cluster domains and tenant ranges are picked by the load balancer when it loads.

The following is a sample configuration of the `loadbalancer.conf` file.

```
appserver {
# multiple hosts should be separated by a comma.
hosts appserver.cloud-test.wso2.com;

domains {
carbon.as1.domain {
tenant_range 1-100;
}
carbon.as2.domain {
tenant_range 101-200;
}
}
```

In the above configuration, there is a host address which maps to the application server service. If required, you can enter multiple host addresses separated by commas.

There are two cluster domains defined in the configuration. The cluster domain named `carbon.as1.domain` is used to load the range of tenants with IDs 1-100. The other cluster domain named `carbon.as2.domain` is used to load the tenants with IDs 101-200.

If the tenant ID of `abc.com` is 22, the request will be directed to the `Carbon.AS1.domain` cluster.

## Adding Internationalization and Localization

The API Manager comes with two Web interfaces as API Publisher and API Store. The following steps show an example of how to localize the API Publisher UI. Same instructions apply to localize the API Store.

### Changing the browser settings

1. Follow the instructions in your Web browser's user guide and set the browser's language to a preferred one. For example, in Google Chrome, you set the language using the **Settings -> Show advanced settings -> Languages** menu.
2. Set the browser's encoding type to UTF-8.

#### Introduction to resource files

3. Go to `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher` directory where `<AM_HOME>` is the API Manager distribution's home.
4. There are two types of resource files used to define localization strings in the API Manager.
  - The resource file used to store the strings defined in .jag files according to browser locale (For example, `locale_en.json`) is located in `.../publisher/site/conf/locales/jaggery` folder.
  - The resource file `i18nResources.json`, which is used to store strings defined in client-side javascript files such as pop-up messages when a UI event is triggered, is located in `.../publisher/site/conf/locales/js` folder.

For example,

```
./locales/jaggery:  
locale_en.json  
  
./locales/js:  
i18nResources.json
```

To implement localization support for jaggery, we use its in-built script module 'i18n'. For more information, refer to <http://jaggeryjs.org/apidocs/i18n.jag>.

#### Localizing strings in Jaggery files

5. To localize the API publisher to Spanish, first localize the strings defined in jaggery files. Create a new file by the name `locale_{localeCode}.json` inside `...publisher/site/conf/locales/jaggery` folder. For example, if the language set in the browser is Spanish, the locale code is `es` and the file name should be `locale_es.json`.
6. Add the key-value pairs to `locale_es.json` file. For an example on adding key value pairs, refer to `locale_en.json` file in `...publisher/site/conf/locales/jaggery` folder. It is the default resource file for jaggery.

In addition, a section of a sample `locale_es.json` file is shown below for your reference.

```
{
  "name" : "Nombre",
  "context" : "Contexto",
  "version" : "Versión",
  "description" : "Descripción",
  "visibility" : "Visibilidad",
  "thumbnail" : "Uña del pulgar",
  "endpoint" : "Producción URL",
  "sandbox" : "Cajón de arena URL"
}
```

### Localizing strings in client-side Javascript files

7. To localize the javascript UI messages, navigate to publisher/site/conf/locales/js folder and update **i18nResources.json** file with relevant values for the key strings.
8. Once done, open the API Publisher web application in your browser (<https://<YourHostName>:9443/publisher>).
9. Note that the UI is now changed to Spanish.

## Configuring Single Sign-on with SAML2

Single sign-on (SSO) allows users, who are authenticated against one application, gain access to multiple other related applications without having to repeatedly authenticate themselves. It also allows the Web applications gain access to a set of back-end services with the logged-in user's access rights, and the back-end services can authorize the user based on different claims like user role.

The **Single Sign-On with SAML 2.0** feature in the API Manager is implemented according to the SAML 2.0 browser-based SSO support that is facilitated by WSO2 Identity Server (IS). This feature is available in any IS version from 4.1.0 onwards. We use **IS 5.0.0** in this guide. WSO2 Identity Server acts as an identity service provider of systems enabled with single sign-on, while the Web applications act as SSO service providers. Using this feature, you can configure SSO across the API Publisher and Store. After configuring, you can access the API Store or API Publisher in a single authentication attempt.

 To learn more about Single Sign-On with WSO2 Identity Server, refer the following article on WSO2 library: <http://wso2.org/library/articles/2010/07/saml2-web-browser-based-sso-wso2-identity-server>

The topics below explain the configurations:

- [Sharing the user store](#)
- [Sharing the registry space](#)
- [Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider](#)
- [Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers](#)

### Sharing the user store

First, point both WSO2 IS and WSO2 API Manager to a single user store using the instructions given in section [Configuring User Stores](#). You do this to make sure that a user who tries to log in to the API Manager console, the API Store or the Publisher is authorized. When a user tries to log in to either of the three applications, s/he is redirected to the configured identity provider (WSO2 IS in this case) where s/he provides the login credentials to be authenticated. In addition to this, the user should also be authorized by the system as some user roles do not have permission to perform certain actions. For the purpose of authorization, the IS and API Manager need to have a shared user store and user management database (by default, this is the H2 database in the `<APIM_HOME>/repository/conf/user-mgt.xml` file) where the user's role and permissions are stored.

For example, let's take a common JDBC user store (MySQL) for both IS and API Manager.

1. Create a MySQL database (e.g., `410_um_db`) and run the `<AM_HOME>/dbscripts/mysql.sql` script on it to create the required tables. If you are using a different database type, find the relevant script from the `<AM_HOME>/dbscripts` directory.
2. Open `<AM_HOME>/repository/conf/datasources/master-datasources.xml` file and add the datasource configuration for the database that you use for the shared user store and user management

information. For example,

```
<datasource>
    <name>WSO2_UM_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2UMDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/410_um_db</url>
            <username>username</username>
            <password>password</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

3. Add the same datasource configuration above to `<IS_HOME>/repository/conf/datasources/master-data sources.xml` file.
4. Copy the database driver JAR file to the `<IS_HOME>/repository/components/lib` and `<AM_HOME>/repository/components /lib` directories.
5. Open `<AM_HOME>/repository/conf/user-mgt.xml` file. The `dataSource` property points to the default H2 database. Change it to the `jndiConfig` name given above (i.e., `jdbc/WSO2UMDB`). This changes the datasource reference that is pointing to the default H2 database.

```
<Realm>
    <Configuration>
        ...
        <Property name="dataSource">jdbc/WSO2UMDB</Property>
    </Configuration>
    ...
</Realm>
```

6. Add the same configuration above to the `<IS_HOME>/repository/conf/user-mgt.xml` file.
7. The Identity Server has an embedded LDAP user store by default. As this is enabled by default, follow the instructions in [Internal JDBC User Store Configuration](#) to disable the default LDAP and enable the JDBC user store instead.

## Sharing the registry space

In a multi-tenanted environment, by default, the Identity Server uses the key store of the super tenant to sign SAML responses. The API Store and Publishers are already registered as SPs in the super tenant. However, if you want the Identity Server to use the registry key store of the tenant that the user belongs to, you can create a common registry database and mount it on both the IS and the APIM.

1. Create a MySQL database (e.g., `registry`) and run the `<IS_HOME>/dbscripts/mysql.sql` script on it to create the required tables.  
If you are using a different database type, find the relevant script from the `<IS_HOME>/dbscripts` directory.

2. Add the following datasource configuration to both the <IS\_HOME>/repository/conf/datasources/master-datasources.xml and <AM\_HOME>/repository/conf/datasources/master-datasources.xml files.

```
<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used for registry</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

<url>jdbc:mysql://localhost:3306/registry?autoReconnect=true&relaxAutoCommit=true&lt;/url>
            <username>apiuser</username>
            <password>apimanager</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

3. Create the registry mounts by inserting the following sections into the <IS\_HOME>/repository/conf/registry.xml file.

 When doing this change, do not replace the existing <dbConfig> for "wso2registry". Simply add the following configuration to the existing configurations.

```
<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>

<mount path="/_system/governance" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/governance</targetPath>
</mount>

<mount path="/_system/config" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/config</targetPath>
</mount>
```

4. Repeat the above step in the <AM\_HOME>/repository/conf/registry.xml file as well.

Next, let us look at the SSO configurations.

### Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider

1. Start the IS server and log in to its Management Console UI (<https://localhost:9443/carbon>).

If you use login pages that are hosted externally to log in to the Identity Server, give the absolute URLs of those login pages in the authenticators.xml and application-authenticators.xml files in the <IS\_HOME>/repository/conf/security directory.

2. Select **Add** under **Service Providers** menu.



3. Give a service provider name and click **Register**.

Add Service Provider

Basic Information	
Service Provider Name: <sup>*</sup>	<input type="text" value="API_Manager"/>
<small>⑦ A unique name for the service provider</small>	
Description:	<input type="text"/>
<small>⑦ A meaningful description about the service provider</small>	
<input style="border: 2px solid red; border-radius: 5px; padding: 2px 10px; margin-right: 10px;" type="button" value="Register"/> <input type="button" value="Cancel"/>	

In a multi tenanted environment, for all tenants to be able to log in to the APIM Web applications, do the following:

- Click the **SaaS Application** option that appears after registering the service provider.

Basic Information

Service Provider Name: <sup>*</sup>	<input type="text" value="API_Manager"/>
<small>⑦ A unique name for the service provider</small>	
Description:	<input type="text"/>
<small>⑦ A meaningful description about the service provider</small>	
<input type="checkbox"/> <b>SaaS Application</b>	

If not, only users in the current tenant domain (the one you are defining the service provider in) will be allowed to log in to the Web application and you have to register new service providers for all Web applications (API Store and API Publisher in this case) from each tenant space separately. For example, let's say you have three tenants as TA, TB and TC and you register the service provider in TA only. If you tick the **SaaS Application** option, all users in TA, TB, TC tenant domains will be able to log in. Else, only users in TA will be able to log in.

- Add the following inside the <SSOService> element in the <IS\_HOME>/repository/conf/identity.xml file and restart the server.

```
<SSOService>

<UseAuthenticatedUserDomainCrypto>true</UseAuthenticatedUserDomainCrypto>
...
</SSOService>
```

If not, you get an exception as SAML response signature verification fails.

- Because the servers in a multi-tenanted environment interact with all tenants, all nodes should share the same user store. Therefore, make sure you have a shared registry (JDBC mount, WSO2 Governance Registry etc.) instance across all nodes.
4. You are navigated to the detailed configuration page. Expand **SAML2 Web SSO Configuration** inside the **Inbound Authentication Configuration** section.
  5. Provide the configurations to register the API Publisher as the SSO service provider. These sample values may change depending in your configuration.
    - Issuer : API\_PUBLISHER
    - Assertion Consumer URL : [https://localhost:9443/publisher/jagg/jaggery\\_acs.jag](https://localhost:9443/publisher/jagg/jaggery_acs.jag). Change the IP and port accordingly. This is the URL for the acs page in your running publisher app.
    - Select the following options:
      - **Use fully qualified username in the NameID**
      - **Enable Response Signing**
      - **Enable Assertion Signing**
      - **Enable Single Logout**
    - Click **Register** once done.

For

example:

## Register New Service Provider

**New Service Provider**

<b>Issuer *</b>	<input type="text" value="API_PUBLISHER"/>
<b>Assertion Consumer URL *</b>	<input type="text" value="localhost:9443/publisher/jagg/jaggery_acs.jag"/>
<b>NameID format</b>	<input type="text" value="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"/>
<input checked="" type="checkbox"/> <b>Use fully qualified username in the NameID</b>	
<input type="checkbox"/> <b>Define Claim Uri for NameID</b> <input type="text" value="http://wso2.org/claims/otherphone"/>	
<input checked="" type="checkbox"/> <b>Enable Response Signing</b>	
<input checked="" type="checkbox"/> <b>Enable Assertion Signing</b>	
<input type="checkbox"/> <b>Enable Signature Validation in Authentication Requests and Logout Requests</b>	
<b>Certificate Alias</b>	<input type="text" value="wso2carbon.cert"/>
<input type="checkbox"/> <b>Enable Assertion Encryption</b>	
<b>Certificate Alias</b>	<input type="text" value="wso2carbon.cert"/>
<input checked="" type="checkbox"/> <b>Enable Single Logout</b>	
<b>Custom Logout URL</b>	
<input type="checkbox"/> <b>Enable Attribute Profile</b>	

6. Similarly, provide the configurations to register the API Store as the SSO service provider. These sample

values may change depending in your configuration.

- Issuer : API\_STORE
  - Assertion Consumer URL : [https://localhost:9443/store/jagg/jaggery\\_acs.jag](https://localhost:9443/store/jagg/jaggery_acs.jag). Change the IP and port accordingly. This is the URL for the acs page in your running store app.
  - Select the following options:
    - **Use fully qualified username in the NameID**
    - **Enable Response Signing**
    - **Enable Assertion Signing**
    - **Enable Single Logout**
  - Click **Register** once done.
7. Make sure that the `<responseSigningEnabled>` element is set to `true` in both the following files:
- `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json`
  - `<AM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json`

## Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

1. Open `<AM_Home>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` and modify the following configurations found under **ssoConfiguration**.
  - **enabled**: Set this value to `true` to enable SSO in the application
  - **issuer**: API\_PUBLISHER. This value can change depending on the **Issuer** value defined in WSO2 IS SSO configuration above.
  - **identityProviderURL**: <https://localhost:9444/samlss>. Change the IP and port accordingly. This is the redirecting SSO URL in your running WSO2 IS server instance.
  - **keyStoreName**: The keystore of the running IDP. As you use a remote instance of WSO2 IS here, you can import the public certificate of the IS keystore to the APIM and then point to the APIM keystore. The default keystore of the APIM is `<APIM_HOME>/repository/resources/security/wso2carbon.jks`. **Be sure to give the full path of the keystore here.**
  - **keyStorePassword**: Password for the above keystore
  - **identityAlias**: wso2carbon
2. Similarly, configure the API Store with SSO. The only difference in API Store SSO configurations is setting **API\_STORE** as the **issuer**.

 **Tip:** If you configure SSO for the API Manager's Management Console as well, you must reduce the priority of the SAML2SSOAuthenticator configuration in the `<APIM_HOME>/repository/conf/security/authenticators.xml` file.

If not, there will be login issues in the API Publisher/Store. This is because the `SAML2SSOAuthenticator` handler does not process only SAML authentication requests. If you set its priority higher than that of the `BasicAuthenticator` handler, the `SAML2SSOAuthenticator` tries to process the basic authentication requests as well.

```
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
  <Priority>0</Priority>
  ...
</Authenticator>
```

3. Access the API Publisher : <https://localhost:<Port number>/publisher> (e.g., <https://localhost:9443/publisher>). Observe the request redirect to the WSO2 IS SAML2.0 based SSO login page. For example,



4. Enter user credentials. If the user authentication is successful against WSO2 IS, it will redirect to the API Publisher Web application with the user already authenticated.
5. Access the API Store application, click its **Login** link (top, right-hand corner) and verify that the same user is already authenticated in API Store.

**!** Even with SSO enabled, if the user doesn't have sufficient privileges to access API Publisher/Store or any other application, s/he will not be authorized to access them.

**i** The steps above explain how to configure SSO between the API Publisher and Store Jagger applications, using WSO2 IS as the IDP. If there are many WSO2 products in your environment, you can configure SSO for the management consoles of those products by changing the `SAML2SSOAuthenticator` configuration in `<PRODUCT_HOME>/repository/conf/security/authenticators.xml` file as follows:

- Set disabled attributes in `<Authenticator>` element to `false`
- `ServiceProviderID` : In this example, it is the issuer name of the service provider created in step 1
- `IdentityProviderSSOServiceURL` : In this example, it is the Identity Server port

```

<Authenticator name="SAML2SSOAuthenticator" disabled="false">
    <Priority>10</Priority>
    <Config>
        <Parameter
            name="LoginPage">/carbon/admin/login.jsp</Parameter>
        <Parameter name="ServiceProviderID">carbonserver1</Parameter>
        <Parameter
            name="IdentityProviderSSOServiceURL">https://localhost:9444/samlsso</Parameter>
        <Parameter
            name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
    </Config>

```

Make sure the `<priority>` of the `SAML2SSOAuthenticator` is less than that of the `BasicAuthenticator` handler. See [here](#) for more information.

## Changing the Default Transport

APIs are synapse configurations in the back-end and API Manager accesses them using HTTP-NIO transport by default. You can switch to a different transport such as PassThrough. To change the default transport of API Manager, go to `<APIM_HOME>/repository/conf/axis2` folder and rename `axis2.xml_PT` file to `axis2.xml`. Similarly, you can switch back to NHTTP by simply renaming `axis2.xml_NHTTP` file to `axis2.xml`.

The following topics explain HTTP-NIO and PassThrough transports:

- [HTTP-NIO transport](#)
- [HTTP PassThrough transport](#)

#### **HTTP-NIO transport**

**HTTP-NIO transport** is a module of the Apache Synapse project. Apache Synapse as well as WSO2 APIM ship the HTTP-NIO transport as the default HTTP transport implementation. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.nhttp.HttpCoreNIOListener` and `org.apache.synapse.transport.nhttp.HttpCoreNIOListener` respectively. These classes are available in the JAR file named `synapse-nhttp-transport.jar`. This non-blocking transport implementation improves performance. The transport implementation is based on Apache HTTP Core - NIO and uses a configurable pool of non-blocking worker threads to grab incoming HTTP messages off the wire.

#### **Transport receiver parameters**

 In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

Parameter Name	Description	Required	Possible Values	Default Value
port	The port on which this transport receiver should listen for incoming messages.	No	A positive integer less than 65535	8280
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	<i>true</i>	
bind-address	The address of the interface to which the transport listener should bind.	No	A host name or an IP address	127.0.0.1
hostname	The host name of the server to be displayed in service EPRs, WSDLs etc. This parameter takes effect only when the WSDLEPRPrefix parameter is not set.	No	A host name or an IP address	localhost
WSDLEPRPrefix	A URL prefix which will be added to all service EPRs and EPRs in WSDLs etc.	No	A URL of the form <protocol>://<hostname>:<port>/	

#### **Transport sender parameters**

Parameter Name	Description	Required	Possible Values	Default Value
http.proxyHost	If the outgoing messages should be sent through an HTTP proxy server, use this parameter to specify the target proxy.	No	A host name or an IP address	
http.proxyPort	The port through which the target proxy accepts HTTP traffic.	No	A positive integer less than 65535	

http.nonProxyHosts	The list of hosts to which the HTTP traffic should be sent directly without going through the proxy.	No	A list of host names or IP addresses separated by ' '	
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	<i>true</i>	

### HTTP PassThrough transport

HTTP PassThrough Transport is the default, non-blocking HTTP transport implementation based on HTTP Core NIO and is specially designed for streaming messages. It is similar to the old message relay transport, but it does not care about the content type and simply streams all received messages through. It also has a simpler and cleaner model for forwarding messages back and forth. It can be used as an alternative to the NHTTP transport.

The HTTP PassThrough Transport is enabled by default. If you want to use the NHTTP transport instead, uncomment the relevant NHTTP transport entries in `axis2.xml` and comment out the HTTP PassThrough transport entries. The PassThrough Transport does not require the binary relay builder and expanding formatter.

### Connection throttling

With the HTTP PassThrough and HTTP NIO transports, you can enable connection throttling to restrict the number of simultaneous open connections. To enable connection throttling, edit the `<PRODUCT_HOME>/repository/conf/nhttp.properties` (for the HTTP NIO transport) or `<PRODUCT_HOME>/repository/conf/passthru.properties` (for the PassThrough transport) and add the following line: `max_open_connections = 2`

This will restrict simultaneous open incoming connections to 2. To disable throttling, delete the `max_open_connections` setting or set it to -1.

**i** Connection throttling is never exact. For example, setting this property to 2 will result in roughly two simultaneous open connections at any given time.

**i** WSO2 products do not use the HTTP/S servlet transport configurations that are in `axis2.xml` file. Instead, they use Tomcat-level servlet transports, which are used by the management console in `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.

### Configuring Caching

When an API call hits the API Gateway, the Gateway carries out security checks to verify if the token is valid. During these verifications, the API Gateway extracts parameters such as access token, API and API version that are passed on to it. Since the entire load of traffic to APIs goes through the API Gateway, this verification process needs to be fast and efficient in order to prevent overhead and delays. The API Manager uses caching for this purpose, where the validation information is cached with the token, API name and version, and the cache is stored in either the API Gateway or the key manager server.

This section covers the following:

- [API Gateway cache](#)
- [Resource cache](#)
- [Key Manager cache](#)
- [Response cache](#)
- [API Store cache](#)

**✓** Apart from response caching, all the other caches are enabled by product. When the API Manager components are clustered, they work as distributed caches. This means that a change done by one node is visible to another node in the cluster.

### **API Gateway cache**

When caching is enabled at the Gateway and a request hits the Gateway, it first populates the cached entry for a given token. If a cache entry does not exist in cache, it calls the key manager server. This process is carried out using Web service calls. Once the key manager server returns the validation information, it gets stored in the Gateway. Because the API Gateway issues a Web service call to the key manager server only if it does not have a cache entry, this method reduces the number of Web service calls to the key manager server. Therefore, it is faster than the alternative method.

By default, the API Gateway cache is enabled by setting the `<EnableGatewayKeyCache>` element to true in `<API M_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayKeyCache>true</EnableGatewayKeyCache>
```

### **Clearing the API Gateway cache**

To remove old tokens that might still remain active in the Gateway cache, you configure the `<RevokeAPIURL>` element in `api-manager.xml` file by providing the URL of the [Revoke API](#) that is deployed in the API Gateway node. The revoke API invokes the cache clear handler, which extracts information from transport headers of the revoke request and clears all associated cache entries. If there's a cluster of API Gateways in your setup, provide the URL of the revoke API deployed in one node in the cluster. This way, all revoke requests route to the OAuth service through the Revoke API.

Given below is how to configure this in a distributed API Manager setup.

1. In the `api-manager.xml` file of the key manager node, point the revoke endpoint as follows:

```
<RevokeAPIURL>https:// ${carbon.local.ip} : ${https.nio.port} /revoke</RevokeAPIURL>
```

2. In the API Gateway, point the Revoke API to the OAuth application deployed in the key manager node. For example,

```
<api name="_WSO2AMRevokeAPI_" context="/revoke">
    <resource methods="POST" url-mapping="/*" faultSequence="_token_fault_">
        <inSequence>
            <send>
                <endpoint>
                    <address
uri="https://keymgt.wso2.com:9445/oauth2/revoke"/>
                </endpoint>
            </send>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerCacheExtensionHandle
r"/>
    </handlers>
</api>
```

### **Resource cache**

An API's resources are HTTP methods that handle particular types of requests such as GET, POST etc. They are similar to methods of a particular class. Each resource has parameters such as its throttling level, Auth type etc.

## Resources



### /default

<b>GET</b>	/* <a href="#">+ Summary</a>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>POST</b>	/* <a href="#">+ Summary</a>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>PUT</b>	/* <a href="#">+ Summary</a>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>DELETE</b>	/* <a href="#">+ Summary</a>	Application & Application User	Unlimited	<a href="#">+ Scope</a>
<b>OPTIONS</b>	/* <a href="#">+ Summary</a>	None	Unlimited	<a href="#">+ Scope</a>

Users can make requests to an API by calling any one of the HTTP methods of the API's resources. The API Manager uses the resource cache at the Gateway node to store the API's resource-level parameters (Auth type and throttling level). The cache entry is identified by a cache key, which is based on the API's context, version, request path and HTTP method. Caching avoids the need to do a separate back-end call to check the Auth type and throttling level of a resource, every time a request to the API comes. It improves performance.

Note that if you change a resource's parameters such as the Auth type through the UI, it takes about 15 minutes to refresh the resource cache. During that time, the server returns the old Auth type from the cache. If you want the changes to be reflected immediately, please restart the server after changing the value.

By default, the resource cache is enabled by setting the `<EnableGatewayResourceCache>` element to true in `<APIM_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayResourceCache>true</EnableGatewayResourceCache>
```

### Key Manager cache

The following caches are available:

- Key cache
- JWT cache
- OAuth cache

### Key cache

In a typical API Manager deployment, the Gateway is deployed in a DMZ while the Key Manager is in MZ. By default, caching is enabled at the Gateway. If you do not like to cache token related information in a leniently secured zone, you can do that on the Key Manager side. In this method, for each and every API call that hits the API Gateway, the Gateway issues a Web service call to the Key Manager server. If the cache entry is available in the Key Manager server, it is returned to the Gateway. Else, the database will be checked for the validity of the token.

This method has low performance compared to the earlier one, but you do not have to store any security-related information at the Gateway side. Using this cache combined with the Gateway cache is not recommended.

- Disable caching at the API Gateway by adding the following entry under the `<APIGateway>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

```
<EnableGatewayKeyCache>false</EnableGatewayKeyCache>
```

- Enable the Key Manager cache by adding the following entry under the `<APIKeyValidator>` element in the `api-manager.xml` file.

```
<EnableKeyMgtValidationInfoCache>true</EnableKeyMgtValidationInfoCache>
```

## JWT cache

You sometimes pass certain enduser attributes to the backend using **JSON Web Tokens (JWT)**. If you enable JWT generation, the token is generated in the Key Manager server for each validation information object and is sent as part of the key validation response. Usually, the JWT also gets cached with the validation information object, but you might want to generate JWTs per each call. You can do this by enabling JWT caching at key manager server. Add the following entry under the `<APIKeyValidator>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

```
<EnableJWTCache>true</EnableJWTCache>
```

-  You must disable caching at the Key Manager server side in order to generate JWTs per each call. Disabling the JWT cache only works if you have enabled the Key Manager cache, which is disabled by default.

Also enable token generation by setting the following entry to `true` at the root level of the `api-manager.xml` file.

```
<APIConsumerAuthentication>
  <EnableTokenGeneration>true</EnableTokenGeneration>
</APIConsumerAuthentication>
```

## OAuth cache

The OAuth token is saved in this cache, which is enabled by default. Whenever a new OAuth token is generated, it is saved in this cache to prevent constant database calls. Unless an OAuth expires or is revoked, the same token is sent back for the same user. Therefore, you do not need to change this cached token most of the time.

### Response cache

The API Manager uses **WSO2 ESB's cache mediator** to cache response messages per each API. Caching improves performance, because the backend server does not have to process the same data for a request multiple times. To offset the risk of stale data in the cache, you set an appropriate timeout period.

You enable response caching when creating a new API or editing an existing one using the API Publisher UI. Go to the API Publisher and click the **Add API** menu (to create a new API) or the **Edit** link associated with an existing API. Then, navigate to the **Manage** tab where you find the response caching section. You can set it to `Enabled` and give a timeout value. This enables the default response caching settings.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'Browse', 'Add' (which is highlighted with a red box), 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', and 'Tier Permissions'. The main area has a progress bar at the top with three steps: 'Design', 'Implement', and 'Manage' (which is also highlighted with a red box). Below the progress bar, the title 'PhoneVerify1 : /phone/1.0.0' is shown. Underneath, the 'Configurations' section contains several settings: 'Make this default version' (checkbox is off), 'Tier Availability' (set to 'Bronze'), 'Transports' (HTTP and HTTPS are checked), 'Sequences' (checkbox is checked, with a note to select a custom sequence), and a table for 'In Flow', 'Out Flow', and 'Fault Flow' with dropdown menus for each. At the bottom of the configurations section, there are fields for 'Response Caching' (set to 'Enabled') and 'Cache Timeout (seconds)' (set to '300'), both of which are highlighted with a red box. There's also a 'Subscriptions' dropdown set to 'Available to current tenant'.

To change the default response caching settings, edit the following cache mediator properties in <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file:

Property	Description
collector	<ul style="list-style-type: none"> <li>• true: specifies that the mediator instance is a response collection instance</li> <li>• false: specifies that the mediator instance is a cache serving instance</li> </ul>
max Message Size	Specifies the maximum size of a message to be cached in bytes. An optional attribute, with the default value set to unlimited.
maxSize	Defines the maximum number of elements to be cached
hashGenerator	<p>Defines the hash generator class.</p> <p>When caching response messages, a hash value is generated based on the request's URI, transport headers and the payload (if available). WSO2 has a default REQUESTHASHGenerator class written to generate the hash value. See sample <a href="#">here</a>.</p> <p>If you want to change this default implementation (for example, to exclude certain headers), you can write a new hash generator implementation by extending the REQUESTHASHGenerator and overriding its getDigest( ) method. Once done, add the new class as the hashGenerator attribute of the &lt;cache&gt; element in the velocity_template.xml file.</p>

#### API Store cache

The API Store has several caches to reduce the page-load times and increase its responsiveness when multiple users access it simultaneously.

- **Tag cache:** This cache saves the API's tags after they have been retrieved from registry. If your APIs and

associated tags change frequently, it is recommended to configure a smaller cache refresh time (in milliseconds). This cache disabled by default. To enable it, uncomment the `<TagCacheDuration>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

- **Recently-added-API cache:** This cache saves the five most recently added APIs. It is disabled by default. If you have multiple API modifications during a short time period, it is recommended to not enable this cache. To enable it, set the `<EnableRecentlyAddedAPICache>` to true in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

## Working with Databases

The default databases that WSO2 products uses to store registry, user manager and product-specific data are the H2 databases in `<PRODUCT_Home>/repository/database` as follows:

- `WSO2CARBON_DB.h2.db`: used to store registry and user manager data
- `WSO2AF_DB.h2.db`: used to store App Factory specific data

These embedded H2 databases are suitable for development, testing, and some production environments. For most production environments, however, we recommend you to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. You can use the scripts provided with WSO2 products to install and configure several other types of relational databases, including MySQL, IBM DB2, Oracle, and more.

The following sections explain how to change the default databases:

- [Setting up the Physical Database](#)
- [Managing Datasources](#)

### Setting up the Physical Database

The topics in this section describe how to use scripts in `<PRODUCT_HOME>/dbscripts/` folder to set up each type of physical database. After you set up the database, you create datasources to connect to it.

- [Setting up IBM DB2](#)
- [Setting up Derby](#)
- [Setting up H2](#)
- [Setting up IBM Informix](#)
- [Setting up Microsoft SQL](#)
- [Setting up MySQL](#)
- [Setting up MySQL Cluster](#)
- [Setting up OpenEdge](#)
- [Setting up Oracle](#)
- [Setting up Oracle RAC](#)
- [Setting up PostgreSQL](#)

#### Setting up IBM DB2

The following sections describe how to replace the default H2 databases with IBM DB2:

- [Prerequisites](#)
- [Setting up the database and users](#)
- [Setting up DB2 JDBC drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity/storage databases](#)

#### **Prerequisites**

Download the latest version of [DB2 Express-C](#) and install it on your computer.

For instructions on installing DB2 Express-C, see this ebook.

#### **Setting up the database and users**

Create the database using either [DB2 command processor](#) or [DB2 control center](#) as described below.

## Using the DB2 command processor

1. Run DB2 console and execute the db2start command in CLI to open DB2.
2. Create the database using the following command:  

```
create database <DB_NAME>
```
3. Before issuing a SQL statement, establish the connection to the database using the following command:  

```
connect to <DB_NAME> user <USER_ID> using <PASSWORD>
```
4. Grant required permissions for users as follows:

```
connect to DB_NAME
grant <AUTHORITY> on database to user <USER_ID>
```

For example:

```
db2 => connect to regdb user greg using 18091980
Database Connection Information
Database server      = DB2/LINUX 9.7.4
SQL authorization ID = GREG
Local database alias = REGDB
db2 => GRANT DBADM, CREATETAB, BINDADD, CONNECT, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, LOAD ON DATABASE TO USER user DB200001 The SQL command completed successfully.
db2 => ■
```



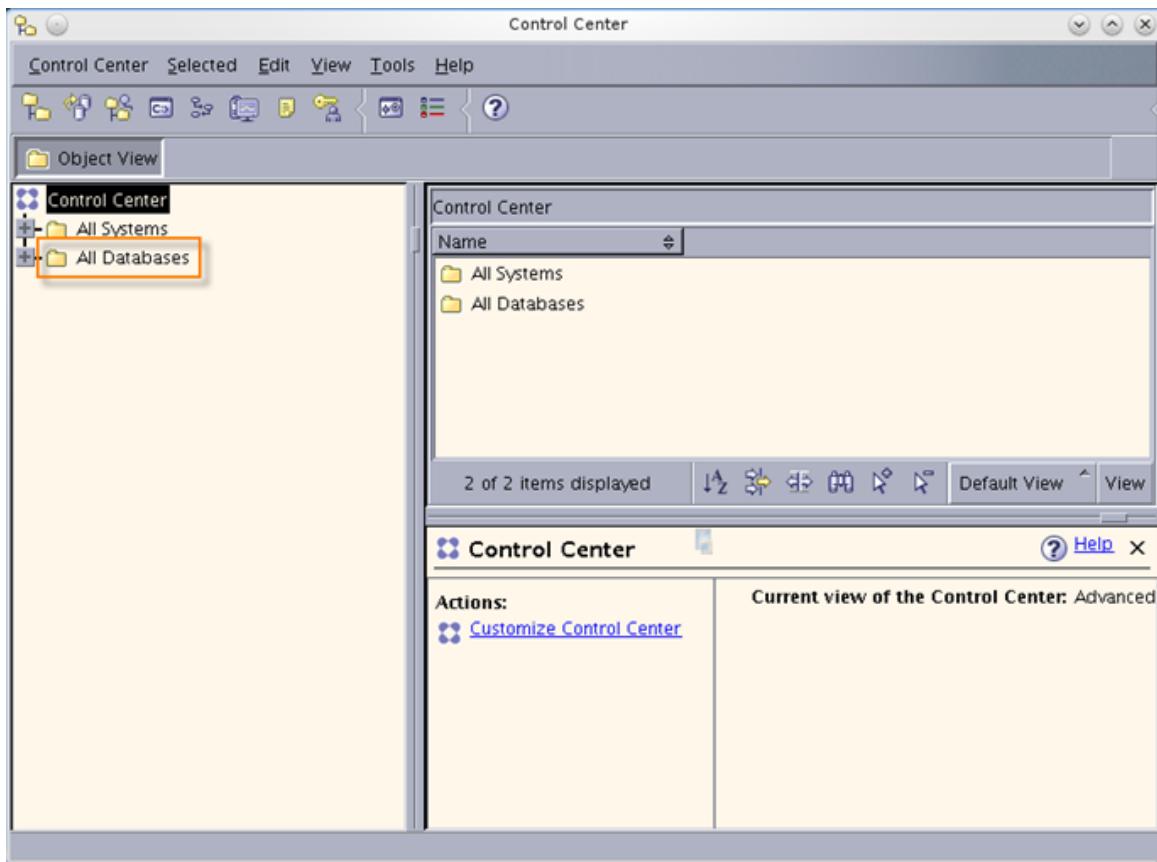
For more information on DB2 commands, see the [DB2 Express-C Guide](#).

## Using the DB2 control center

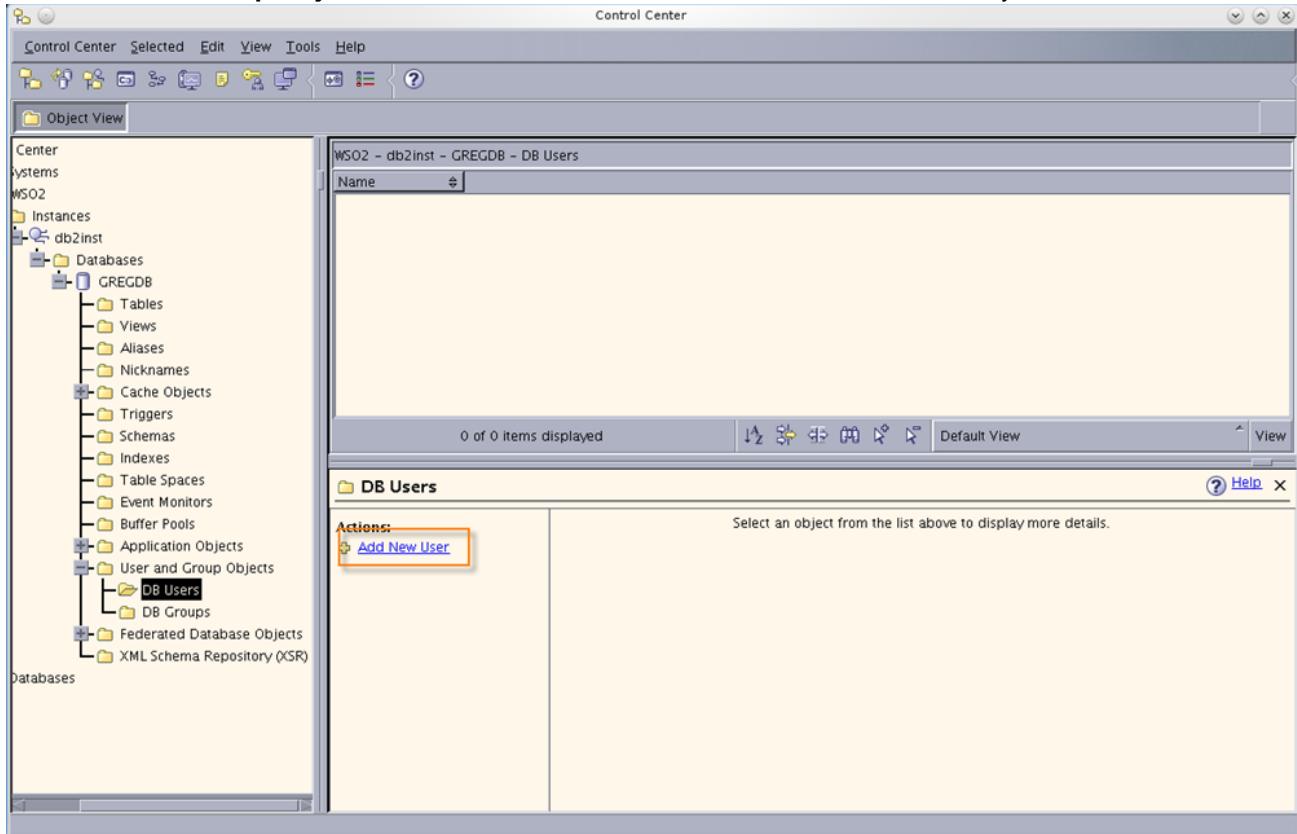
1. Open the DB2 control center using the db2cc command as follows:

```
greg@wso2:~/sqllib/bin$ ./db2cc
```

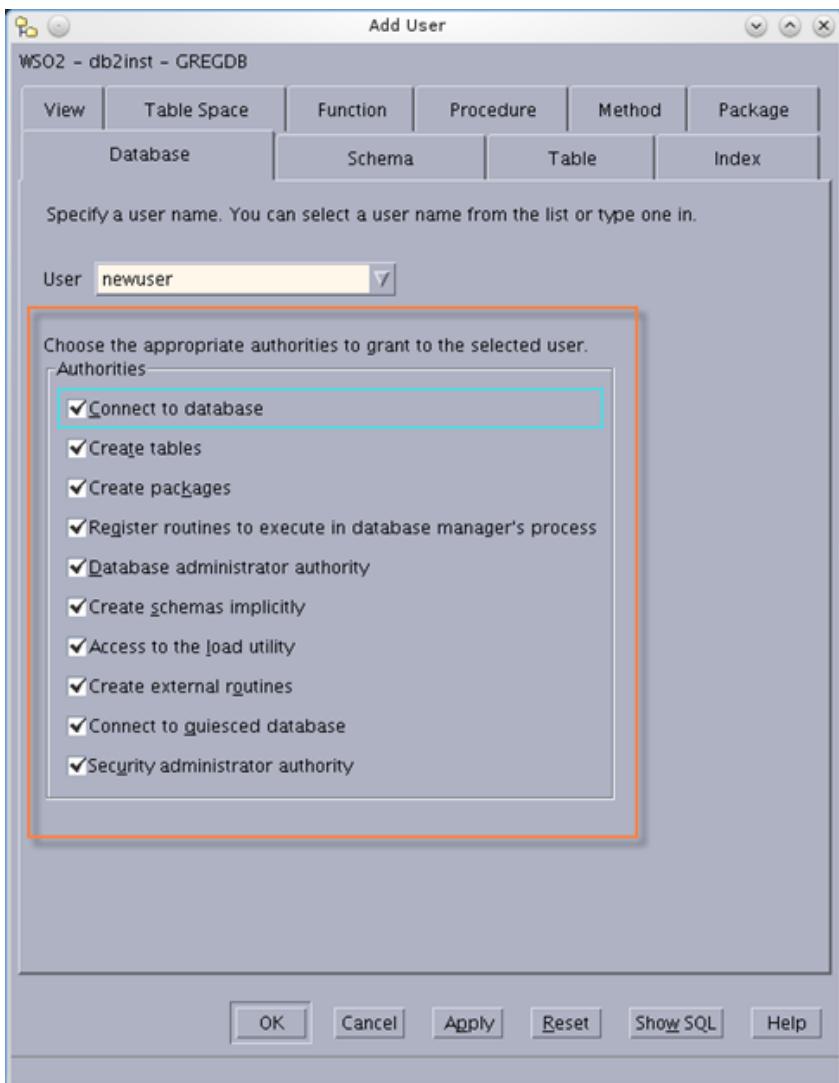
2. Right-click **All Databases** in the control center tree (inside the object browser), click **Create Database**, and then click **Standard** and follow the steps in the **Create New Database** wizard.



- Click **User and Group Objects** in the control center tree to create users for the newly created database.



- Give the required permissions to the newly created users.



### Setting up DB2 JDBC drivers

Copy the DB2 JDBC drivers (`db2jcc.jar` and `db2jcc_license_cu.jar`) from `<DB2_HOME>/SQLLIB/java/` directory to the `<PRODUCT_HOME>/repository/components/lib/` directory.

```
user@wso2:~/sqllib/java$ cp db2jcc.jar db2jcc_license_cu.jar /home/user/wso2/greg/repository/components/lib/
```

**i** `<DB2_HOME>` refers to the installation directory of DB2 Express-C, and `<PRODUCT_HOME>` refers to the directory where you run the WSO2 product instance.

### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM DB2 database to replace the default H2 database, either [change the default configurations of the `WSO2\_CARBON\_DB` datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:db2://SERVER_NAME:PORT/DB_NAME</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>360000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PROD_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### **Creating database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts in the DB2 Express-C command editor.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/db2.sql
```

2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### **Changing the product-specific/identity/storage databases**

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database that come by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the `master-datasources.xml` file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
------------------------------------------	------------------------------------------------------------------------------

<b>For the identity database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder
----------------------------------	-------------------------------------------------------------

## Setting up Derby

You can set up either an embedded Derby database or a remote database as described in the following topics:

- Setting up Embedded Derby
- Setting up Remote Derby

### Setting up Embedded Derby

The following sections describe how to replace the default H2 databases with embedded Derby:

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity/storage databases

#### Setting up the database

Follow the steps below to set up an embedded Derby database:

1. Download [Apache Derby](#).
2. Install Apache Derby on your computer.



For instructions on installing Apache Derby, see the [Apache Derby documentation](#).

#### Setting up the drivers

Copy `derby.jar`, `derbyclient.jar`, and `derbynnet.jar` from the `<DERBY_HOME>/lib/` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the WSO2 Carbon web application).

#### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Embedded Derby database to replace the default H2 database, either [change the default configurations of the WSO2\\_CARBON\\_DB datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

#### Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>

            <driverClassName>org.apache.derby.jdbc.EmbeddedDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

## Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

## Creating database tables

You can create database tables by executing the database scripts as follows:

1. Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory as illustrated below:

```
client@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
ij version 10.8
ij> ■
```

2. Create the database and connect to it using the following command inside the `ij` prompt:

```
connect 'jdbc:derby:repository/database/WSOCARBON_DB;create=true';
```

**i** Replace the database file path in the above command with the full path to your database.

3. Exit from the the `ij` tool by typing the `exit` command.

```
exit;
```

4. Log in to the `ij` tool with the username and password that you set in `registry.xml` and `user-mgt.xml`:

```
connect 'jdbc:derby:repository/database/WSOCARBON_DB' user 'regadmin' password
'regadmin';
```

5. Use the scripts given in the following locations to create the database tables:

- To create tables for the **registry and user manager database (WSOCARBON\_DB)**, run the below command:

```
run '<PRODUCT_HOME>/dbscripts/derby.sql';
```

**i** Now the product is running using the embedded Apache Derby database.

6. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-D`

setup parameter as follows:

- For Windows: <PRODUCT\_HOME>/bin/wso2server.bat -Dsetup
- For Linux: <PRODUCT\_HOME>/bin/wso2server.sh -Dsetup

**i** The product is configured to run using an embedded Apache Derby database.

**i** In contrast to setting up with remote Derby, when setting up with the embedded mode, set the database driver name (the `driverClassName` element) to the value `org.apache.derby.jdbc.EmbeddedDriver` and the database URL (the `url` element) to the database directory location relative to the installation. In the above sample configuration, it is inside the `<DERBY_HOME>/WSO2_CARBON_DB/` directory.

Changing the product-specific/identity/storage databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database that come by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the `master-datasources.xml` file by the name `wso2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

### Setting up Remote Derby

The following sections describe how to replace the default H2 databases with a remote Derby database:

- [Setting up the database](#)
- [Setting up the drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity databases](#)

Setting up the database

Follow the steps below to set up a remote Derby database.

1. Download [Apache Derby](#).
2. Install Apache Derby on your computer.

**i** For instructions on installing Apache Derby, see the [Apache Derby documentation](#).

3. Go to the `<DERBY_HOME>/bin/` directory and run the Derby network server start script. Usually it is named `startNetworkServer`.

Setting up the drivers

Copy `derby.jar`, `derbyclient.jar`, and `derbynnet.jar` from the `<DERBY_HOME>/lib/` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the Carbon web application).

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the

Remote Derby database to replace the default H2 database, either [change the default configurations of the wso2\\_carbon\\_db datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>

            <driverClassName>org.apache.derby.jdbc.ClientDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

**i** For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

**i** In contrast to setting up with embedded Derby, in the remote registry you set the database driver name (the `driverName` element) to the value `org.apache.derby.jdbc.ClientDriver` and the database URL (the `url` element) to the database remote location.

#### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file . Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

#### Creating database tables

You can create database tables by executing the following script(s):

1. Run the `ij` tool located in the <DERBY\_HOME>/bin/ directory.

```
client@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
ij version 10.8
ij> ■
```

2. Create the database and connect to it using the following command inside the `ij` prompt:

```
connect
'jdbc:derby://localhost:1527/db;user=regadmin;password=regadmin;create=true';
```

**i** Replace the database file path, user name, and password in the above command to suit your requirements.

3. Exit from the `ij` tool by typing the `exit` command as follows:

```
exit;
```

- Log in to the ij tool with the username and password you just used to create the database.

```
connect 'jdbc:derby://localhost:1527/db' user 'regadmin' password 'regadmin';
```

- You can create database tables manually by executing the following scripts.

- To create tables in the registry and user manager database (WSO2CARBON\_DB), use the below script:

```
run '<PRODUCT_HOME>/dbscripts/derby.sql';
```

- Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the -Dsetup parameter as follows:

- For Windows: <PRODUCT\_HOME>/bin/wso2server.bat -Dsetup
- For Linux: <PRODUCT\_HOME>/bin/wso2server.sh -Dsetup

**i** The product is now configured to run using a remote Apache Derby database.

#### Changing the product-specific/identity databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that come by default or set up a separate database for identity related data, the instructions are the same. In summary:

- Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
- Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/apimgt folder
<b>For the identity database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

#### Setting up H2

You can set up either an embedded H2 database or a remote H2 database using the instructions in the following topics:

- Setting up Embedded H2
- Setting up Remote H2

#### **Setting up Embedded H2**

The following sections describe how to replace the default H2 databases with Embedded H2:

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

#### Setting up the database

Download and install the H2 database engine in your computer.

**i** For instructions on installing DB2 Express-C, see [H2 installation guide](#).

#### Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.\* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:  
`<PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar`
2. Find the JAR file of the new H2 database driver (`<H2_HOME>/bin/h2-*.jar`, where `<H2_HOME>` is the H2 installation directory) and copy it to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory.

#### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2database, which stores registry and user management data. After setting up the Embedded H2 database to replace the default H2 database, either [change the default configurations of the `WSO2\_CARBON\_DB` datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

#### Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>
            <driverClassName>org.h2.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<code>url</code>	The URL of the database. The default port for a DB2 instance is 50000.
<code>username</code> and <code>password</code>	The name and password of the database user
<code>driverClassName</code>	The class name of the database driver

<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in the H2 shell or web console:

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/h2.sql
```

Follow the steps below to run the script in Web console:

1. Run the `./h2.sh` command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click **Run**.
5. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

Changing the product-specific/identity databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

### Setting up Remote H2

The following sections describe how to replace the default H2 databases with Remote H2:

- Setting up the remote H2 database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

Setting up the remote H2 database

Follow the steps below to set up a Remote H2: database.

1. Download and install the H2 database engine on your computer as follows.

**i** For instructions on installing, see the [H2 installation guide](#).

```
client@wso2:~/dtb$ wget -c http://www.h2database.com/h2-2011-09-11.zip
--2011-09-30 00:28:27-- http://www.h2database.com/h2-2011-09-11.zip
Resolving www.h2database.com... 80.74.147.171
Connecting to www.h2database.com|80.74.147.171|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6007851 (5.7M) [application/zip]
Saving to: "h2-2011-09-11.zip"

15% [=====] 923,304      111K/s  eta 45s
```

2. Go to the `<H2_HOME>/bin/` directory and run the H2 network server starting script as follows, where `<H2_HOME>` is the H2 installation directory:

```
client@wso2:~/dtb/h2/bin$ chmod 0744 h2.sh
```

3. Run the H2 database server with the following commands:

- For Linux:

```
$ ./h2.sh
```

- For Windows:

```
$ h2.bat
```



The script starts the database engine and opens a pop-up window.

4. Click **Start Browser** to open a web browser containing a client application, which you use to connect to a database. If a database does not already exist by the name you provided in the **JDBC URL** text box, H2 will automatically create a database.

#### Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.\* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:

```
<PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar
```

2. Find the JAR file of the new H2 database driver (<H2\_HOME>/bin/h2-\*.jar, where <H2\_HOME> is the H2 installation directory) and copy it to your WSO2 product's <PRODUCT\_HOME>/repository/components/lib/ directory.

#### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, wso2\_carbon\_db datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Remote H2 database to replace the default H2 database, either [change the default configurations of the wso2\\_carbon\\_db datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

#### Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default wso2\_carbon\_db datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:h2:tcp://localhost/~/registryDB;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.h2.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

## Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

## Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in H2 shell or web console:

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/h2.sql
```

Follow the steps below to run the script in Web console:

1. Run the `./h2.sh` command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click **Run**.

The screenshot shows the H2 Console interface running in a web browser. The URL is <http://127.0.1.1:8082/login.do?sessionid=0cd14a9efc1a047a86dea3d67b330b11>. The left sidebar shows a database named 'registryDB\_1' with a schema named 'INFORMATION\_SCHEMA'. The main area contains SQL scripts for creating tables such as 'REG\_CLUSTER\_LOCK', 'REG\_LOG', 'REG\_PATH', and 'REG\_CONTENT'. A red circle highlights the 'Run (Ctrl+Enter)' button at the top of the code editor. Below the code editor is a section titled 'Important Commands' with four buttons: 'Displays this Help Page', 'Shows the Command History', 'Executes the current SQL statement', and 'Disconnects from the database'.

```

CREATE TABLE IF NOT EXISTS REG_CLUSTER_LOCK (
    REG_LOCK_NAME VARCHAR(20),
    REG_LOCK_STATUS VARCHAR(20),
    REG_LOCKED_TIME TIMESTAMP,
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOCK_NAME)
);

CREATE TABLE IF NOT EXISTS REG_LOG (
    REG_LOG_ID INTEGER AUTO_INCREMENT,
    REG_PATH VARCHAR(2000),
    REG_USER_ID VARCHAR(31) NOT NULL,
    REG_LOGGED_TIME TIMESTAMP NOT NULL,
    REG_ACTION INTEGER NOT NULL,
    REG_ACTION_DATA VARCHAR(500),
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOG_ID, REG_TENANT_ID)
);

CREATE TABLE IF NOT EXISTS REG_PATH(
    REG_PATH_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_PATH_VALUE VARCHAR(2000) NOT NULL,
    REG_PATH_PARENT_ID INT,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_PATH PRIMARY KEY(REG_PATH_ID, REG_TENANT_ID)
);

CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_NAME ON REG_PATH(REG_PATH_VALUE, REG_TENANT_ID);
CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_PARENT_ID ON REG_PATH(REG_PATH_PARENT_ID, REG_TENANT_ID);

CREATE TABLE IF NOT EXISTS REG_CONTENT (
    REG_CONTENT_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_CONTENT_DATA LONGBLOB,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_CONTENT PRIMARY KEY(REG_CONTENT_ID, REG_TENANT_ID)
);

```

## 5. Restart the server.



You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

## Changing the product-specific/identity databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

## Setting up IBM Informix

The following sections describe how to replace the default H2 databases with IBM Informix:

- Prerequisites
- Creating the database
- Setting up Informix JDBC drivers
- Setting up datasource configurations

- Creating database tables
- Changing the product-specific/identity databases

### **Prerequisites**

Download the latest version of [IBM Informix](#) and install it on your computer.

### **Creating the database**

Create the database and users in Informix. For instructions on creating the database and users, see [Informix product documentation](#).

-  Do the following changes to the default database when creating the Informix database.

- Use page size as 4K or higher when creating the dbspace as shown in the following command (i.e. denoted by -k 4):  

```
onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000
```
- Add the following system environment variables.

```
export DB_LOCALE=en_US.UTF-8
export CLIENT_LOCALE=en_US.UTF-8
```

- Create a sbspace other than the dbspace by executing the following command: `onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000`
- Add the following entry to the `<INFORMIX_HOME>/etc/onconfig` file, and replace the given example sbspace name (i.e. `testspace4`) with your sbspace name: `SBSPACENAME testspace4`

### **Setting up Informix JDBC drivers**

Download the Informix JDBC drivers and copy them to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory.

-  Use Informix JDBC driver version 3.70.JC8, 4.10.JC2 or higher.

### **Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM Informix database to replace the default H2 database, either [change the default configurations of the `WSO2\_CARBON\_DB` datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```

<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <!-- IP ADDRESS AND PORT OF DB SERVER -->

<url>jdbc:informix-sqli://localhost:1533/AM_DB;CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>

        <driverClassName>com.informix.jdbc.IfxDriver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	<p>The URL of the database. The default port for a DB2 instance is 50000.</p> <p>You need to add the following configuration when specifying the connection URL as shown example above:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <span style="color: green;">✔</span> Add the following configuration to the connection URL when specifying it as shown example above: <code>CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</code> </div>
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from the pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If an object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

**i** For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the WSO2\_CARBON\_DB datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### **Creating database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (WSO2CARBON\_DB), use the below script:

```
<PRODUCT_HOME>/dbscripts/informix.sql
```

2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the -Dsetup parameter as follows:

- For Windows: <PRODUCT\_HOME>/bin/wso2server.bat -Dsetup
- For Linux: <PRODUCT\_HOME>/bin/wso2server.sh -Dsetup

### **Changing the product-specific/identity databases**

The topics above show how to change the WSO2\_CARBON\_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2AM\_DB) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/apimgt folder
<b>For the identity database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

### Setting up Microsoft SQL

The following sections describe how to replace the default H2 database with MS SQL:

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

#### Setting up the database and users

Follow the steps below to set up the Microsoft SQL database and users.

Enable TCP/IP

1. In the start menu, click **Programs** and launch **Microsoft SQL Server 2005**.
2. Click **Configuration Tools**, and then click **SQL Server Configuration Manager**.
3. Enable **TCP/IP** and disable **Named Pipes** from protocols of your Microsoft SQL server.
4. Double click **TCP/IP** to open the TCP/IP properties window, and set **Listen All** to **Yes** on the **Protocol** tab.
5. On the **IP Address** tab, disable **TCP Dynamic Ports** by leaving it blank and give a valid TCP port, so that Microsoft SQL server will listen on that port.

**i** The best practice is to use port 1433, because you can use it in order processing services.

6. Similarly, enable TCP/IP from **SQL Native Client Configuration** and disable **Named Pipes**. Also check whether the port is set correctly to 1433.
7. Restart Microsoft SQL Server.

Create the database and user

1. Open Microsoft SQL Management Studio to create a database and user.
2. Click **New Database** from the **Database** menu, and specify all the options to create a new database.
3. Click **New Login** from the **Logins** menu, and specify all the necessary options.

Grant permissions

Assign newly created users the required grants/permissions to log in, create tables, and insert, index, select, update, and delete data in tables in the newly created database, as the minimum set of SQL server permissions.

#### Setting up the JDBC driver

Download and copy the `sqljdbc4` Microsoft SQL JDBC driver file to the WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory. Use `com.microsoft.sqlserver.jdbc.SQLServerDriver` as the `<driverClassName>` in your datasource configuration in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.

#### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Microsoft SQL database to replace the default H2 database, either [change the default configurations of the `WSO2\_CARBON\_DB` datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:sqlserver://<IP>:1433;databaseName=wso2greg</url>
            <username>regadmin</username>
            <password>regadmin</password>

        <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. Change the <IP> with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.

**validationInterval**

The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.



For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### ***Creating the database tables***

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/mssql.sql
```

2. Restart the server.



You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: <PRODUCT\_HOME>/bin/wso2server.bat -Dsetup
- For Linux: <PRODUCT\_HOME>/bin/wso2server.sh -Dsetup

### ***Changing the product-specific/identity databases***

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/apimgt folder
<b>For the identity database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

### Setting up MySQL

The following sections describe how to replace the default H2 databases with MySQL:

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the registry/user management databases
- Changing the product-specific/identity databases

### Setting up the database and users

Follow the steps below to set up a MySQL database:

1. Download and install MySQL on your computer using the following command:

 For instructions on installing MySQL on MAC OS, go to [Homebrew](#).

```
sudo apt-get install mysql-server mysql-client
```

2. Start the MySQL service using the following command:

```
sudo /etc/init.d/mysql start
```

3. Log in to the MySQL client as the root user (or any other user with database creation privileges).

```
mysql -u root -p
```

4. Enter the password when prompted.

 In most systems, there is no default root password. Press the Enter key without typing anything if you have not changed the default root password.

5. In the MySQL command prompt, create the database using the following command:

```
create database regdb;
```

 For users of Microsoft Windows, when creating the database in MySQL, it is important to specify the character set as latin1. Failure to do this may result in an error (error code: 1709) when starting your cluster. This error occurs in certain versions of MySQL (5.6.x), and is related to the UTF-8 encoding. MySQL originally used the latin1 character set by default, which stored characters in a 2-byte sequence. However, in recent versions, MySQL defaults to UTF-8 to be friendlier to international users. Hence, you must use latin1 as the character set as indicated below in the database creation commands to avoid this problem. Note that this may result in issues with non-latin characters (like Hebrew, Japanese, etc.). The database creation command should be as follows:

```
mysql> create database <DATABASE_NAME> character set latin1;
```

For users of other operating systems, the standard database creation commands will suffice. For these operating systems, the database creation command should be as follows::

```
mysql> create database <DATABASE_NAME>;
```

6. Give authorization of the database to the regadmin user as follows:

```
GRANT ALL ON regdb.* TO regadmin@localhost IDENTIFIED BY "regadmin";
```

- Once you have finalized the permissions, reload all the privileges by executing the following command:

```
FLUSH PRIVILEGES;
```

- Log out from the MySQL prompt by executing the following command:

```
quit;
```

### **Setting up the drivers**

Download the MySQL Java connector JAR file, and copy it to the <PRODUCT\_HOME>/repository/components/lib/ directory.

 **Tip:** Be sure to use the connector version that is supported by the MySQL version you use. If you come across any issues due to version incompatibility, follow the steps below:

- Shut down the server and remove all existing connectors from <PRODUCT\_HOME>/repository/components/lib and <PRODUCT\_HOME>/repository/components/dropins.
- Download the connector JAR that is compatible with your current MySQL version.
- Copy the JAR file **only to** <PRODUCT\_HOME>/repository/components/lib. Files will be copied automatically to the dropins folder at the server startup.
- Start the server with the -Dsetup parameter as sh wso2server.sh -Dsetup.

### **Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, WSO2\_CARBON\_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the MySQL database to replace the default H2 database, either [change the default configurations of the WSO2\\_CARBON\\_DB datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

- Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.



Do not change the datasource name WSO2\_CARBON\_DB in the below configurations.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS ">
        <configuration>
            <url>jdbc:mysql://localhost:3306/regdb</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for MySQL is 3306
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/`. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### ***Creating database tables***

To create the database tables, connect to the database that you created earlier and run the following scripts

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

**i** You may have to enter the password for each command when prompted.

```
mysql -u regadmin -p -Dregdb < '<PRODUCT_HOME>/dbscripts/mysql.sql';
```

2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### ***Changing the registry/user management databases***

If you change the database that come by default or set up a separate database for registry or user management related data, follow the below instructions.

1. Add the datasource to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasource.xml` file . Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).

### ***Changing the product-specific/identity databases***

The topics above show how to change the `WSO2_CARBON_DB` , which is used to store registry and user manager

information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

#### Setting up MySQL Cluster

For instructions on setting up any WSO2 product with a MySQL cluster, see [this article](#), which is published in the WSO2 library.

#### Setting up OpenEdge

The following sections describe how to set up the default H2 database with OpenEdge:

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

#### ***Setting up the database and users***

Follow the steps below to set up an OpenEdge (OE) database.

1. Download and install OpenEdge on your computer.
2. Go to the `<OE_HOME>/bin/` directory and use the `proenv` script to set up the environment variables.
3. Add `<OE_HOME>/java/prosp.jar` to the `CLASSPATH` environment variable.
4. Create an empty database using the `prodb` script as follows. This script creates a database by copying an existing database provided with the installation.

```
prodb CARBON_DB <OE-installation-directory>/empty8
```

5. Start the database using the `proserve` script as follows. Provide the database name and a port as arguments to this script using the `-db` and `-S` parameters.

```
proserve -db CARBON_DB -S 6767
```

6. Use the `sqlexp` script to start the default SQL explorer that comes with the OpenEdge installation. Connect to the database you just created by using the `-db` and `-S` parameters as follows:

```
sqlexp -db CARBON_DB -S 6767
```

7. Use the following commands to create a user and grant that user the required permissions to the database:

```
CREATE USER 'wso2carbon', 'wso2carbon';
GRANT dba,resource TO 'wso2carbon';
COMMIT;
```

8. Log out from the SQL explorer by typing the following command: `exit`

#### ***Setting up the drivers***

Copy the `<OE_HOME>/java/openedge.jar` file to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory.

## Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2\_CARBON\_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the OpenEdge database to replace the default H2 database, either [change the default configurations of the WSO2\\_CARBON\\_DB datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:datadirect://localhost:6767;databaseName=CARBON_DB</url>
            <username>regadmin</username>
            <password>regadmin</password>

            <driverClassName>com.ddtek.jdbc.openedge.OpenEdgeDriver</driverClassName>
                <maxActive>80</maxActive>
                <maxWait>60000</maxWait>
                <minIdle>5</minIdle>
                <testOnBorrow>true</testOnBorrow>
                <validationQuery>SELECT 1</validationQuery>
                <validationInterval>30000</validationInterval>
            </configuration>
        </definition>
    </datasource>
```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.

<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

**i** For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/openedge.sql
```

Follow the steps below to create the database tables by executing the scripts.

1. Modify the OpenEdge script provided with the product to create the tables manually. Make a backup of the <PRODUCT\_HOME>/dbscripts/openedge.sql script under the name `openedge_manual.sql`.
2. Replace all the "/" symbols in the `openedge_manual.sql` script with the ";" symbol.
3. At the end of the `openedge_manual.sql` script, add the following line and save the script:

```
COMMIT;
```

4. Run the modified script using the SQL explorer as follows:

```
sqlexp -db CARBON_DB -S 6767 -user wso2carbon -password wso2carbon
<PRODUCT_HOME>/dbscripts/openedge_manual.sql
```

## 5. Restart the server.

- i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:
- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
  - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

## Setting up Oracle

The following sections describe how to replace the default H2 database with Oracle:

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

### **Setting up the database and users**

Follow the steps below to set up a Oracle database.

1. Create a new database by using the Oracle database configuration assistant (dbca) or manually.
2. Make the necessary changes in the Oracle `tnsnames.ora` and `listner.ora` files in order to define addresses of the databases for establishing connections to the newly created database.
3. After configuring the `.ora` files, start the Oracle instance using the following command:

```
$ sudo /etc/init.d/oracle-xe restart
```

4. Connect to Oracle using SQL\*Plus as `SYSDBA` as follows:

```
$ ./$<ORACLE_HOME>/config/scripts/sqlplus.sh sysadm/password as SYSDBA
```

5. Connect to the instance with the username and password using the following command:

```
$ connect
```

6. As `SYSDBA`, create a database user and grant privileges to the user as shown below:

```
Create user <USER_NAME> identified by password account unlock;
grant connect to <USER_NAME>;
grant create session, create table, create sequence, create trigger to
<USER_NAME>;
alter user <USER_NAME> quota <SPACE_QUOTA_SIZE_IN_MEGABYTES> on
'<TABLE_SPACE_NAME>';
commit;
```

7. Exit from the SQL\*Plus session by executing the `quit` command.

### **Setting up the JDBC driver**

1. Copy the Oracle JDBC libraries (for example, `<ORACLE_HOME/jdbc/lib/ojdbc14.jar`) to the `<PRODUCT_HOME>/repository/components/lib/` directory.
2. Remove the old database driver from the `<PRODUCT_HOME>/repository/components/dropins/` directory.

If you get a **timezone region not found** error when using the ojdbc6.jar with WSO2 servers, set the Java property as follows: `export JAVA_OPTS="-Duser.timezone='+05:30'"`

The value of this property should be the GMT difference of the country. If it is necessary to set this property permanently, define it inside the `wso2server.sh` as a new `JAVA_OPT` property.

### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle database to replace the default H2 database, either [change the default configurations of the `WSO2\_CARBON\_DB` data source](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:oracle:thin:@SERVER_NAME:PORT/DB_NAME</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
<code>url</code>	The URL of the database. The default port for a DB2 instance is 50000.
<code>username</code> and <code>password</code>	The name and password of the database user
<code>driverClassName</code>	The class name of the database driver
<code>maxActive</code>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.

<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 The default port for Oracle is 1521.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### Creating the database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL\*Plus:

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
```

## 2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### **Changing the product-specific/identity databases**

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

### **Setting up Oracle RAC**

The following sections describe how to replace the default H2 database with Oracle RAC:

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

Oracle Real Application Clusters (RAC) is an option for the Oracle Database for clustering and high availability in Oracle database environments. In the Oracle RAC environment, some of the commands used in `oracle.sql` are considered inefficient. Therefore, the product has a separate SQL script `oracle_rac.sql` for Oracle RAC. The Oracle RAC-friendly script is located in the `dbscripts` folder together with other `.sql` scripts.

**i** To test products on Oracle RAC, rename `oracle_rac.sql` to `oracle.sql` before running `-Dsetup`.

### **Setting up the database and users**

Follow the steps below to set up an Oracle RAC database.

1. Set environment variables `<ORACLE_HOME>`, `PATH`, and `ORACLE_SID` with the corresponding values `/oracle/app/oracle/product/11.2.0/dbhome_1`, `$PATH:<ORACLE_HOME>/bin`, and `orcl1` as follows:

```
[oracle@node1 ~]$ export ORACLE_HOME=/oracle/app/oracle/product/11.2.0/dbhome_1
[oracle@node1 ~]$ export PATH=$PATH:$ORACLE_HOME/bin
[oracle@node1 ~]$ export ORACLE_SID=orcl1
```

2. Connect to Oracle using SQL\*Plus as SYSDBA.

```
[oracle@node1 ~]$ sqlplus SYSDBA/1 as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 18 18:10:42 2011
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

SQL> select 2+2 from dual;

 2+2
-----
        4

SQL> create user dbgreg identified by dbgreg account unlock;
User created.

SQL> grant connect to dbgreg;
Grant succeeded.

SQL> grant create session, dba to dbgreg;
Grant succeeded.

SQL> commit;
Commit complete.

SQL> quit
Disconnected From Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options
[oracle@node1 ~]$
```

3. Create a database user and grant privileges to the user as shown below:

```
Create user <USER_NAME> identified by password account unlock;
grant connect to <USER_NAME>;
grant create session, create table, create sequence, create trigger to
<USER_NAME>;
alter user <USER_NAME> quota <SPACE_QUOTA_SIZE_IN_MEGABYTES> on
'<TABLE_SPACE_NAME>';
commit;
```

4. Exit from the SQL\*Plus session by executing the `quit` command.

### **Setting up the JDBC driver**

Copy the Oracle JDBC libraries (for example, the `<ORACLE_HOME>/jdbc/lib/ojdbc14.jar` file) to the `<PRODUCT_HOME>/repository/components/lib/` directory.

**i** Remove the old database driver from the `<PRODUCT_HOME>/repository/components/dropins/` directory when you upgrade the database driver.

### **Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, WSO2\_CARBON\_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle RAC database to replace the default H2 database, either [change the default configurations of the WSO2\\_CARBON\\_DB datasource](#), or [configure a new datasource](#) to point it to the new database as explained below.

Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)
                (ADDRESS=(PROTOCOL=TCP)(HOST=racnode1) (PORT=1521))
                (ADDRESS=(PROTOCOL=TCP)(HOST=racnode2) (PORT=1521))
                (CONNECT_DATA=(SERVICE_NAME=rac)))</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.

<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

**i** For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/registry.xml file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### ***Creating the database tables***

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL\*Plus:

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
```

2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: <PRODUCT\_HOME>/bin/wso2server.bat -Dsetup
- For Linux: <PRODUCT\_HOME>/bin/wso2server.sh -Dsetup

## **Changing the product-specific/identity databases**

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/apimgt folder
<b>For the identity database</b>	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

### **Setting up PostgreSQL**

The following sections describe how to replace the default H2 database with PostgreSQL:

- Setting up the database and login role
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

### **Setting up the database and login role**

Follow the steps below to set up a PostgreSQL database.

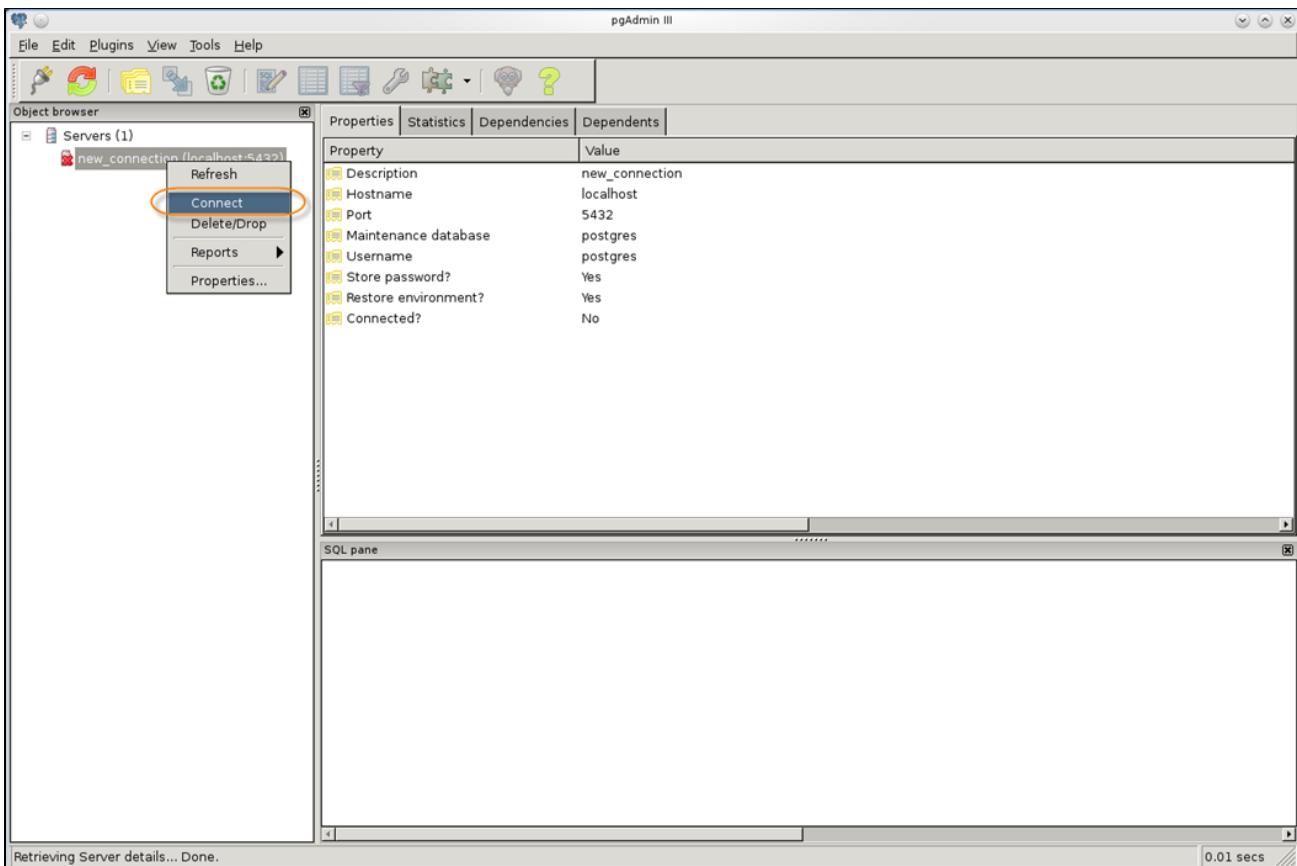
1. Install PostgreSQL on your computer as follows:

```
client@wso2:~/dtb$ sudo apt-get install postgresql
```

2. Start the PostgreSQL service using the following command:

```
client@wso2:~$ sudo /etc/init.d/postgresql start
Starting PostgreSQL 8.4 database server: main.
client@wso2:~$
```

3. Create a database and the login role from a GUI using the PGAdminIII tool.
4. To connect PGAdminIII to a PostgreSQL database server, locate the server from the object browser, right-click the client, and click **Connect**. This will show you the databases, tablespaces, and login roles as follows:



5. To create a database, click **Databases** in the tree (inside the object browser), and click **New Database**.
6. In the **New Database** dialog box, give a name to the database (for example: gregdb) and click **OK**.
7. To create a login role, click **Login Roles** in the tree (inside the object browser), and click **New Login Role**. Enter the role name and a password.

**i** These values will be used in the product configurations as described in the following sections. In the sample configuration, gregadmin will be used as both the role name and the password.

8. Optionally enter other policies, such as the expiration time for the login and the connection limit.
9. Click **OK** to finish creating the login role.

### Setting up the drivers

1. Download the PostgreSQL JDBC4 driver.
2. Copy the driver to your WSO2 product's <PRODUCT\_HOME>/repository/components/lib directory.

### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2\_CARBON\_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the PostgreSQL database to replace the default H2 database, either change the default configurations of the WSO2\_CARBON\_DB datasource, or configure a new datasource to point it to the new database as explained below. Changing the default WSO2\_CARBON\_DB datasource

Follow the steps below to change the type of the default WSO2\_CARBON\_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:postgresql://localhost:5432/gregdb</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.postgresql.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

 For more information on other parameters that can be defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PROD_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
<dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```
<Configuration>
<Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### ***Creating database tables***

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<PRODUCT_HOME>/dbscripts/postgresql.sql
```

2. Restart the server.

**i** You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### ***Changing the product-specific/identity databases***

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2AM_DB`. Change its elements with your custom values.
2. Create the database tables using the following scripts:

<b>For the product-specific database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/apimgt</code> folder
<b>For the identity database</b>	Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity</code> folder

## Managing Datasources

A datasource provides information that a server can use to connect to a database. Datasource management is provided by the following feature in the WSO2 feature repository:

Name : WSO2 Carbon - datasource management feature  
Identifier: org.wso2.carbon.datasource.feature.group

If datasource management capability is not included in your product by default, add it by installing the above feature, using the instructions given under the Feature Management section of this documentation.

Click **Data Sources** on the **Configure** tab of the product's management console to view, edit, and delete the datasources in your product instance.

- i** You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** on the **Configure** tab of the product management console. However, you cannot edit or delete the default <WSO2\_CARBON\_DB> datasource.

### Adding Datasources

If the datasource management feature is installed in your WSO2 product instance, you can add datasources that allow the server to connect to databases and other external data stores.

Use the following steps to add a datasource:

1. In the product management console, click **Data Sources** on the **Configure** tab.



2. Click **Add Data Source**.
3. Specify the required options for connecting to the database. The available options are based on the type of datasource you are creating:
  - Configuring a RDBMS Datasource
  - Configuring a Custom Datasource

After adding datasources, they will appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

- !** When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to <PROD UCT\_HOME>/repository/components/lib.

### Configuring an RDBMS Datasource

When adding a datasource, if you select RDBMS as the datasource type, the following screen appears:

### New Data Source

New Data Source

Data Source Type*	<input type="button" value="RDBMS"/>	
Name*	<input type="text"/>	
Description	<input type="text"/>	
Data Source Provider*	<input type="button" value="default"/>	
Driver*	<input type="text"/>	
URL*	<input type="text"/>	
User Name	<input type="text"/>	
Password	<input type="text"/>	
<input type="checkbox"/> Expose as a JNDI Data Source		
<input type="checkbox"/> Data Source Configuration Parameters		
<input type="button" value="Test Connection"/>	<input type="button" value="Save"/>	<input type="button" value="Cancel"/>

This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the custom datasource option. Enter values for the following fields when using the default RDBMS datasource configuration:

- **Data Source Type:** RDBMS
- **Name:** Name of the datasource (must be a unique value)
- **Data Source Provider:** Specify the datasource provider.
- **Driver:** The class name of the JDBC driver to use. Make sure to copy the JDBC driver relevant to the database engine to the <PRODUCT\_HOME>/repository/components/lib/ directory. For example, if you are using MySQL, specify com.mysql.jdbc.Driver as the driver and copy mysql-connector-java-5.5.25-bin.jar file to this directory. If you do not copy the driver to this directory when you create the datasource, you will get an exception similar to Cannot load JDBC driver class com.mysql.jdbc.Driver.
- **URL:** The connection URL to pass to the JDBC driver to establish the connection.
- **User Name:** The connection user name that will be passed to the JDBC driver to establish the connection.
- **Password:** The connection password that will be passed to the JDBC driver to establish the connection.
- **Expose as a JNDI Data Source:** Allows you to specify the JNDI datasource.
- **Data Source Configuration Parameters:** Allows you to specify the datasource connection pool parameters when creating a RDBMS datasource.

For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

After creating datasources, they appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

#### Configuring the Datasource Provider

A datasource provider connects to a source of data such as a database, accesses its data, and returns the results of the access queries. When creating a RDBMS datasource, use the default provider or link to an external provider. Default datasource provider

To use the default datasource provider, select **default**, and then enter the Driver, URL, User Name, and Password connection properties as follows:

## New Data Source

New Data Source

Data Source Type*	RDBMS
Name*	rdbmsdatasource
Description	RDBMS Data Source
Data Source Provider*	default
Driver*	com.mysql.jdbc.Driver
URL*	jdbc:mysql://localhost:3306/test
User Name	root
Password	*****
<input type="checkbox"/> Expose as a JNDI Data Source	
<input type="checkbox"/> Data Source Configuration Parameters	
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

External datasource provider

If you need to add a datasource supported by an external provider class such as `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`, select **External Data Source**, click **Add Property**, and then enter the name and value of each connection property you need to configure. Following is an example datasource for an external datasource provider:

## New Data Source

New Data Source

Data Source Type*	RDBMS												
Name*	rdbmsdatasource												
Description	RDBMS Data Source												
Data Source Provider*	External Data Source												
Data Source Class Name*	<code>ibc.jdbc2.optional.MysqlXADataSource</code>												
<input type="button" value="Add Property"/> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>url</td> <td><code>:mysql://localhost:3306/test</code></td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>user</td> <td>root</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>password</td> <td>root</td> <td><input type="button" value="Delete"/></td> </tr> </tbody> </table>		Name	Value	Action	url	<code>:mysql://localhost:3306/test</code>	<input type="button" value="Delete"/>	user	root	<input type="button" value="Delete"/>	password	root	<input type="button" value="Delete"/>
Name	Value	Action											
url	<code>:mysql://localhost:3306/test</code>	<input type="button" value="Delete"/>											
user	root	<input type="button" value="Delete"/>											
password	root	<input type="button" value="Delete"/>											
<input type="checkbox"/> Expose as a JNDI Data Source													
<input type="checkbox"/> Data Source Configuration Parameters													
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>													

### Configuring a JNDI Datasource

Java Naming and Directory Interface (JNDI) is a Java Application Programming Interface (API) that provides naming and directory functionality for Java software clients, to discover and look up data and objects via a name. It helps decoupling object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI look-up and use it.

When adding a datasource, to expose a RDBMS datasource as a JNDI datasource, click **Expose as a JNDI Data Source** to display the JNDI fields as follows:

## New Data Source

The screenshot shows the 'New Data Source' configuration dialog. At the top, 'Data Source Type' is set to 'RDBMS'. Below it, fields include 'Name', 'Description', 'Data Source Provider' (set to 'default'), 'Driver', 'URL', 'User Name' (set to 'admin'), and 'Password' (represented by five dots). A section titled 'Expose as a JNDI Data Source' is highlighted with a red border. It contains three fields: 'Name' (empty), 'Use Data Source Factory' (unchecked), and 'JNDI Properties' (with an 'Add Property' button). Below this section is another section titled 'Data Source Configuration Parameters'. At the bottom are 'Test Connection', 'Save', and 'Cancel' buttons.

Following are descriptions of the JNDI fields:

- **Name:** Name of the JNDI datasource that will be visible to others in object look-up.
- **Use Data Source Factory:** To make the datasource accessible from an external environment, you must use a datasource factory. When this option is selected, a reference object will be created with the defined datasource properties. The datasource factory will create the datasource instance based on the values of the reference object when accessing the datasource from an external environment. In the datasource configuration, this is set as: <jndiConfig useDataSourceFactory="true" >.
- **JNDI Properties:** Properties related to the JNDI datasource (such as password). When you select this option, set the following properties:
  - `java.naming.factory.initial`: Selects the registry service provider as the initial context.
  - `java.naming.provider.url`: Specifies the location of the registry when the registry is being used as the initial context.

### Configuring the Datasource Connection Pool Parameters

When the server processes a database operation, it spawns a database connection from an associated datasource. After using this connection, the server returns it to the pool of connections. This is called datasource connection pooling. It is a recommended way to gain more performance/throughput in the system. In datasource connection pooling, the physical connection is not dropped with the database server, unless it becomes stale or the datasource connection is closed.

RDBMS datasources in WSO2 products use Tomcat JDBC connection pool (`org.apache.tomcat.jdbc.pool`). It is common to all components that access databases for data persistence, such as the registry, user management (if configured against a JDBC userstore), etc.

You can configure the datasource connection pool parameters, such as how long a connection is persisted in the pool, using the datasource configuration parameters section that appears in the product management console when creating a datasource. Click and expand the option as shown below:

## Add New Data Source

New Data Source

DataSource Id*	oracle-ds
Data Source Type*	RDBMS
Database Engine*	Oracle
Driver Class*	oracle.jdbc.driver.OracleDriver
URL*	jdbc:oracle:[drivertype]:[username/password]@[host]:[port]/[database]
User Name	
Password	
<input type="checkbox"/> Data source configuration parameters	
Transaction Isolation	TRANSACTION_UNKNOWN
Initial Size	
Max. Active	
Max. Idle	
Min. Idle	
Max. Wait	
Validation Query	
Test On Return	false
Test On Borrow	true
Test While Idle	false
Time Between Eviction Runs Mills	
Minimum Evictable Idle Time	
Remove Abandoned	false
Remove Abandoned Timeout	
Log Abandoned	false
Default Auto Commit	false
Default Read Only	false
Default Catalog	
Validator Class Name	
Connection Properties	
Init SQL	
JDBC Interceptors	
Validation Interval	
JMX Enabled	false
Fair Queue	false
Abandon When Percentage Full	
Max Age	
Use Equals	false

The screenshot shows a configuration dialog for a JDBC connection pool. At the top, there are two tabs: "Suspect Timeout" and "Alternate User Name Allowed". Under "Suspect Timeout", a dropdown menu is set to "false". Below the tabs are three buttons: "Test Connection", "Save", and "Cancel".

Following are descriptions of the parameters you can configure. For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

Parameter name	Description
Transaction isolation	The default TransactionIsolation state of connections created by this pool are as follows: <ul style="list-style-type: none"> <li>• TRANSACTION_UNKNOWN</li> <li>• TRANSACTION_NONE</li> <li>• TRANSACTION_READ_COMMITTED</li> <li>• TRANSACTION_READ_UNCOMMITTED</li> <li>• TRANSACTION_REPEATABLE_READ</li> <li>• TRANSACTION_SERIALIZABLE</li> </ul>
Initial Size (int)	The initial number of connections created, when the pool is started. Default value is zero.
Max. Active (int)	Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.
Max. Idle (int)	Maximum number of connections that should be kept in the pool at all times. Default value is 8. Idle connections are checked periodically (if enabled), and connections that have been idle for longer than <code>minEvictableIdleTimeMillis</code> will be released. (also see <code>testWhileIdle</code> )
Min. Idle (int)	Minimum number of established connections that should be kept in the pool at all times. The connection pool can shrink below this number, if validation queries fail. Default value is zero. For more information, see <code>testWhileIdle</code> .
Max. Wait (int)	Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. Default value is 30000 (30 seconds).
Validation Query (String)	The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1 (mysql), select 1 from dual (oracle), SELECT 1 (MS Sql Server).
Test On Return (boolean)	Used to indicate if objects will be validated before returned to the pool. The default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <span style="color: #0070C0; font-weight: bold;">i</span> For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. </div>
Test On Borrow (boolean)	Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. Default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <span style="color: #0070C0; font-weight: bold;">i</span> For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code> . </div>

Test While Idle (boolean)	The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. The default value is false and this property has to be set in order for the pool cleaner/test thread to run. For more information, see <a href="#">timeBetweenEvictionRunsMillis</a> .
	<p> For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</p>
Time Between Eviction Runs Mills (int)	Number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. This value should not be set under 1 second. It indicates how often we check for idle, abandoned connections, and how often we validate idle connections. The default value is 5000 (5 seconds).
Minimum Evictable Idle Time (int)	Minimum amount of time an object may sit idle in the pool before it is eligible for eviction. The default value is 60000 (60 seconds).
Remove Abandoned (boolean)	Flag to remove abandoned connections if they exceed the <code>removeAbandonedTimeout</code> . If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <code>removeAbandonedTimeout</code> . Setting this to true can recover database connections from applications that fail to close a connection. For more information, see <a href="#">logAbandoned</a> . The default value is false.
Remove Abandoned Timeout (int)	Timeout in seconds before an abandoned (in use) connection can be removed. The default value is 60 (60 seconds). The value should be set to the longest running query that your applications might have.
Log Abandoned (boolean)	Flag to log stack traces for application code which abandoned a connection. Logging of abandoned connections, adds overhead for every connection borrowing, because a stack trace has to be generated. The default value is false.
Auto Commit (boolean)	The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the <code>setAutoCommit</code> method will not be called.
Default Read Only (boolean)	The default read-only state of connections created by this pool. If not set then the <code>setReadOnly</code> method will not be called. (Some drivers don't support read only mode. For example: Informix)
Default Catalog (String)	The default catalog of connections created by this pool.
Validator Class Name (String)	The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.Validator</code> interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a <code>Validator</code> instance, which is then used instead of any validation query to validate connections. The default value is null. An example value is <code>com.mycompany.project.SimpleValidator</code> .
Connection Properties (String)	Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]*</code> . The default value is null.
	<p> The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.</p>

Init SQL	Ability to run a SQL statement exactly once, when the connection is created.
JDBC Interceptors	Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.
Validation Interval (long)	To avoid excess validation, only run validation at most at this frequency - time in milliseconds. If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).
JMX Enabled (boolean)	Register the pool with JMX or not. The default value is true.
Fair Queue (boolean)	Set to true, if you wish that calls to <code>getConnection</code> should be treated fairly in a true FIFO fashion. This uses the <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue</code> implementation for the list of the idle connections. The default value is true. This flag is required when you want to use asynchronous connection retrieval. Setting this flag ensures that threads receive connections in the order they arrive. During performance tests, there is a very large difference in how locks and lock waiting is implemented. When <code>fairQueue=true</code> , there is a decision making process based on what operating system the system is running. If the system is running on Linux (property <code>os.name=Linux</code> ), then to disable this Linux specific behavior and still use the fair queue, simply add the property <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true</code> to your system properties, before the connection pool classes are loaded.
Abandon When Percentage Full (int)	Connections that have been abandoned (timed out) will not get closed and reported up, unless the number of connections in use are above the percentage defined by <code>abandonWhenPercentageFull</code> . The value should be between 0-100. The default value is zero, which implies that connections are eligible for closure as soon as <code>removeAbandonedTimeout</code> has been reached.
Max Age (long)	Time in milliseconds to keep this connection. When a connection is returned to the pool, the pool will check to see if the current time when connected, is greater than the <code>maxAge</code> that has been reached. If so, it closes the connection rather than returning it to the pool. The default value is zero, which implies that connections will be left open and no age check will be done upon returning the connection to the pool.
Use Equals (boolean)	Set to true, if you wish the <code>ProxyConnection</code> class to use <code>String.equals</code> , and set to false when you wish to use <code>==</code> when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.
Suspect Timeout (int)	Timeout value in seconds. Default value is zero. Similar to the <code>removeAbandonedTimeout</code> value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if <code>logAbandoned</code> is set to true. If this value is equal or less than zero, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than zero, and the connection was not abandoned, or if abandon check is disabled. If a connection is suspected, a warning message gets logged and a JMX notification will be sent.
Alternate User Name Allowed (boolean)	By default, the <code>jdbc-pool</code> will ignore the <code>DataSource.getConnection(username, password)</code> call, and simply return a previously pooled connection under the globally configured properties username and password, for performance reasons.  The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the <code>DataSource.getConnection(username, password)</code> call, simply set the property <code>alternateUsernameAllowed</code> , to true. If you request a connection with the credentials user1/password1, and the connection was previously connected using different user2/password2, then the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per-schema level. The default value is false.

## Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

Following are descriptions of the custom datasource fields:

- **Data Source Type:** Custom
- **Custom Data Source Type:** Specify whether the data is in a table or accessed through a query as described below.
- **Name:** Enter a unique name for this datasource
- **Description:** Description of the datasource
- **Configuration:** XML configuration of the datasource

### **Custom datasource type**

When creating a custom datasource, specify whether the datasource type is DS\_CUSTOM\_TABULAR (the data is stored in tables), or DS\_CUSTOM\_QUERY (non-tabular data accessed through a query). More information about each type are explained below.

#### Custom tabular datasources

Tabular datasources are used for accessing tabular data, that is, the data is stored in rows in named tables that can be queried later. To implement tabular datasources, the interface `org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS` is used. For more information, see a sample implementation of a tabular custom datasource at [InMemoryDataSource](#).

A tabular datasource is typically associated with a SQL data services query. WSO2 products use an internal SQL parser to execute SQL against the custom datasource. For more information, see a sample data service descriptor at [InMemoryDSSample](#). Carbon datasources also support tabular data with the `org.wso2.carbon.dataservices.core.custom.datasource.CustomTabularDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for more information see the `<PRODUCT_HOME>\repository\conf\datasources\custom-datasources.xml` file, which is a sample Carbon datasource configuration.

#### Custom query datasources

Custom query-based datasources are used for accessing non-tabular data through a query expression. To implement query-based datasources, the `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS` interface is used. You can create any non-tabular datasource using the query-based approach. Even if the target datasource does not have a query expression format, you can create and use your own. For example, you can support any NoSQL type datasource using this type of a datasource.

For more information, see a sample implementation of a custom query-based datasource at [EchoDataSource](#), and a sample data service descriptor with custom query datasources in [InMemoryDSSample](#). Carbon datasources

also support query-based data with the `org.wso2.carbon.dataservices.core.custom.datasource.Cust`  
`omQueryDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for  
more information, see the `<PRODUCT_HOME>\repository\conf\datasources\custom-datasources.xml` file, which is a sample Carbon datasource configuration.

In the `init` methods of all custom datasources, user-supplied properties will be parsed to initialize the datasource  
accordingly. Also, a property named `<__DATASOURCE_ID__>`, which contains a UUID to uniquely identify the  
current datasource, will be passed. This can be used by custom datasource authors to identify the datasources  
accordingly, such as datasource instances communicating within a server cluster for data synchronization.

Shown below is an example configuration of a custom datasource of type `<DS_CUSTOM_TABULAR>`:

```

<configuration>
    <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.ds
    <customDataSourceProps>
        <property name="inmemory_datasource_schema">{Vehicles:[ID,M
        <property name="inmemory_datasource_records">
            {Vehicles:[["S10_1678","Harley Davidson Ultimate Choppe
            ["S10_1949","Alpine Renault 1300","Classic Cars","1952"
            ["S10_2016","Moto Guzzi 1100i","Motorcycles","1996"],[
            ["S10_4698","Harley-Davidson Eagle Drag Bike","Motorcyc
            ["S10_4757","Alfa Romeo GTA","Classic Cars","1972"],
            ["S10_4962","Lancia A Delta 16V","Classic Cars","1962"],
            ["S12_1099","Ford Mustang","Classic Cars","1968"],
            ["S12_1108","Ferrari Enzo","Classic Cars","2001"]]}
        </property>
    </customDataSourceProps>
</configuration>

```

After creating datasources, they will appear on the **Data Sources** page. You can edit and delete them as needed by  
clicking **Edit** or **Delete** links.

## Managing Users and Roles

Before you begin, note the following:

- Only system administrators can add, modify and remove users and roles. To set up administrators, see [Real m Configuration](#).
- Your product has a primary user store where the users/roles that you create using the management console are stored by default. It's default RegEx configurations are as follows. RegEx configurations ensure that parameters like the length of a user name/password meet the requirements of the user store.

```

PasswordJavaRegEx----- ^[\S]{5,30}$
PasswordJavaScriptRegEx-- ^[\S]{5,30}$
UsernameJavaRegEx----- ^~!#$;%*+={}\\{3,30}$
UsernameJavaScriptRegEx-- ^[\S]{3,30}$
RolenameJavaRegEx----- ^~!#$;%*+={}\\{3,30}$
RolenameJavaScriptRegEx-- ^[\S]{3,30}$

```

When creating users/roles, if you enter a username, password etc. that does not conform to the RegEx config  
urations, the system throws an exception. You can either change the RegEx configuration or enter values that

conform to the RegEx. If you change the default user store or set up a secondary user store, configure the RegEx accordingly under the user store manager configurations in <APIM\_HOME>/repository/conf/user-mgt.xml file.

This chapter contains the following information:

- [Adding User Roles](#)
- [Adding Users](#)

## Adding User Roles

Roles contain permissions for users to manage the server. They can be reused and they eliminate the overhead of granting permissions to users individually.

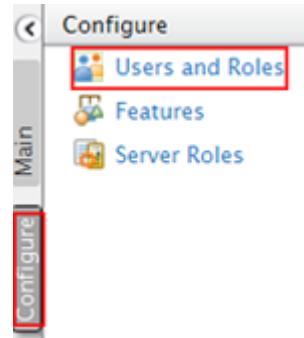
Throughout this documentation, we use the following roles that are typically used in many enterprises. You can also define different user roles depending on your requirements.

- **admin:** The API management provider who hosts and manages the [API Gateway](#). S/he is responsible for creating user roles in the system, assign them roles, managing databases, security etc. The Admin role is available by default with credentials admin/admin.
- **creator:** A creator is typically a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions etc.) and uses the [API publisher](#) to provision APIs into the API store. The creator uses the API Store to consult ratings and feedback provided by API users. Creator can add APIs to the store but cannot manage their lifecycle.
- **publisher:** A person in a managerial role and overlooks a set of APIs across the enterprise and controls the API lifecycle, subscriptions and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **subscriber:** A user or an application developer who searches the [API store](#) to discover APIs and use them. S/he reads the documentation and forums, rates/comments on the APIs, subscribes to APIs, obtains access tokens and invokes the APIs.

Follow the instructions below to create the creator, publisher and subscriber roles in the API Manager.

### Create user roles

1. Log in to the management console (<https://localhost:9443/carbon>) as admin (default credentials are admin/admin).



2. Select **Users and Roles** under the **Configure** menu.
3. In the **User Management** page that opens, click **Roles**.

User Management

System User Store

- Users
- Roles**

Change Password

Change My Password

4. Click **Add** **New Role**.

Name	Actions
admin	Assign Users  View Users
Internal/everyone	Permissions
Internal/subscriber	Rename  Permissions  Assign Users  View Users  Delete
<b>+ Add New Role</b>	
<b>+ Add New Internal Role</b>	

5. Enter the name of the user role (e.g., creator) and click **Next**.

### Step 1 : Enter role details

Enter role details

Domain	PRIMARY
Role Name*	creator
<input type="button" value="Next &gt;"/> <input type="button" value="Finish"/> <input type="button" value="Cancel"/>	

**Tip:** The **Domain** drop-down list contains all user stores configured in the system. By default, you only have the PRIMARY user store. To configure secondary user stores, see [Configuring Secondary User Stores](#).

6. The permissions page opens. Select the permissions according to the role that you create. The table below lists the permissions of the `creator`, `publisher` and `subscriber` roles:

Roles	Permissions	UI
creator	<ul style="list-style-type: none"> <li>Configure &gt; Governance and all underlying permissions.</li> <li>Login</li> <li>Manage &gt; API &gt; Create</li> <li>Manage &gt; Resources &gt; Govern and all underlying permissions</li> </ul>	

publisher	<ul style="list-style-type: none"> <li>• Login</li> <li>• Manage &gt; API &gt; Publish</li> </ul>	
subscriber	<ul style="list-style-type: none"> <li>• Login</li> <li>• Manage &gt; API &gt; Subscribe</li> </ul>	

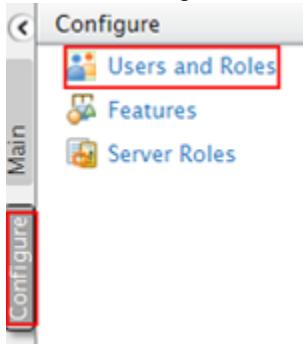
7. Click **Finish** once you are done adding permission.

### Adding Users

Users are consumers who interact with your enterprise's applications, databases or any other systems. These users can be persons, devices or applications/programs within or outside of the enterprise's network. Since these users interact with internal systems and access data, the need to define which user is allowed to do what is critical. This is called user management.

Follow the steps below to create users and assign them to roles via the admin console. Also, if you want to authenticate users via **e-mail**, **social media**, **multiple user store attributes**, see [Maintaining Logins and passwords](#).

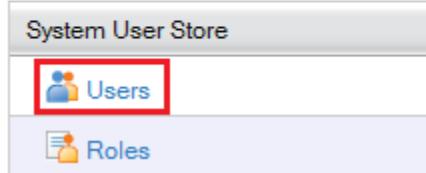
- o the Management Console and select **Users and Roles** from the **Configure** menu.



2. Click **Users** in the **User Management** window that opens.

[Home](#) > [Configure](#) > [Users and Roles](#)

### User Management



**⚠** The **Users** link is only visible to admins.

3. Click **Add** **New** **User**.

## Users

Search

Name	Actions
admin	Change Password  Assign Roles  View Roles User Profile

Add New User Bulk Import Users

4. The **Add User** page opens. Provide the username and password and click **Next**.

### Add User

#### Step 1 : Enter user name

Enter user name	
Domain	PRIMARY
User Name*	user1
Password*	*****
Password Repeat*	*****
<b>Next &gt;</b> <b>Finish</b> <b>Cancel</b>	

**Tip:** The **Domain** drop-down list contains all user stores configured in the system. By default, you only have the PRIMARY user store. To configure secondary user stores, see [Configuring Secondary User Stores](#).

5. Select the roles you want to assign to the user. In this example, we assign the `creator` role defined in the previous section.

## Add User

**Step 2 : Select roles of the user**

Enter role name pattern (\* for all)

Users of Role	
<a href="#">Select all on this page</a>   <a href="#">Unselect all on this page</a>	
<input type="checkbox"/> admin	
<input checked="" type="checkbox"/> creator	
<input checked="" type="checkbox"/> Internal/everyone	
<input type="checkbox"/> Internal/subscriber	

6. Click **Finish** to complete. The new user appears in the **Users** list.

## Users

Enter user name pattern (\* for all)

Name	Actions
admin	Change Password  Roles
apicreator	Change Password  Roles  Delete

From here, you can change the user's password, assign different roles or delete it.

You cannot change the user name of an existing user.

## Configuring User Stores

A user store is the database where information of the users and/or user roles is stored. User information includes log-in name, password, first name, last name, e-mail etc.

All WSO2 products have an embedded H2 database except for WSO2 Identity Server, which has an embedded LDAP as its user store. Permission is stored in a separate database called the user management database, which by default is H2. However, users have the ability to connect to external user stores as well.

The user stores of Carbon products can be configured to operate in either one of the following modes.

- User store operates in read/write mode - In Read/Write mode, WSO2 Carbon reads/writes into the user store.
- User store operates in read only mode - In Read Only mode, WSO2 Carbon guarantees that it does not modify any data in the user store. Carbon maintains roles and permissions in the Carbon database but it can read users/roles from the configured user store.

The sections below provide configuration details:

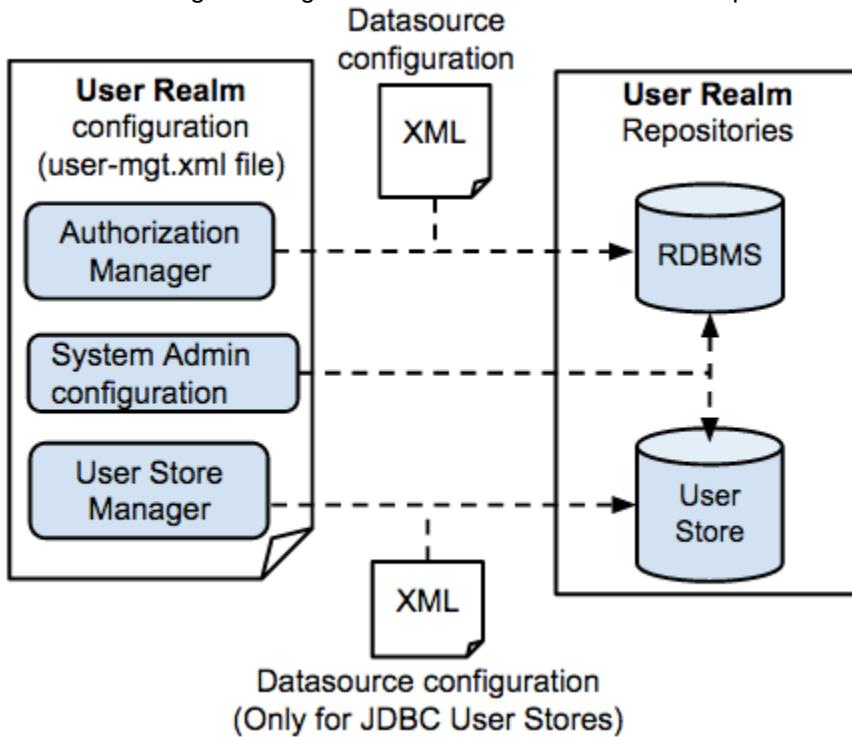
- [Realm Configuration](#)
- [Changing the RDBMS](#)
- [Configuring Primary User Stores](#)

- Configuring Secondary User Stores

## Realm Configuration

The complete functionality and contents of the User Management module is called a **user realm**. The realm includes the user management classes, configurations and repositories that store information. Therefore, configuring the User Management functionality in a WSO2 product involves setting up the relevant repositories and updating the relevant configuration files.

The following diagram illustrates the required configurations and repositories:



See the following topics for more details:

- Configuring the system administrator
- Configuring the authorization manager

### Configuring the system administrator

The **admin** user is the super tenant that will be able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that should have admin permissions is required to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores will explain how to configure the administrator while configuring the user store. The information under this topic will explain the main configurations that are relevant to setting up the system administrator.

**!** If the primary user store is read-only, you will be using a user ID and role that already exists in the user store, for the administrator. If the user store is read/write, you have the option of creating the administrator user in the user store as explained below. By default, the embedded H2 database (with read/write enabled) is used for both these purposes in WSO2 products.

Note the following key facts about the system administrator in your system:

- The admin user and role is always stored in the primary user store in your system.
- An administrator is configured for your system by default. This **admin** user is assigned to the **admin** role, which has all permissions enabled.

- The permissions assigned to the default **admin** role cannot be modified.

### **Updating the administrator**

The `<Configuration>` section at the top of the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file allows you to configure the administrator user in your system as well as the RDBMS that will be used for storing information related to user authentication (i.e. role-based permissions).

```

<Realm>
    <Configuration>
        <AddAdmin>true</AddAdmin>
        <AdminRole>admin</AdminRole>
        <AdminUser>
            <UserName>admin</UserName>
            <Password>admin</Password>
        </AdminUser>
        <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role
see the registry root -->
        <Property name=""></Property>
        .....
    </Configuration>
    ...
</Realm>

```

Note the following regarding the configuration above.

Element	Description
<AddAdmin>	When <code>true</code> , this element creates the admin user based on the <code>AdminUser</code> element. It also indicates whether to create the specified admin user if it doesn't already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be <code>false</code> if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be <code>true</code> , so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the <code>AddAdmin</code> value is set to <code>true</code> in this case, it will generate an exception.
<AdminRole>wso2admin</AdminRole>	This is the role that has all administrative privileges of the WSO2 product, so all users having this role are admins of the product. You can provide any meaningful name for this role. This role is created in the internal H2 database when the product starts. This role has permission to carry out any actions related to the Management Console. If the user store is read-only, this role is added to the system as a special internal role where users are from an external user store.
<AdminUser>	Configures the default administrator for the WSO2 product. If the user store is read-only, the admin user must exist in the user store or the system will not start. If the external user store is read-only, you must select a user already existing in the external user store and add it as the admin user that is defined in the <code>&lt;AdminUser&gt;</code> element. If the external user store is in read/write mode, and you set <code>&lt;AddAdmin&gt;</code> to <code>true</code> , the user you specify will be automatically created.

<UserName>	This is the username of the default administrator or super tenant of the user store. If the user store is read-only, the admin user MUST exist in the user store for the process to work.
<Password>	Do NOT put the password here but leave the default value. If the user store is read-only, this element and its value are ignored. This password is used only if the user store is read-write and the <code>AddAdmin</code> value is set to true.   Note that the password in the <code>user-mgt.xml</code> file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the <code>user-mgt.xml</code> file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the <code>user-mgt.xml</code> file. To change the admin password, you must use the <b>Change Password</b> option from the management console.
<EveryOneRoleName>	The name of the "everyone" role. All users in the system belong to this role.

## Configuring the authorization manager

According to the default configuration in WSO2 products, the Users, Roles and Permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the Users and Roles are stored in one repository (User Store) and the Permissions are stored in a separate repository. A user store can be a typical RDBMS, an LDAP or an external Active Directory.

The repository that stores Permissions should always be an RDBMS. The Authorization Manager configuration in the `user-mgt.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/` directory) connects the system to this RDBMS.

Follow the steps given below to set up and configure the Authorization Manager.

### Step 1: Setting up the repository

By default, the embedded H2 database is used for storing permissions. You can change this as follows:

1. Change the default H2 database or set up another RDBMS for storing permissions.
2. When you set up an RDBMS for your system, it is necessary to create a corresponding datasource, which allows the system to connect to the database.
  - If you are replacing the default H2 database with a new RDBMS, update the `master-datasource.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/datasources/` directory) with the relevant information.
  - Alternatively, create a new XML file with the datasource information of your new RDBMS and store it in the same `<PRODUCT_HOME>/repository/conf/datasources/` directory.

Refer the [related topics](#) for detailed information on setting up databases and configuring datasources.

### Step 2: Updating the user realm configurations

Once you have set up a new RDBMS and configured the datasource, the `user-mgt.xml` file (user realm configuration) should be updated as explained below.

1. Set up the database connection by update the datasource information using the `<Property>` element under `<Configuration>`. The jndi name of the datasource should be used to refer to the datasource. In the

following example, the jndi name of the default datasource defined in the <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml file is linked from the user-mgt.xml file.

```
<Realm>
  <Configuration>
    .....
    <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
  </Configuration>
  ...
</Realm>
```

You can add more configurations using the <Property> element:

Property Name	Description
testOnBorrow	It is recommended to set this property to 'true' so that object connections will be validated before being borrowed from the JDBC pool. For this property to be effective, the validationQuery parameter in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file should be a non-string value. This setting will avoid connection failures. See the section on performance tuning of WSO2 products for more information.

2. The default Authorization Manager section in the user-mgt.xml file is shown below. This can be updated accordingly.

```
<AuthorizationManager
class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
  <Property name="AdminRoleManagementPermissions">/permission</Property>
  <Property name="AuthorizationCacheEnabled">true</Property>
</AuthorizationManager>
```

- The org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager class enables the Authorization Manager for your product.
- The AdminRoleManagementPermissions property sets the registry path where the authorization information (role-based permissions) are stored. Note that this links to the repository that you defined in Step 1.
- It is recommended to enable the GetAllRolesOfUserEnabled property in the AuthorizationManager as follows:

```
<Property name="GetAllRolesOfUserEnabled">true</Property>
```

Although using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permission stats.

- By default, the rules linked to a permission (role name, action, resource) are not case sensitive. If you want to make them case sensitive, enable the following property:

```
<Property name="CaseSensitiveAuthorizationRules">true</Property>
```

## Related topics

1. See [Maintaining Logins and passwords](#) for information on how to change the super admin credentials.

## Changing the RDBMS

The default database of user manager is the H2 database that comes with WSO2 products. You can configure it to point to databases by other vendors.

1. Add the JDBC driver to the classpath by dropping the JAR into <PRODUCT\_HOME>/repository/components/lib.
2. Change values of properties given in on the [Realm Configuration](#) page appropriately.
3. Create the database by running the relevant script in <PRODUCT\_HOME>/dbscript and restart the server:
  - **For Linux:** sh wso2server.sh or sh wso2server.sh
  - **For Windows:** wso2server.bat or wso2server.bat

## Configuring Primary User Stores

Every WSO2 product comes with an embedded, internal user store, which is configured in <PRODUCT\_HOME>/repository/conf/user-mgt.xml. In WSO2 Identity Server, the embedded user store is LDAP, and in other products it is JDBC. Because the domain name (unique identifier) of this default user store is set to PRIMARY by default, it is called the primary user store.

Instead of using the embedded user store, you can set your own user store as the primary user store. Since the user store you want to connect to might have different schemas from the ones available in the embedded user store, it needs to go through an adaptation process. WSO2 products provide the following adapters to enable you to authenticate users from different types of user stores and plug into LDAP, Active Directory, and JDBC to perform authentication:

User store manager class	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAPUsers external LDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUsers both read and write operations uncommented in the code in
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUser Domain Service (AD DS) or LDS). This can be used <b>only</b> read-only you must use org APUserStoreManager.
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager stores.

The user-mgt.xml file already has sample configurations for all of the above user stores. To enable these configurations, you must uncomment them in the code and comment out the ones that you do not need.

The following topics provide details on the various primary user stores you can configure.

- [Configuring an external LDAP or Active Directory user store](#)
- [Configuring an internal/external JDBC user store](#)



If you are using ldaps (secured) to connect to the Active Directory as shown below, you need to import the certificate of Active Directory to the client-truststore.jks of the WSO2 product. See the topic on configuring keystores for information on how to add certificates to the trust-store.

```
<Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
```

## Configuring an external LDAP or Active Directory user store

All WSO2 products can read and write users and roles from external Active Directory or LDAP user stores. You can configure WSO2 products to access these user stores in one of the following modes:

- Read-only mode
- Read/write mode

### **Read-only mode**



## Before you begin

- If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.
- The class attribute for a read-only LDAP is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager" >`

When you configure a product to read users/roles from your company LDAP in read-only mode, it does not write any data into the LDAP.

1. Comment out the following user store which is enabled by default in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.  
`<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager" >`
2. Given below is a sample for the LDAP user store. This configuration is found in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file, however, you need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.

```

<UserManager>
    <Realm>
        ...
        <UserStoreManager
            class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager" >
                <Property
                    name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
                <Property name="ReadOnly">true</Property>
                <Property name="Disabled">false</Property>
                <Property name="MaxUserNameListLength">100</Property>
                <Property name="ConnectionURL">ldap://localhost:10389</Property>
                <Property name="ConnectionName">uid=admin,ou=system</Property>
                <Property name="ConnectionPassword">admin</Property>
                <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
                <Property name="UserSearchBase">ou=system</Property>
                <Property name="UserNameListFilter">(objectClass=person)</Property>
                <Property
                    name="UserNameSearchFilter">(& objectClass=person)(uid=?))</Property>
                <Property name="UserNameAttribute">uid</Property>
                <Property name="ReadGroups">true</Property>
                <Property name="GroupSearchBase">ou=system</Property>
                <Property
                    name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
                <Property
                    name="GroupNameSearchFilter">(& objectClass=groupOfNames)(cn=?))</Property>
                <Property name="GroupNameAttribute">cn</Property>
                <Property name="SharedGroupNameAttribute">cn</Property>
                <Property
                    name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
                <Property
                    name="SharedGroupNameListFilter">(objectClass=groupOfNames)</Property>
                <Property
                    name="SharedTenantNameListFilter">(objectClass=organizationalUnit)</Property>
                <Property name="SharedTenantNameAttribute">ou</Property>
                <Property
                    name="SharedTenantObjectClass">organizationalUnit</Property>
                <Property name="MembershipAttribute">member</Property>
                <Property name="UserRolesCacheEnabled">true</Property>
                <Property name="ReplaceEscapeCharactersAtUserLogin">true</Property>
                <Property name="MaxRoleNameListLength">100</Property>
                <Property name="MaxUserNameListLength">100</Property>
                <Property name="SCIMEnabled">false</Property>
            </UserStoreManager>
        </Realm>
    </UserManager>

```

- a. Update the connection details to match your user store. For example:

```
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

- b. Obtain a user who has permission to read all users/attributes and perform searches on the user store from your LDAP/Active Directory administrator. For example, if the privileged user is "AdminLDAP" and the password is "2010#Avrudu", update the following sections of the realm configuration as follows:

```
<Property name="ConnectionName">uid=AdminLDAP,ou=system</Property>
<Property name="ConnectionPassword">2010#Avrudu</Property>
```

- c. Update `<Property name="UserSearchBase" />` with the directory name where the users are stored. When LDAP searches for users, it will start from this location of the directory.

```
<Property name="UserSearchBase">ou=system</Property>
```

- d. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute" />` and `<Property name="UserNameSearchFilter" />` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

```
<Property name="UserNameAttribute">uid</Property>
```

- e. Set the `ReadGroups` property to 'true', if it should be allowed to read roles from this user store. When this property is 'true', you must also specify values for the `GroupSearchBase`, `GroupSearchFilter` and `GroupNameAttribute` properties as shown in the following example:

```
<Property name="ReadGroups">true</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
```

If the `ReadGroups` property is set to 'false', only Users can be read from the user store.

- f. Optionally, configure the realm to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute. The following code snippet represents reading roles based on a membership attribute. This is used by the ApacheDirectory server and OpenLDAP

```
<Property name="ReadLDAPGroups">false</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MembershipAttribute">member</Property>
```

- g. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

- Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

#### **Read/write mode**



## Before you begin

- To read and write to an Active Directory user store, set the `WriteGroups` property to `true` instead of `false`.
- To write user entries to an LDAP user store (roles are not written, just user entries), you follow the steps in the [Read-only mode](#) section but specify the following class instead:

```
<UserStoreManager  
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```

- Use the following class for Active Directory.

```
<UserStoreManager  
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
```

The `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file has commented-out configurations for external LDAP/AD user stores.

- Enable the `<ReadWriteLDAPUserStoreManager>` or the `<ActiveDirectoryUserStoreManager>` in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file by uncommenting the code. When it is enabled, the user manager reads/writes into the LDAP/AD user store. Note that these configurations already exist in the `user-mgt.xml` file so you only need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.
- The default configuration for the external read/write user store in the `user-mgt.xml` file is as follows. Change the values according to your requirements.

---

LDAP User Store

LDAP user store sample:

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
    <Property
name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
    <Property name="ConnectionName">uid=admin,ou=system</Property>
    <Property name="ConnectionPassword">admin</Property>
    <Property name="PasswordHashMethod">SHA</Property>
    <Property name="UserNameListFilter">(objectClass=person)</Property>
    <Property name="UserEntryObjectClass">wso2Person</Property>
    <Property name="UserSearchBase">ou=Users,dc=wso2,dc=org</Property>
    <Property
name="UserNameSearchFilter">(&& (objectClass=person) (uid=?))</Property>
    <Property name="UserNameAttribute">uid</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="UsernameJavaRegEx">^[^~!@#$%^*+={}\\|\\\\\\&lt;,&gt;,\\\"]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^[^~!@#$%^*+={}\\|\\\\\\&lt;,&gt;,\\\"]{3,30}$</Property>
    <Property name="ReadLDAPGroups">true</Property>
    <Property name="WriteLDAPGroups">true</Property>
    <Property name="EmptyRolesAllowed">true</Property>
    <Property name="GroupSearchBase">ou=Groups,dc=wso2,dc=org</Property>
    <Property name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
    <Property name="GroupEntryObjectClass">groupOfNames</Property>
    <Property
name="GroupNameSearchFilter">(&& (objectClass=groupOfNames) (cn=?))</Property>
    <Property name="GroupNameAttribute">cn</Property>
    <Property name="SharedGroupNameAttribute">cn</Property>
    <Property
name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
    <Property name="SharedGroupEntryObjectClass">groups</Property>
    <Property
name="SharedTenantNameListFilter">(object=organizationalUnit)</Property>
    <Property name="SharedTenantNameAttribute">ou</Property>
    <Property name="SharedTenantObjectClass">organizationalUnit</Property>
    <Property name="MembershipAttribute">member</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property>
</UserStoreManager>

```

 **Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up- `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

Active directory user store sample:

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
        <Property name="defaultRealmName">WSO2.ORG</Property>
        <Property name="Disabled">false</Property>

        <Property name="kdcEnabled">false</Property>
        <Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
        <Property
name="ConnectionName">CN=admin,CN=Users,DC=WSO2,DC=Com</Property>
            <Property name="ConnectionPassword">A1b2c3d4</Property>
        <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
            <Property name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
            <Property name="UserEntryObjectClass">user</Property>
            <Property name="UserNameAttribute">cn</Property>
            <Property name="isADLDSRole">false</Property>
        <Property name="userAccountControl">512</Property>
            <Property name="UserNameListFilter">(objectClass=user)</Property>
        <Property
name="UserNameSearchFilter">(&& (objectClass=user) (cn=?))</Property>
            <Property
name="UsernameJavaRegEx">[a-zA-Z0-9._-|//]{3,30}\$</Property>
                <Property name="UsernameJavaScriptRegEx">^\$\{3,30\}\$</Property>
                <Property name="PasswordJavaScriptRegEx">^\$\{5,30\}\$</Property>
            <Property name="RolenameJavaScriptRegEx">^\$\{3,30\}\$</Property>
                <Property
name="RolenameJavaRegEx">[a-zA-Z0-9._-|//]{3,30}\$</Property>
                <Property name="ReadGroups">true</Property>
                <Property name="WriteGroups">true</Property>
                <Property name="EmptyRolesAllowed">true</Property>
                    <Property name="GroupSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
                <Property name="GroupEntryObjectClass">group</Property>
                    <Property name="GroupNameAttribute">cn</Property>
                    <Property name="SharedGroupNameAttribute">cn</Property>
                    <Property
name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
                    <Property name="SharedGroupEntryObjectClass">groups</Property>
                    <Property
name="SharedTenantNameListFilter">(object=organizationalUnit)</Property>
                    <Property name="SharedTenantNameAttribute">ou</Property>
                    <Property
name="SharedTenantObjectClass">organizationalUnit</Property>
                    <Property name="MembershipAttribute">member</Property>
                    <Property
name="GroupNameListFilter">(objectcategory=group)</Property>
                    <Property
name="GroupNameSearchFilter">(&& (objectClass=group) (cn=?))</Property>
                        <Property name="UserRolesCacheEnabled">true</Property>
                        <Property name="Referral">follow</Property>
                    <Property name="BackLinksEnabled">true</Property>
                        <Property name="MaxRoleNameListLength">100</Property>
                        <Property name="MaxUserNameListLength">100</Property>
                        <Property name="SCIMEnabled">false</Property>
                </UserStoreManager>

```

 **Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up- `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

 When working with Active Directory it is best to enable the  `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows.

```
<AuthorizationManager
class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
<Property name="AdminRoleManagementPermissions">/permission</Property>
<Property name="AuthorizationCacheEnabled">true</Property>
<Property name="GetAllRolesOfUserEnabled">true</Property>
</AuthorizationManager>
```

While using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permissions stats.

 If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.

The `class` attribute of the `UserStoreManager` element indicates whether it is an Active Directory or LDAP user store:

- Active Directory: `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager" />`
- Read-only LDAP: `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager" />`

3. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

LDAP Active Directory

```
<Property name="UserNameAttribute">uid</Property>
```

```
<Property name="UserNameAttribute">sAMAccountName</Property>
```

4. The following code snippet represents reading roles based on a backlink attribute. This is used by the Active Directory.

```
<Property name="ReadLDAPGroups">true</Property>
<Property name="GroupSearchBase">cn=users,dc=wso2,dc=lk</Property>
<Property name="GroupSearchFilter">(objectcategory=group)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MemberOfAttribute">memberOf</Property>
```

5. For Active Directory, you can use <Property name="Referral">follow</Property> to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the user store configurations in the user-mgt.xml file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the <Property name="Referral">follow</Property> property ensures that all the domains in the directory will be searched to locate the requested object.
6. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

**i** When configuring an external LDAP for Governance Registry or API Manager, the user name and password for the default admin will change to the LDAP admin. As a result, the <PRODUCT\_HOME>/repository/conf/api-manager.xml file must be updated with the new LDAP admin credentials.

## Configuring an internal/external JDBC user store

The default internal JDBC user store reads/writes into the internal database of the Carbon server. JDBC user stores can be configured using the <PRODUCT\_HOME>/repository/conf/user-mgt.xml file's JDBCUserStoreManager configuration section. Additionally, all Carbon-based products can work with an external RDBMS. You can configure Carbon to read users/roles from your company RDBMS and even write to it. Therefore, in this scenario, the user core connects to two databases:

- The Carbon database where authorization information is stored internally.
- Your company database where users/roles reside.

Therefore, the user-mgt.xml file must contain details for two database connections. The connection details mentioned earlier are used by the authorization manager. If we specify another set of database connection details inside the UserStoreManager, it reads/writes users to that database. The following are step-by-step guidelines for connecting to an internal and external JDBC user store in read-only mode:

1. Uncomment the following section in <PRODUCT\_HOME>/repository/conf/user-mgmt.xml:

```
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
```

The following are samples for the internal and external JDBC user store configuration:

[Internal JDBC User Store](#)[External JDBC User Store](#)

Internal JDBC user store configuration sample:

```

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="ReadOnly">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="IsEmailUserName">false</Property>
    <Property name="DomainCalculation">default</Property>
        <Property name="PasswordDigest">SHA-256</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordJavaRegEx">[\S]{5,30}\$</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property
name="UsernameJavaRegEx">^\~!#%;%^*+={}\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^\~!@#%;%^*+={}\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>

```

#### External JDBC user store configuration sample:

```

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="driverName">com.mysql.jdbc.Driver</Property>
    <Property name="url">jdbc:mysql://localhost:3306/tcsdev</Property>
    <Property name="userName">shavantha</Property>
    <Property name="password">welcome</Property>
    <Property name="Disabled">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="PasswordDigest">SHA-256</Property>
    <Property name="ReadGroups">true</Property>
    <Property name="ReadOnly">false</Property>
    <Property name="IsEmailUserName">false</Property>
    <Property name="DomainCalculation">default</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="WriteGroups">false</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordJavaRegEx">^\S{5,30}\$</Property>
    <Property name="PasswordJavaScriptRegEx">^\S{5,30}\$</Property>
    <Property name="UsernameJavaRegEx">^\S{5,30}\$</Property>
    <Property name="UsernameJavaScriptRegEx">^\S{5,30}\$</Property>
    <Property name="RolenameJavaRegEx">^\S{5,30}\$</Property>
    <Property name="RolenameJavaScriptRegEx">^\S{5,30}\$</Property>
    <Property name="SCIMEnabled">false</Property>
    <Property name="SelectUserSQL">SELECT * FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name="GetRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_TENANT_ID=? AND

```

```

UM_SHARED_ROLE = '0' ORDER BY UM_ROLE_NAME</Property>
    <Property name="GetSharedRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID,
UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_SHARED_ROLE = '1'
ORDER BY UM_ROLE_NAME</Property>
    <Property name="UserFilterSQL">SELECT UM_USER_NAME FROM UM_USER WHERE
UM_USER_NAME LIKE ? AND UM_TENANT_ID=? ORDER BY UM_USER_NAME</Property>
    <Property name="UserRoleSQL">SELECT UM_ROLE_NAME FROM UM_USER_ROLE,
UM_ROLE, UM_USER WHERE UM_USER.UM_USER_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID
AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="UserSharedRolesSQL">SELECT UM_ROLE_NAME,
UM_ROLE.UM_TENANT_ID, UM_SHARED_ROLE FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER
ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON
UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_USER.UM_USER_NAME = ? AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>
    <Property name="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name="GetUserListOfRoleSQL">SELECT UM_USER_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_ROLE.UM_ROLE_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID
AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME FROM
UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID =
UM_USER.UM_ID INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID =
UM_ROLE.UM_ID WHERE UM_ROLE.UM_ROLE_NAME= ? AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID</Property>
    <Property name="IsUserExistingSQL">SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name=" GetUserPropertiesForProfileSQL">SELECT UM_ATTR_NAME,
UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID =
UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
    <Property name=" GetUserPropertyForProfileSQL">SELECT UM_ATTR_VALUE FROM
UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID = UM_USER_ATTRIBUTE.UM_USER_ID AND
UM_USER.UM_USER_NAME=? AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
    <Property name=" GetUserLisForPropertySQL">SELECT UM_USER_NAME FROM UM_USER,
UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID = UM_USER.UM_ID AND
UM_USER_ATTRIBUTE.UM_ATTR_NAME =? AND UM_USER_ATTRIBUTE.UM_ATTR_VALUE =? AND
UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="GetProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM
UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?</Property>
    <Property name=" GetUserProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM
UM_USER_ATTRIBUTE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=? ) AND UM_TENANT_ID=?</Property>
    <Property name="GetUserIDFromUserNameSQL">SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name="GetUserNameFromTenantIDSQl">SELECT UM_USER_NAME FROM
UM_USER WHERE UM_TENANT_ID=?</Property>
    <Property name="GetTenantIDFromUserNameSQL">SELECT UM_TENANT_ID FROM
UM_USER WHERE UM_USER_NAME=?</Property>
    <Property name="AddUserSQL">INSERT INTO UM_USER (UM_USER_NAME,
UM_USER_PASSWORD, UM_SALT_VALUE, UM_REQUIRE_CHANGE, UM_CHANGED_TIME,
```

```

UM_TENANT_ID) VALUES (?, ?, ?, ?, ?, ?)</Property>
    <Property name="AddUserToRoleSQL">INSERT INTO UM_USER_ROLE (UM_USER_ID,
UM_ROLE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=?
AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND
UM_TENANT_ID=?), ?)</Property>
    <Property name="AddRoleSQL">INSERT INTO UM_ROLE (UM_ROLE_NAME,
UM_TENANT_ID) VALUES (?, ?)</Property>
    <Property name="AddSharedRoleSQL">UPDATE UM_ROLE SET UM_SHARED_ROLE = ?
WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID = ?</Property>
    <Property name="AddRoleToUserSQL">INSERT INTO UM_USER_ROLE (UM_ROLE_ID,
UM_USER_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=?
AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND
UM_TENANT_ID=?), ?)</Property>
    <Property name="AddSharedRoleToUserSQL">INSERT INTO UM_SHARED_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_USER_TENANT_ID, UM_ROLE_TENANT_ID) VALUES ((SELECT
UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM
UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?)</Property>
    <Property name="RemoveUserFromSharedRoleSQL">DELETE FROM
UM_SHARED_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM UM_USER
WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_USER_TENANT_ID=? AND
UM_ROLE_TENANT_ID = ?</Property>
    <Property name="RemoveUserFromRoleSQL">DELETE FROM UM_USER_ROLE WHERE
UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?)
AND UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND
UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
    <Property name="RemoveRoleFromUserSQL">DELETE FROM UM_USER_ROLE WHERE
UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?)
AND UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND
UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
    <Property name="DeleteRoleSQL">DELETE FROM UM_ROLE WHERE UM_ROLE_NAME = ?
AND UM_TENANT_ID=?</Property>
    <Property name="OnDeleteRoleRemoveUserRoleMappingSQL">DELETE FROM
UM_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND
UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
    <Property name="DeleteUserSQL">DELETE FROM UM_USER WHERE UM_USER_NAME = ?
AND UM_TENANT_ID=?</Property>
    <Property name="OnDeleteUserRemoveUserRoleMappingSQL">DELETE FROM
UM_USER_ROLE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND
UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
    <Property name="OnDeleteUserRemoveUserAttributeSQL">DELETE FROM
UM_USER_ATTRIBUTE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
    <Property name="UpdateUserPasswordSQL">UPDATE UM_USER SET UM_USER_PASSWORD=?
, UM_SALT_VALUE=?, UM_REQUIRE_CHANGE=?, UM_CHANGED_TIME=? WHERE UM_USER_NAME= ?
AND UM_TENANT_ID=?</Property>
    <Property name="UpdateRoleNameSQL">UPDATE UM_ROLE set UM_ROLE_NAME=? WHERE
UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>
    <Property name="AddUserPropertySQL">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) VALUES
((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?, ?, ?
)</Property>
    <Property name="UpdateUserPropertySQL">UPDATE UM_USER_ATTRIBUTE SET
UM_ATTR_VALUE=? WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=?
AND UM_TENANT_ID=?) AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_TENANT_ID=?</Property>
    <Property name="DeleteUserPropertySQL">DELETE FROM UM_USER_ATTRIBUTE WHERE
UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?)
AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND UM_TENANT_ID=?</Property>

```

```

<Property name="UserNameUniqueAcrossTenantsSQL">SELECT UM_ID FROM UM_USER
WHERE UM_USER_NAME=?</Property>
<Property name="IsDomainExistingSQL">SELECT UM_DOMAIN_ID FROM UM_DOMAIN
WHERE UM_DOMAIN_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="AddDomainSQL">INSERT INTO UM_DOMAIN (UM_DOMAIN_NAME,
UM_TENANT_ID) VALUES (?, ?)</Property>
<Property name="AddUserToRoleSQL-mssql">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?), (?)</Property>
<Property name="AddRoleToUserSQL-mssql">INSERT INTO UM_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), (?)</Property>
<Property name="AddUserPropertySQL-mssql">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT
(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (?), (?),
(?), (?)</Property>
<Property name="AddUserToRoleSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT UU.UM_ID, UR.UM_ID, ? FROM UM_USER
UU, UM_ROLE UR WHERE UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=? AND
UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=?</Property>
<Property name="AddRoleToUserSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT UR.UM_ID, UU.UM_ID, ? FROM UM_ROLE
UR, UM_USER UU WHERE UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=? AND
UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=?</Property>
<Property name="AddUserPropertySQL-openedge">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT
UM_ID, ?, ?, ?, ? FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
```

```
<Property name="DomainName">wso2.org</Property>
<Property name="Description"/>
</UserStoreManager>
```

**i** The sample for the external JDBC user store consists of properties pertaining to various SQL statements. This is because the schema may be different for an external user store, and these adjustments need to be made in order to streamline the configurations with WSO2 products.

**i** You can define a data source in <PRODUCT\_HOME>/repository/conf/datasources/master-datasources.xml and refer to it from the user-mgt.xml file. This takes the properties defined in the master-datasources.xml file and reuses them in the user-mgt.xml file. To do this, you need to define the following property:

```
<Property name = "dataSource">jdbc/WSO2CarbonDB</Property>
```

2. Find a valid user that resides in the RDBMS. For example, say a valid username is AdminSOA. Update the Admin user section of your configuration as follows. You do not have to update the password element; leave it as is.

```
<AdminUser>
    <UserName>AdminSOA</UserName>
    <Password>XXXXXX</Password>
</AdminUser>
```

3. Add the PasswordHashMethod property to the UserStoreManager configuration for JDBCUserStoreManager. For example:

```
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="PasswordHashMethod">SHA</Property>
    ...
</UserStoreManager>
```

The PasswordHashMethod property specifies how the password should be stored. It usually has the following values:

- SHA - Uses SHA digest method.
- MD5 - Uses MD 5 digest method.
- PLAIN\_TEXT - Plain text passwords.

In addition, it also supports all digest methods in <http://docs.oracle.com/javase/6/docs/api/java/security/Mess ageDigest.html>.

4. Update the connection details found within the <UserStoreManager> class based on your preferences.
5. In the realm configuration section, set the value of the MultiTenantRealmConfigBuilder property to org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder. For example:

```
<Property
    name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
```

6. Add the JDBC driver to the classpath by copying its JAR file into the <PRODUCT\_HOME>/repository/components/lib directory.
7. Edit the SQLs in the user-mgt.xml file according to your requirements, and then start the server.

### Related Links

[Properties of Primary User Stores](#) - For a comprehensive understanding on the configuration details.

#### Properties of Primary User Stores

The following table provides descriptions of the key properties you use to configure primary user stores.

Property name	Description
MaxUserNameListLength	Controls the number of users listed in the user store of a WSO2 product. If you want to list them all, Setting this property to 0 displays all users.
ConnectionURL	Connection URL to the user store server. In the case of default port, reference to that port is included in this configuration.
ConnectionName	The username used to connect to the database and perform various operations on the user store or have an administrator role in the WSO2 product that you can use to view users' attributes and to perform search operations on the user store. This attribute is mandatory.
ConnectionPassword	Password for the ConnectionName user.
PasswordHashMethod	Password hash method to use when storing user entries in the user store.
UserNameListFilter	Filtering criteria for listing all the user entries in the user store. This property is optional. In case, the search operation only provides the objects created from the user store in the management console.
UserEntryObjectClass	Object class used to construct user entries. By default, it is a custom class.
UserSearchBase	DN of the context or object under which the user entries are stored. When performing user store searches for users, it will start from this location of the directory tree.
	<span style="border: 1px solid #ccc; padding: 2px;">(i) Different databases have different search bases.</span>
UserNameSearchFilter	Filtering criteria used to search for a particular user entry.
UserNameAttribute	The attribute used for uniquely identifying a user entry. Users can be identified by this attribute.
	<span style="border: 1px solid #ccc; padding: 2px;">(i) The name of the attribute is considered as the username.</span>
UsernameWithEmailJavaScriptRegEx	This property defines the JavaScript regular expression pattern which is used to validate the user name in the configuration file. If you need to support both email as a user name and user name, you can define two regular expressions.
	<span style="border: 1px solid #ccc; padding: 2px;">&lt;Property name="UsernameWithEmailJavaScriptRegEx"&gt;</span>
PasswordJavaScriptRegEx	Policy that defines the password format.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.

UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.
	<pre>&lt;Property     name="UsernameJavaRegEx"&gt;[ a-zA-Z0-9 ._-  !#\$%"' *-=?^</pre>
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role names.
RolenameJavaRegEx	A regular expression used to validate role names. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.
ReadGroups	Specifies whether groups should be read from the user store. If this is set to true, then the underlying user store can be read, and the following group configurations are NOT mandatory.
WriteGroups	Specifies whether groups should be written to user store.
EmptyRolesAllowed	Specifies whether the underlying user store allows empty groups to be created. By default, it is false, which means that empty groups are allowed to be created. Usually LDAP servers do not allow empty groups.
GroupSearchBase	DN of the context under which user entries are stored in the user store.
GroupSearchFilter	The query used to search for groups.
GroupNameListFilter	Filtering criteria for listing all the group entries in the user store. Group search operation only returns objects created from this class.
GroupEntryObjectClass	Object class used to construct group entries.
GroupNameSearchFilter	Filtering criteria used to search for a particular group entry.
GroupNameAttribute	Attribute used for uniquely identifying a user entry. This attribute is mandatory.
MembershipAttribute	Attribute used to define members of groups.
UserRolesCacheEnabled	This is to indicate whether to cache the role list of a user. By default, it is true. If this is false, changes made through external means and those changes should be instantly reflected in the user store.
UserDNPattern	(LDAP) The pattern for the user's DN, which can be defined to improve performance. It is recommended to use a simple pattern. Defining a UserDNPattern provides more impact on performance than using a complex pattern.
ReplaceEscapeCharactersAtUserLogin	(LDAP) If the user name has special characters it replaces it to valid characters.
TenantManager	Includes the location of the tenant manager.
ReadOnly	(LDAP and JDBC) Indicates whether the user store of this realm or not.
IsEmailUserName	(JDBC) Indicates whether the user's email is used as their username.
DomainCalculation	(JDBC) Can be either default or custom (this applies when the realm is JDBC).
PasswordDigest	(JDBC) Digesting algorithm of the password. Has values such as, MD5, SHA1, SHA256, SHA512, etc.
StoreSaltedPassword	(JDBC) Indicates whether to salt the password.
UserNameUniqueAcrossTenants	(JDBC) An attribute used for multi-tenancy.
PasswordJavaRegEx	(LDAP and JDBC) A regular expression to validate passwords. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.

PasswordJavaScriptRegEx	The regular expression used by the front-end components for password validation.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to be alphanumeric.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.
RolenameJavaRegEx	A regular expression to validate role names. By default, strings have to be alphanumeric.
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role name validation.
MultiTenantRealmConfigBuilder	Tenant Manager specific realm config parameter. Can be used to build the configuration for a tenant.
SharedGroupEnabled	This property is used to enable/disable the shared role functionality.
SharedGroupSearchBase	Shared roles are created for other tenants to access under the merged search base.
SharedTenantObjectClass	Object class for the shared groups created.
SharedTenantNameAttribute	Name attribute for the shared group.
SharedTenantNameListFilter	This is currently not used.

## Configuring Secondary User Stores

The default configurations of WSO2 products have a single, embedded user store (primary user store). If required, you can configure WSO2 products to connect to several secondary user stores as well. After configuration, users from different stores can log in and perform operations depending on their roles/permissions. You can also configure your own customized user stores and connect them with the products as secondary stores.

The topics below explain how to configure secondary user stores manually or using the management console:

- Configuring using the management console
- Configuring manually
- Related topics

The default configurations of WSO2 products have a single, embedded user store (primary user store). If required, you can configure WSO2 products to connect to several secondary user stores as well. After configuration, users from different stores can log in and perform operations depending on their roles/permissions. You can also configure your own customized user stores and connect them with the products as secondary stores.

The topics below explain how to configure secondary user stores manually or using the management console:

- Configuring using the management console
- Configuring manually



## Before you begin:

If you are setting up a database other than the default H2 that comes with the product to store user information, select the script relevant to your database type from the <PRODUCT\_HOME>/dbscripts folder and run it on your database. It creates the necessary tables.

### Configuring using the management console

1. Log in to the management console and click **User Store Management** sub menu under **Configure** menu.
2. The **User Store Management** page opens. Initially, there are no secondary user stores.



**Note:** You cannot update the PRIMARY user store at run time, so it is not visible on this page.

3. Click **Add Secondary User Store**.
4. In the User Store Manager Class list, select the type of user store you are creating:

User store manager	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAPUserStoreManager
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUserStoreManager
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUserStoreManager. This can be used in read-only mode, you must use ActiveDirectoryUserStoreManager
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager. This can be configured for JDBCUserStoreManager property: <Property name="JDBCUserStoreManager" value="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager" />

You can also populate this drop-down list with custom user store manager implementations by adding them to the server. A sample custom user store manager can be found in [the repository](#).

- Enter a unique domain name with no underscore (\_) characters, and optionally enter a description for this user store.
- Enter values for the properties, using the descriptions in the Descriptions column for guidance. The properties that appear vary based on the user store manager class you selected, and there may be additional properties in an Optional or Advanced section at the bottom of the screen. See the [related topics](#) for descriptions of user store properties.

User Store Manager

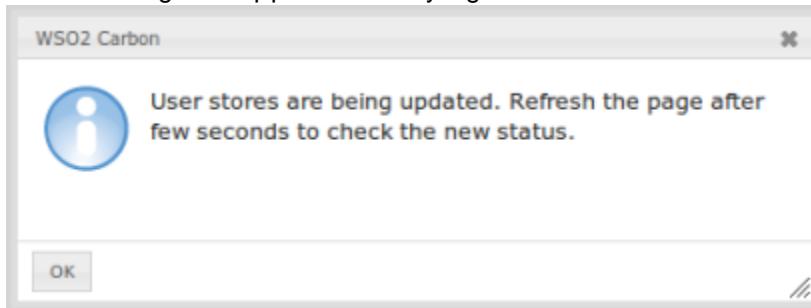
User Store Manager

User Store Manager Class	org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Depending on the class, properties needs to be defined.
Domain Name*	wso2.com	
Description	external database with read write access	

Define Properties For Wso2

Property Name	Property Value	Description
ConnectionName*	uid=admin,ou=system	This should be the DN (Distinguish Name) of the admin user in LDAP
ConnectionURL*	localhost:\${Ports.EmbeddedLDAP.LDAPServerPort}/	Connection URL for the user store
ConnectionPassword*	*****	Password of the admin user
UserSearchBase*	ou=Users,dc=wso2,dc=org	DN of the context under which user entries are stored in LDAP
Disabled*	<input type="checkbox"/>	Whether user store is disabled
UserNameListFilter*	(objectClass=person)	Filtering criteria for listing all the user entries in LDAP
UserNameAttribute*	uid	Attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, uid and etc
UserNameSearchFilter*	(&(objectClass=person)(uid=?))	Filtering criteria for searching a particular user entry
UserEntryObjectClass*	wso2Person	Object Class used to construct user entries

- Ensure that all the mandatory fields are filled and a valid domain name is given and click **Add**.
- A message appears saying that the user stores are being added.



**Note:** The above message does not imply that the user store is added successfully. It simply means that the server is attempting to add the new user store to the end of the available chain of stores.

9. Refresh the page after a few seconds to check the status.
10. If the new user store is successfully added, it will appear in the **User Store Management** page.
11. After adding to the server, you can edit the properties of the new secondary user store and enable/disable it in a dynamic manner.

### Configuring manually

By default, the configuration of the primary user store is saved in the `user-mgt.xml` file. When you create a secondary user store using the management console as explained above, its configuration is saved to an XML file with the same name as the domain name you specify. Alternatively, you can create this XML file manually and save it as follows:

- When you configure multiple user stores, you must **give a unique domain name to each user store** in the `<DomainName>` element. If you configure a user store without specifying a domain name, the server throws an exception at start up.
- If it is the configuration of a super tenant, save the secondary user store definitions in `<PRODUCT_HOME>/repository/deployment/server/userstores` directory.
- If it is a general tenant, save the configuration in `<PRODUCT_HOME>/repository/tenants/<tenantid>/userstores` directory.
- The secondary user store configuration file must have the same name as the domain with an underscore (\_) in place of the period. For example, if the domain is `wso2.com`, name the file as `wso2_com.xml`.
- One file only contains the definition for one user store domain.

### Related topics

#### Directing the Root Context to the API Store

WSO2 API Manager includes separate Web applications as the API Publisher and the API Store. The root context of the API Manager is set to go to the API Publisher by default. For example, assume that the API Manager is hosted on a domain named `apis.com` with default ports. The URLs of the API Store and API Publisher will be as follows:

- API Store - <https://apis.com:9443/store>
- API Publisher - <https://apis.com:9443/publisher>

If you open the root context, which is <https://apis.com:9443> in your browser, it directs to the API Publisher by default. You can set this to go to the API Store as follows:

1. Open the bundle `<AM_HOME>/repository/components/plugins/org.wso2.am.styles_1.x.x.jar`.
2. Open the `component.xml` file that is inside `META-INF` directory.
3. Change the `<context-name>` element, which points to publisher by default, to store:

```
<context>
    <context-id>default-context</context-id>
    <context-name>store</context-name>
    <protocol>http</protocol>
    <description>API Publisher Default Context</description>
</context>
```

4. Restart the server.
5. Open the default context (<https://apis.com:9443>) again in a browser and note that it directs to the API Store.

**Tip:** If you want to configure the API Publisher and Store to pass proxy server requests, configure a **reverse**

proxy server.

## Adding Links to Navigate Between the Store and Publisher

By default, there are no links in the UIs of the API Store and API Publisher applications to traverse between the two apps.

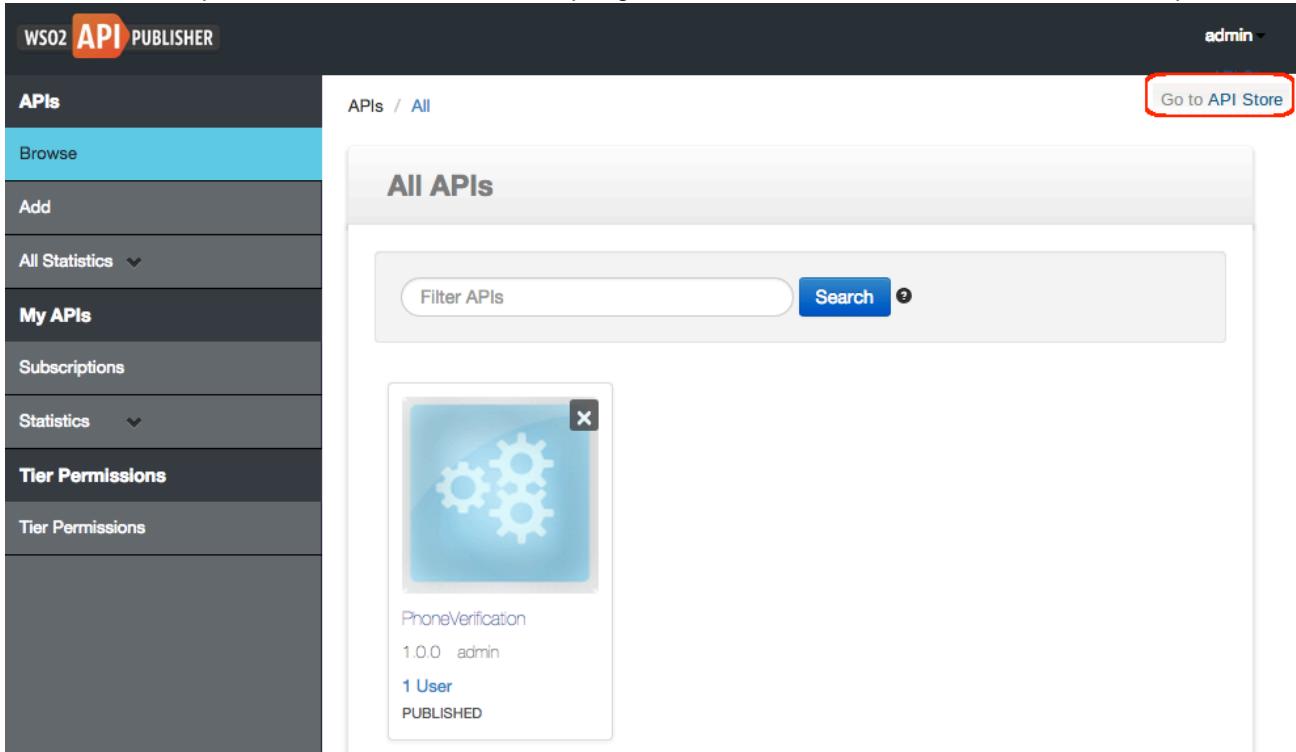
To add a link in API Publisher to API Store:

1. In <AM\_HOME>/repository/conf/api-manager.xml file, set the <DisplayURL> to true and provide the URL of the Store. For example,

 If you are using a **distributed/clustered API Manager setup**, this configuration must be done in the API Publisher node/s.

```
<APIStore>
    <DisplayURL>true</DisplayURL>
    <URL>https://<hostname of the API Publisher node>:9443/store</URL>
</APIStore>
```

2. Note a URL that points to the API Store in the top, right-hand corner of the API Publisher. For example,



The screenshot shows the WSO2 API Publisher application. The top navigation bar includes the WSO2 logo, 'API PUBLISHER', and a user dropdown for 'admin'. Below the header, a sidebar on the left lists 'APIS' with options like 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', and 'Tier Permissions'. The main content area is titled 'All APIs' and displays a single API entry: 'PhoneVerification' (version 1.0.0, created by admin, published, 1 user). At the top of the main content area, there is a 'Go to API Store' button, which is highlighted with a red box.

To add a link in API Store to API Publisher:

1. In <AM\_HOME>/repository/conf/api-manager.xml file, set the <DisplayURL> to true and provide the URL of the Publisher. For example,

 If you are using a **distributed/clustered API Manager setup**, this configuration must be done in the API Store node/s.

```
<APIPublisher>
    <DisplayURL>true</DisplayURL>
    <URL>https://<hostname of the API Store node>:9443/publisher</URL>
</APIPublisher>
```

2. Note a URL in the API Store that points to the API Publisher. For example,

The screenshot shows the WSO2 API Store dashboard. At the top, there's a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Themes, and admin. Below the navigation bar, there's a search bar labeled 'Search API'. On the left, there's a 'Recently Added' section featuring a thumbnail for 'PhoneVerification-1.0.0' by 'admin' with a 5-star rating. The main area is titled 'APIs' and shows a single entry for 'PhoneVerification' version 1.0.0 by 'admin'. To the right, there's a 'Tags' section. In the top right corner of the main content area, there's a link 'Go to API Publisher' which is highlighted with a red box.

- For information on clustering, see [Clustering WSO2 API Manager](#).
- For information on deployment patterns, see [Deployment Patterns of WSO2 API Manager](#).

## Maintaining Separate Production and Sandbox Gateways

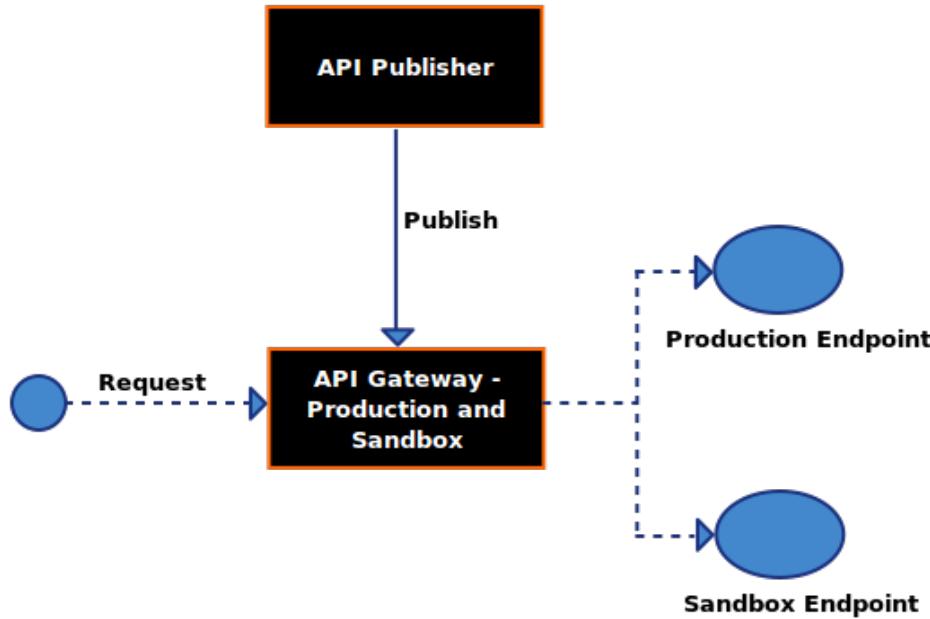
With WSO2 API Manager, you can maintain a production and a sandbox endpoint for a given API. The production endpoint is the actual location of the API, whereas the sandbox endpoint points to its testing/pre-production environment.

When you publish an API using the API Publisher, it gets deployed on the API Gateway. By default, there's a single Gateway instance (deployed either externally or embedded within the publisher), but you can also set up multiple Gateways:

- Single Gateway to handle both production and sandbox requests
- Multiple Gateways to handle production and sandbox requests separately

### Single Gateway to handle both production and sandbox requests

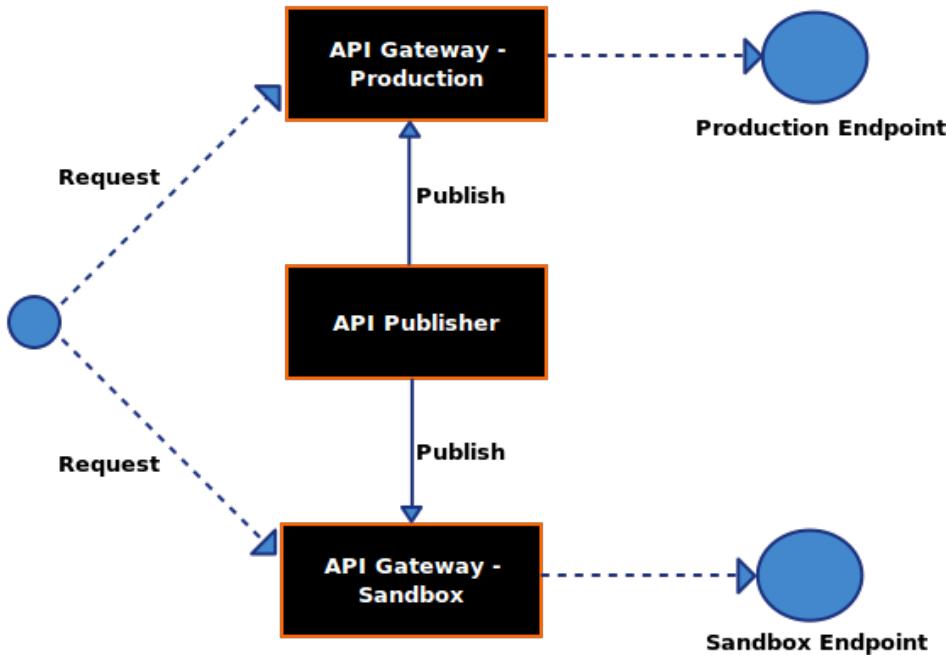
This is the default scenario. Because this Gateway instance handles both production and sandbox token traffic, it is called a hybrid API Gateway. When an API request comes to the API Gateway, it checks whether the requesting token is of type PRODUCTION or SANDBOX and forwards the request to the appropriate endpoint. The diagram below depicts this scenario.



#### Multiple Gateways to handle production and sandbox requests separately

Having a single gateway instance to pass through both types of requests can negatively impact the performance of the production server. To avoid this, you can set up separate API Gateways. The production API Gateway handles requests that are made using PRODUCTION type tokens and the sandbox API Gateway handles requests that are made using SANDBOX type tokens.

The diagram below depicts this using two Gateways:



In either of the two approaches, if an API Gateway receives an invalid token, it returns an error to the requesting client saying that the token is invalid.

You configure production and sandbox gateways using the `<Environments>` element in the `<AM_HOME>/reposi`

tory/conf/api-manager.xml file as shown in the following example:

```
<Environments>
<Environment type="production">
    <Name>Production and Sandbox</Name>
    <ServerURL>https://localhost:9445/services/&lt;/ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>

    <GatewayEndpoint>http://localhost:8282,https://localhost:8245&lt;/GatewayEndpoint>
</Environment>
<Environment type="sandbox">
    <Name>Production and Sandbox</Name>
    <ServerURL>https://localhost:9448/services/&lt;/ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>

    <GatewayEndpoint>http://localhost:8285,https://localhost:8248&lt;/GatewayEndpoint>
</Environment>
</Environments>
```

The `type` attribute of the `<Environment>` element can take the following values:

- **Production:** A production type Gateway
- **Sandbox:** A sandbox type Gateway
- **Hybrid:** The Gateway handles both types of tokens

If you work with Gateways in different geographical locations, configuring multiple environments using the `<APIGateway>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file is recommended. The diagram below depicts a sample setup:

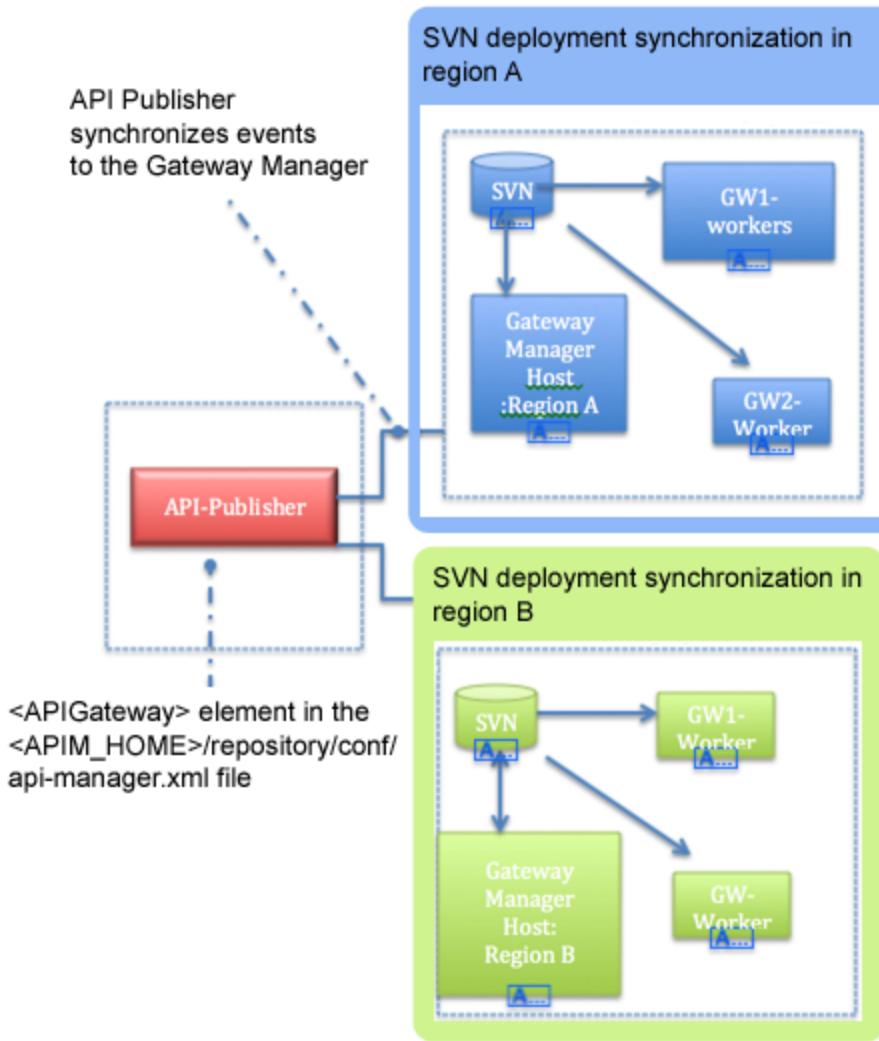


Figure: API Gateways in different geographical regions

## Configuring Transports

A transport is responsible for carrying messages that are in a specific format. WSO2 API Manager supports all the widely used transports including HTTP/s, JMS, Pass-through and VFS, and domain-specific transports like FIX. All WSO2 transports are directly or indirectly based on the Apache Axis2 transports framework. This framework provides two main interfaces that each transport implementation has.

- **org.apache.axis2.transport.TransportListener:** Implementations of this interface specify how incoming messages are received and processed before handing them over to the Axis2 engine for further processing.
- **org.apache.axis2.transport.TransportSender:** Implementations of this interface specify how a message can be sent out from the Axis2 engine.

Because each transport has to implement the two interfaces above, each transport generally contains a transport receiver/listener and a transport sender. You configure, enable, and manage transport listeners and senders independently to each other. For example, you can enable just the JMS transport sender without having to enable the JMS transport listener.

For more information, see the following topics in the WSO2 ESB documentation:

- Available transports
- How to configure transports

## Transforming API Message Payload

When a request comes to the API Manager, it sends the response in the same format of the request. For example, the API Manager handles JSON to JSON transformations out of the box. If the backend does not accept messages of the same content type of the request message, it must be transformed to a different format. The API Gateway of the API Manager handles these transformations using message builders and formatters.

When a message comes in to the API Gateway, the receiving transport selects a **message builder** based on the message's content type. It uses that builder to process the message's raw payload data and convert it into JSON. Conversely, when sending a message out from the Gateway, a **message formatter** is used to build the outgoing stream from the message. As with message builders, the message formatter is selected based on the message's content type.

- JSON message builders and formatters
- XML representation of JSON payloads
- Converting a payload between XML and JSON

 Note that if you edit an API's synapse configuration as mentioned in this guide and then go back to the API Publisher and save the API, your changes will be overwritten. Therefore, we do not recommend changing the API's synapse configuration directly. The recommended way to extend an API's mediation flow is by [engaging sequences](#).

Also see the following sections in the WSO2 ESB documentation. WSO2 ESB is used to implement the API Gateway through which API messages are transformed:

- Accessing content from JSON payloads
- Logging JSON payloads
- Constructing and transforming JSON payloads
- Troubleshooting, debugging, and logging

#### **JSON message builders and formatters**

There are two types of message builders and formatters for JSON. The default builder and formatter keep the JSON representation intact without converting it to XML. You can access the payload content using the JSON Path or XPath and convert the payload to XML at any point in the mediation flow.

- org.apache.synapse.commons.json.JsonStreamBuilder
- org.apache.synapse.commons.json.JsonStreamFormatter

If you want to convert the JSON representation to XML before the mediation flow begins, use the following builder and formatter instead. Note that some data loss can occur during the JSON -> XML -> JSON conversion process.

- org.apache.synapse.commons.json.JsonBuilder
- org.apache.synapse.commons.json.JsonFormatter

The builders and formatters are configured respectively in the `messageBuilders` and `messageFormatters` sections of the Axis2 configuration files located in the `<PRODUCT_HOME>/repository/conf/axis2` directory. Both types of JSON builders use **StAXON** as the underlying JSON processor.

The following builders and formatters are also included for compatibility with older API Manager versions:

- org.apache.axis2.json.JSONBuilder/JSONMessageFormatter
- org.apache.axis2.json.JSONStreamBuilder/JSONStreamFormatter
- org.apache.axis2.json.JSONBadgerfishOMBuilder/JSONBadgerfishMessageFormatter

 Always use the same type of builder and formatter combination. Mixing different builders and formatters will cause errors at runtime.

If you want to handle JSON payloads that are sent using a media type other than `application/json`, you must register the JSON builder and formatter for that media type in the following two files at minimum (for best results, register them in all Axis2 configuration files found in the `<PRODUCT_HOME>/repository/conf/axis2` directory):

- <PRODUCT\_HOME>/repository/conf/axis2/axis2.xml
- <PRODUCT\_HOME>/repository/conf/axis2/axis2\_blocking\_client.xml

For example, if the media type is `text/javascript`, register the message builder and formatter as follows:

```
<messageBuilder contentType="text/javascript"
    class="org.apache.synapse.commons.json.JsonStreamBuilder" />

<messageFormatter contentType="text/javascript"
    class="org.apache.synapse.commons.json.JsonStreamFormatter" />
```

 When you modify the builders/formatters in Axis2 configuration, make sure that you have enabled only one correct message builder/formatter pair for a given media type.

### ***XML representation of JSON payloads***

When building the XML tree, JSON builders attach the converted XML infoset to a special XML element that acts as the root element of the final XML tree. If the original JSON payload is of type `object`, the special element is `<json Object/>`. If it is an `array`, the special element is `<jsonArray/>`. Following are examples of JSON and XML representations of various objects and arrays.

#### ***Null objects***

JSON:

```
{"object":null}
```

XML:

```
<jsonObject>
    <object></object>
</jsonObject>
```

#### ***Empty objects***

JSON:

```
{"object":{}}
```

XML:

```
<jsonObject>
    <object></object>
</jsonObject>
```

#### ***Empty strings***

JSON:

```
{ "object": "" }
```

XML:

```
<j(jsonObject>
  <object></object>
</j(jsonObject>
```

### ***Empty array***

JSON:

```
[ ]
```

XML (JsonStreamBuilder):

```
<jsonArray></jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
  <?xml-multiple jsonElement?>
</jsonArray>
```

### ***Named arrays***

JSON:

```
{ "array": [1, 2] }
```

XML (JsonStreamBuilder):

```
<j jsonObject>
  <array>1</array>
  <array>2</array>
</j jsonObject>
```

XML (JsonBuilder):

```
<j jsonObject>
  <?xml-multiple array?>
  <array>1</array>
  <array>2</array>
</j jsonObject>
```

JSON:

```
{ "array" : [ ] }
```

XML (JsonStreamBuilder):

```
<jsonObject></jsonObject>
```

XML (JsonBuilder):

```
<jsonObject>
<?xml-multiple array?>
</jsonObject>
```

### ***Anonymous arrays***

JSON:

```
[1,2]
```

XML (JsonStreamBuilder):

```
<jsonArray>
<jsonElement>1</jsonElement>
<jsonElement>2</jsonElement>
</jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
<?xml-multiple jsonElement?>
<jsonElement>1</jsonElement>
<jsonElement>2</jsonElement>
</jsonArray>
```

JSON:

```
[1, []]
```

XML (JsonStreamBuilder):

```
<jsonArray>
<jsonElement>1</jsonElement>
<jsonElement>
    <jsonArray></jsonArray>
</jsonElement>
</jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
<?xml-multiple jsonElement?>
<jsonElement>1</jsonElement>
<jsonElement>
    <jsonArray>
        <?xml-multiple jsonElement?>
    </jsonArray>
</jsonElement>
</jsonArray>
```

### ***XML processing instructions (PIs)***

Note that the addition of `xml-multiple` processing instructions to the XML payloads whose JSON representations contain arrays. `JsonBuilder` (via StAXON) adds these instructions to the XML payload that it builds during the JSON to XML conversion so that during the XML to JSON conversion, `JsonFormatter` can reconstruct the arrays that are present in the original JSON payload. `JsonFormatter` interprets the elements immediately following a processing instruction to construct an array.

### ***Special characters***

When building XML elements, the '\$' character and digits are handled in a special manner when they appear as the first character of a JSON key. Following are examples of two such occurrences. Note the addition of the `_JsonReader_PS_` and `_JsonReader_PD_` prefixes in place of the '\$' and digit characters, respectively.

JSON:

```
{"$key":1234}
```

XML:

```
<jsonObject>
<_JsonReader_PS_key>1234</_JsonReader_PS_key>
</jsonObject>
```

JSON:

```
{"32X32":"image_32x32.png"}
```

XML:

```
<jsonObject>
<_JsonReader_PD_32X32>image_32x32.png</_JsonReader_PD_32X32>
</jsonObject>
```

### ***Converting a payload between XML and JSON***

To convert an XML payload to JSON, set the `messageType` property to `application/json` in the `axis2` scope before sending message to an endpoint. Similarly, to convert a JSON payload to XML, set the `messageType` property to `application/xml` or `text/xml`. For example:

```

<api name="admin--TOJSON" context="/tojson" version="1.0" version-type="url">
    <resource methods="POST GET DELETE OPTIONS PUT" url-mapping="/">
        <inSequence>
            <property name="POST_TO_URI" value="true" scope="axis2"/>
            <property name="messageType" value="application/json" scope="axis2"/>
            <filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
                <then>
                    <send>
                        <endpoint name="admin--Test_APIproductionEndpoint_0">
                            <http
uri-template="http://localhost:9767/services/StudentService">
                                <timeout>
                                    <duration>30000</duration>
                                    <responseAction>fault</responseAction>
                                </timeout>
                                <suspendOnFailure>
                                    <errorCodes>-1</errorCodes>
                                    <initialDuration>0</initialDuration>
                                    <progressionFactor>1.0</progressionFactor>
                                    <maximumDuration>0</maximumDuration>
                                </suspendOnFailure>
                                <markForSuspension>
                                    <errorCodes>-1</errorCodes>
                                </markForSuspension>
                            </http>
                        </endpoint>
                    </send>
                </then>
                <else>
                    <sequence key="_sandbox_key_error_" />
                </else>
            </filter>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
            <property name="id" value="A"/>
            <property name="policyKey"
value="gov:/apimgt/applicationdata/tiers.xml"/>
        </handler>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
    </handlers>

```

An example command to invoke above API:

```
curl -v -X POST -H "Content-Type:application/xml" -H "Authorization: Bearer xxx"
-d@request1.xml "http://10.100.1.110:8280/tojson/1.0"
```

If the request payload is as follows:

```
<coordinates>
  <location>
    <name>Bermuda Triangle</name>
    <n>25.0000</n>
    <w>71.0000</w>
  </location>
  <location>
    <name>Eiffel Tower</name>
    <n>48.8582</n>
    <e>2.2945</e>
  </location>
</coordinates>
```

The response payload will look like this:

```
{
  "coordinates": {
    "location": [
      {
        "name": "Bermuda Triangle",
        "n": 25.0000,
        "w": 71.0000
      },
      {
        "name": "Eiffel Tower",
        "n": 48.8582,
        "e": 2.2945
      }
    ]
  }
}
```

Note that we have used the Property mediator to mark the outgoing payload to be formatted as JSON. For more information about the Property Mediator, see the [Property Mediator](#) page on WSO2 ESB documentation.

```
<property name="messageType" value="application/json" scope="axis2" />
```

Similarly if the response message needs to be transformed, set the messageType property in the outSequence.

```

<api name="admin--TOJSON" context="/tojson" version="1.0" version-type="url">
    <resource methods="POST GET DELETE OPTIONS PUT" url-mapping="/">
        <inSequence>
            <property name="POST_TO_URI" value="true" scope="axis2"/>
            <filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
                <then>
                    <send>
                        <endpoint name="admin--Test_APIproductionEndpoint_0">
                            <http
uri-template="http://localhost:9767/services/StudentService">
                                <timeout>
                                    <duration>30000</duration>
                                    <responseAction>fault</responseAction>
                                </timeout>
                                <suspendOnFailure>
                                    <errorCodes>-1</errorCodes>
                                    <initialDuration>0</initialDuration>
                                    <progressionFactor>1.0</progressionFactor>
                                    <maximumDuration>0</maximumDuration>
                                </suspendOnFailure>
                                <markForSuspension>
                                    <errorCodes>-1</errorCodes>
                                </markForSuspension>
                            </http>
                        </endpoint>
                    </send>
                </then>
                <else>
                    <sequence key="_sandbox_key_error_" />
                </else>
            </filter>
        </inSequence>
        <outSequence>
            <property name="messageType" value="application/json" scope="axis2"/>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
            <property name="id" value="A"/>
            <property name="policyKey"
value="gov:/apimgt/applicationdata/tiers.xml"/>
        </handler>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
    </handlers>

```

## Sharing Applications and Subscriptions

The API Manager provides facility to users of a specific logical group such as an organization to view each others' applications and subscriptions.

By default, the API Manager considers the organization name that you give at the time you sign up to the API Store as the group ID. It extracts the claim <http://wso2.org/claims/organization> of a user and uses the value specified in it as the group ID. This way, all users who specify the same organization name belong to the same group and therefore, can view each others' subscriptions and applications. The API Manager also provides flexibility to change this default authentication implementation.

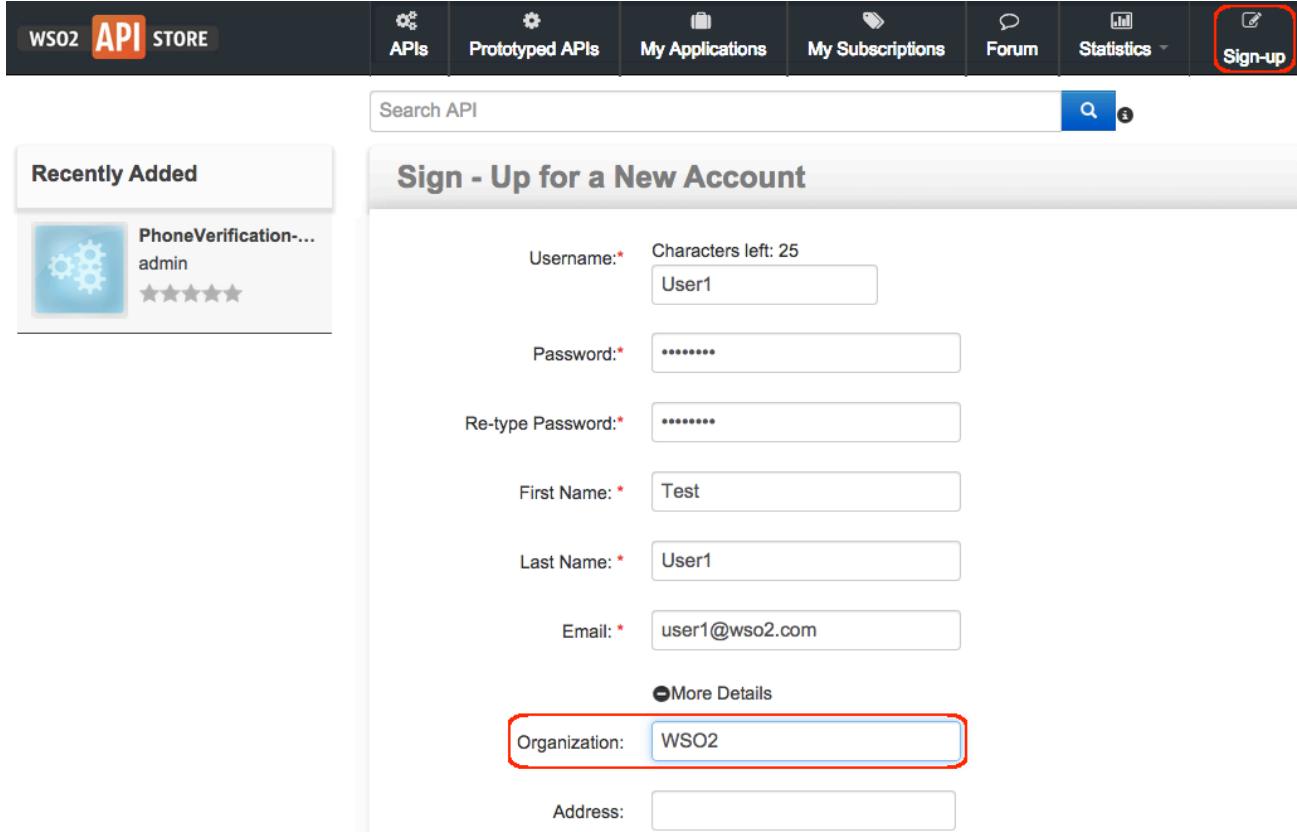
The steps below explain how to share applications and subscriptions.

1. Uncomment the <GroupingExtractor> element in the < APIM\_HOME>/repository/conf/api-manager.xml file.

```
<GroupingExtractor>org.wso2.carbon.apimgt.impl.DefaultGroupIdExtractorImpl</GroupingExtractor>
```

 **Tip:** This default extractor doesn't work with SAML SSO. You need to write a custom implementation using the [WSO2ISGroupIdExtractor.java](#) class as an example.

2. Start the API Manager and sign up to the API Store as two different users (user1 and user2) with the same organization name. For example,



The screenshot shows the WSO2 API Manager's sign-up interface. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, and a red-highlighted 'Sign-up' button. Below the navigation bar is a search bar labeled 'Search API'. On the left, there is a sidebar titled 'Recently Added' showing a thumbnail of a gear icon and the text 'PhoneVerification...' followed by 'admin' and a five-star rating. The main area is titled 'Sign - Up for a New Account'. It contains several input fields with validation messages: 'Username:' (User1), 'Password:' (\*\*\*\*\*), 'Re-type Password:' (\*\*\*\*\*), 'First Name:' (Test), 'Last Name:' (User1), and 'Email:' (user1@wso2.com). Below these fields is a link 'More Details'. Under 'More Details', there is another input field for 'Organization:' which has 'WSO2' typed into it and is also highlighted with a red box. There is also a field for 'Address:' which is currently empty.

3. Log in as user1, create a new application (e.g., TestApp1) and subscribe to an API using the new application.

The screenshot shows the WSO2 API Manager interface. At the top, there is a navigation bar with icons for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Themes, and a user icon labeled "user1". Below the navigation bar is a search bar with the placeholder "Search API" and a magnifying glass icon. The main content area displays the "PhoneVerification - 1.0.0" API details. On the left, there is a profile picture for "admin". To the right, the API details are listed: Rating (Your rating: N/A, 5 stars), Version (1.0.0), Status (PUBLISHED), and Updated (23/Jun/2015 10:33:08 AM IST). On the right side, there are dropdown menus for "Applications" (set to "TestApp1") and "Tiers" (set to "Bronze"). A note below states "Allows 1 request(s) per minute." A blue "Subscribe" button is located at the bottom right. Below the main content, there are tabs for Overview, Documentation, API Console, and Throttling Info.

4. Log out of the API Store and log back in as user2.
5. Go to the **My Subscriptions** page, select the application that the previous user created (e.g., TestApp1) from the drop-down list and note that the previous user's subscription is listed under **Subscribed APIs**.

The screenshot shows the WSO2 API Manager interface. At the top, there is a navigation bar with icons for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Themes, and a user icon labeled "user2". Below the navigation bar is a search bar with the placeholder "Search API" and a magnifying glass icon. The main content area displays the "Subscriptions" page. A dropdown menu labeled "TestApp1" is shown. To the right, there is a checkbox labeled "Show Keys" which is checked. Below the dropdown, there are links for "Keys - Production" and "Keys - Sandbox". A section titled "Subscribed APIs" contains a box with a red border. Inside the box, it shows "PhoneVerif.. - 1.0.0" with a delete icon, "admin", and "Bronze Subscription".

## Extending the API Manager

The following topics cover different ways in which you can extend the API Manager:

- Writing Custom Handlers
- Adding Mediation Extensions

- Integrating with WSO2 Governance Registry
- Adding Workflow Extensions
- Adding new Throttling Tiers
- Adding a Reverse Proxy Server
- Adding a new API Store Theme

## Writing Custom Handlers

This section introduces handlers and using an example, explains how to write a custom handler:

- Introducing Handlers
- Writing a custom handler
- Engaging the custom handler

### ***Introducing Handlers***

You find the default handlers in any API's Synapse definition as shown below.

hen an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/api folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration.

```
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
    <property name="id" value="A"/>
    <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
</handler>
<handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
<handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
</handlers>
```

Let's see what each handler does:

- **APIAuthenticationHandler:** Validates the OAuth2 bearer token used to invoke the API. It also determines whether the token is of type Production or Sandbox and sets MessageContext variables as appropriate.
- **APIThrottleHandler:** Throttles requests based on the throttling policy specified by the `policyKey` property. Throttling is applied both at the application level as well as subscription level.
- **APIMgtUsageHandler:** Publishes events to BAM for collection and analysis of statistics. This handler only comes to effect if API usage tracking is enabled. See [Publishing API Runtime Statistics](#) for more information.
- **APIMgtGoogleAnalyticsTrackingHandler:** Publishes events to Google Analytics. This handler only comes into effect if Google analytics tracking is enabled. See [Integrating with Google Analytics](#) for more information.
- **APIManagerExtensionHandler:** Triggers extension sequences. By default, the extension handler is listed at last in the handler chain, and therefore is executed last. You cannot change the order in which the handlers are executed, except the extension handler. To configure the API Gateway to execute extension handler first, uncomment the `<ExtensionHandlerPosition>` section in the `<APIM_HOME>/repository/conf/api-manager.xml` file and provide the value `top`. This is useful when you want to execute your own extensions before our default handlers in situations like doing additional security checks such as signature verification on access tokens before executing the default security handler. See [Adding Mediation Extensions](#).

### ***Writing a custom handler***

Let's see how you can write a custom handler and apply it to the API Manager. In this example, we extend the authentication handler. Make sure your custom handler name is not the same as the name of an existing handler.

WSO2 API Manager provides the OAuth2 bearer token as its default authentication mechanism. A sample implementation is [here](#). Similarly, you can extend the API Manager to support any custom authentication mechanism by writing your own authentication handler class. This custom handler must extend `org.apache.synapse.rest.AbstractHandler` class and implement the `handleRequest()` and `handleResponse()` methods.

Given below is an example implementation:

```
package org.wso2.carbon.test;

import org.apache.synapse.MessageContext;
import org.apache.synapse.core.axis2.Axis2MessageContext;
import org.apache.synapse.rest.AbstractHandler;

import java.util.Map;

public class CustomAPIAuthenticationHandler extends AbstractHandler {

    public boolean handleRequest(MessageContext messageContext) {
        try {
            if (authenticate(messageContext)) {
                return true;
            }
        } catch (APISecurityException e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean handleResponse(MessageContext messageContext) {
        return true;
    }

    public boolean authenticate(MessageContext synCtx) throws APISecurityException {
        Map headers = getTransportHeaders(synCtx);
        String authHeader = getAuthorizationHeader(headers);
        if (authHeader.startsWith("userName")) {
            return true;
        }
        return false;
    }

    private String getAuthorizationHeader(Map headers) {
        return (String) headers.get("Authorization");
    }

    private Map getTransportHeaders(MessageContext messageContext) {
        return (Map) ((Axis2MessageContext) messageContext).getAxis2MessageContext().
            getProperty(org.apache.axis2.context.MessageContext.TRANSPORT_HEADERS);
    }
}
```

### ***Engaging the custom handler***

You can engage a custom handler to all APIs at once or only to selected APIs.

To engage to all APIs, the recommended approach is to add it to the `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml` file. For example, the following code segment adds the custom

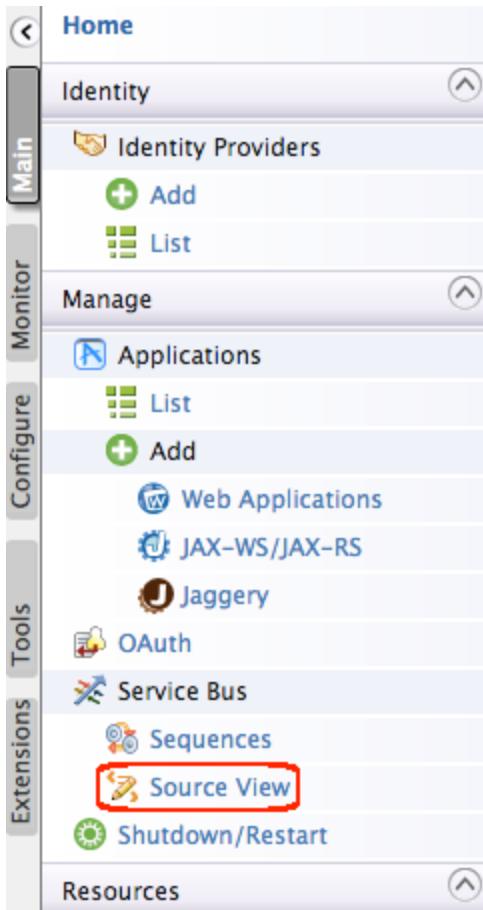
authentication handler that you wrote earlier to the `velocity_template.xml` file while making sure that it skips the default `APIAuthenticationHandler` implementation:

```
<handler
class="org.wso2.carbon.apimgt.custom.authentication.handler.CustomAPIAuthenticationHan
dler" />
    #foreach($handler in $handlers)
        #if(!$handler.className ==
"org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler" )
            <handler xmlns="http://ws.apache.org/ns/synapse"
class="$handler.className">
                #if($handler.hasProperties())
                    #set ($map = $handler.getProperties() )
                    #foreach($property in $map.entrySet())
                        <property name="$!property.key" value="$!property.value"/>
                    #end
                #end
            </handler>
        #end
    #end
</handlers>
```

Given below is how to engage handlers to a single API, by editing its source view.

 **Note** that when you engage a handler by editing the API's source view, your changes will be overwritten every time you save the API through the API Publisher.

1. Build the class and copy the JAR file to `<APIM_HOME>/repository/components/lib` folder.
2. Log in to the management console and select **Service Bus > Source View** in the **Main** menu.



3. In the configuration that opens, select an API and navigate to the <Handlers> section. The following line appears as the first handler. This is the current authentication handler used in the API Manager.

#### Service Bus Configuration

Make the required modifications to the configuration and click 'Update' to apply the changes to the server. Use 'Reset' button to undo your changes.

##### ESB Configuration

```

390     <send/>
391     </outSequence>
392   </resource>
393   <handlers>
394     <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
395     <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
396       <property name="id" value="A"/>
397       <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
398     </handler>
399     <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
400     <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler">
401       <property name="configKey" value="gov:/apimgt/statistics/ga-config.xml"/>
402     </handler>
403     <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
404   </handlers>
405   </api>
406   <api name="admin--PhoneVerification"
407     context="/phoneverify"
408     version="2.0.0"
409     version-type="url">
410     <resource methods="OPTIONS GET">

```

4. Replace the above line with the handler that you created. It will engage your custom handler to the API Manager instance. According to this example, it is as follows:

```

<handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.CustomAPIAuthenticationHa
ndler" />

```

## Adding Mediation Extensions

The API Gateway has a default mediation flow, which you can extend by adding custom mediation sequences. You create a custom mediation sequences either manually or using WSO2 tooling support (i.e., WSO2 Developer Studio), and then engage it per API or globally to all APIs.

### Creating per-API extensions

The recommended way to engage a mediation extension sequence per API is to upload an XML file through the registry and then engage it using the API Publisher. The following tutorial shows how to do this: [Change the Default Mediation Flow of API Requests](#).

Alternatively, you can name the mediation XML file in the pattern <API\_NAME>:v<VERSION>--<DIRECTION> and save it directly in the following location:

- In the **single-tenant mode**, save the XML file in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/sequences directory.
- In the **multi-tenant mode**, save the XML file in the tenant's synapse sequence folder. For example, if tenant id is 1, then save it in <API\_Gateway>/repository/tenants/1/synapse-configs/default/sequences folder.

In the naming pattern, the <DIRECTION> can be **In** or **Out**. When it is **In**, the extension is triggered on the in-flow (request path) and when it is **Out**, the extension is triggered on the out-flow (response path). To change the default fault sequence, you can either modify the default sequence or write a custom fault sequence and engage it to APIs through the API Publisher.

### Creating global extensions

You can also engage mediation extension sequences to all APIs in the API Manager at once. To do that, simply create the XML with the naming pattern **WSO2AM--Ext--<DIRECTION>** and save it in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/sequences directory.

## Integrating with WSO2 Governance Registry

[WSO2 Governance Registry](#) is a registry-repository for storing and managing metadata related to services and other artifacts. Services in the Governance Registry are implemented as [configurable governance artifacts](#) (RXT files). Usually, APIs are created using the API Publisher Web interface. Instead, you can integrate the API Manager with the Governance Registry to directly create APIs in the API Publisher using the services deployed in the Governance Registry.

The steps below explain how to configure the two products to expose services in the Governance Registry as APIs.

- The following steps are applicable for WSO2 API Manager 1.9.x versions and WSO2 Governance Registry version 5.0.0 and above.
- In WSO2 Governance Registry 4.6.0, we do a simple POST to create APIs in the API Publisher. It does not involve registry mounting.

Follow the steps below to publish services on Governance Registry to the API Manager.

1. Download both WSO2 Governance Registry (G-Reg) and WSO2 API Manager.
2. Replace the `<execution forEvent="Publish" class="org.wso2.carbon.governance.registry.extensions.executors.apistore.ApiStoreExecutor">` element in the <GREG\_HOME>/repository/resources/lifecycles/ServiceLifeCycle.xml file with a code block which defines an execution element in production state as shown in the example below. Add the API Manager credentials in it, so that it provides the API Manager's endpoint, username and password as executor parameters.

```

<execution forEvent="Publish"
class="org.wso2.carbon.governance.registry.extensions.executors.apistore.ServiceT
oAPIExecutor">
    <parameter name="apim.endpoint" value="http://localhost:9763//"/>
    <parameter name="apim.username" value="admin"/>
    <parameter name="apim.password" value="admin"/>
    <parameter name="apim.env" value="" />
    <parameter name="default.tier" value="Unlimited"/>
    <parameter name="throttlingTier"
value="Unlimited,Unlimited,Unlimited,Unlimited,Unlimited"/>
</execution>

```

**!** **Note:** If you started the G-Reg server at least once before executing step 2, editing the `configurations.xml` file and restarting the server does not apply the configurations. You need to add the configurations using the G-Reg management console as follows:

- Log in to the G-Reg Management console and select **Extensions -> Configure -> Lifecycles** menu.
- Click the Edit link associated with **ServiceLifeCycle**.
- Add the configuration given in step 2 above and **Save**.

### 3. Run the G-Reg and the API Manager.

When running more than one WSO2 products on the same server, change the default port of one product to avoid port conflicts. You can do this by changing the `<offset>` value of one product in `<PRODUCT_HOME>/repository/conf/carbon.xml` file. In this example, we set the port offset value of Governance Registry to 1 as follows: `<Offset>1</Offset>`

**!** **Note:** If you offset the default API Manager port, you must also change the default API endpoints and the Thrift port accordingly. See [Changing the Default Ports with Offset](#).

- Access the API Manager server using the following URL: <https://<HostName>:9443/carbon>. As you changed the default port of G-Reg, you can access the server using the following URL: <https://<HostName>:<9443+off set>/carbon>.
- Log in to the G-Reg management console and create a new service in it and attach the default service lifecycle to it. For instructions on how to add a new service and associate a new lifecycle, see <http://docs.wso2.org/governance-registry/Managing+Services> in the Governance Registry documentation.

**!** Add an endpoint attribute to the service you create, which matches the WSO2 API Manager environment you defined in the lifecycle executer in step 2.

- Promote the service until it gets to the production state.
- When it is in the production state, publish it using the **Publish** button. You should get a confirmation message once the API is successfully published.
- You have now created an API using a service in the Governance Registry. Open the API Publisher to see that this service is successfully created as an API.

## Adding Workflow Extensions

Use workflow extensions to attach a workflow to the following API Store/API Publisher operations:

- [Adding an Application Creation Workflow](#)
- [Adding an Application Registration Workflow](#)
- [Adding an API Subscription Workflow](#)
- [Adding a User Signup Workflow](#)
- [Invoking the API Manager from the BPEL Engine](#)
- [Customizing a Workflow Extension](#)

- Configuring Workflows for Tenants
- Configuring Workflows in a Cluster
- Changing the Default User Role in Workflows

## Adding an Application Creation Workflow

This section explains how to attach a custom workflow to the application creation operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

 **Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.
- Change the credentials of the workflow configurations in the registry resource \_system/governance/apimgt/applicationdata/workflow-extensions.xml.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Change the user credentials in <APIM\_HOME>/repository/conf/api\_manager.xml file.
- Change the .ht file of the relevant human task.
- Change the allowedRoles parameter in the <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file.

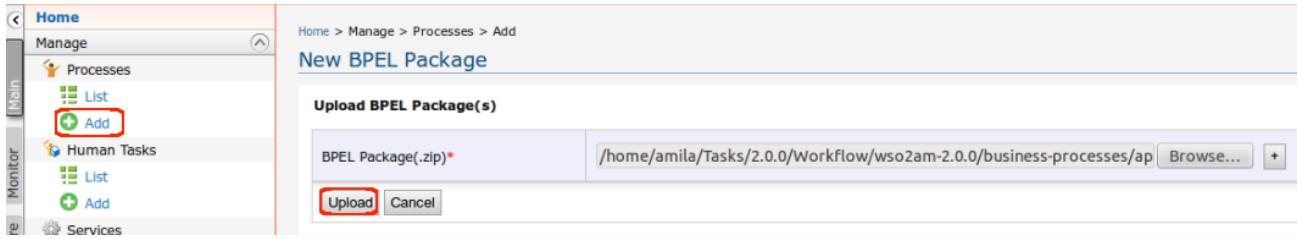
## Configuring the Business Process Server

1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in <BPS\_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

```
<Offset>2</Offset>
```

 **Tip:** If you change the BPS port **offset to a value other than 2 or run the API Manager and BPS on different machines** (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset.)
  - Search and replace port 9445 in <AM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file with the new port (9443 + port offset.)
3. Copy the following from <APIM\_HOME>/business-processes/epr to <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder isn't there, please create it.
    - ApplicationService.epr
    - ApplicationCallbackService.epr
  4. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).
  5. Select **Add** under **Processes** menu and upload the <APIM\_HOME>/business-processes/application-creation/BPEL/ApplicationApprovalWorkFlowProcess\_1.0.0.zip file to BPS. This is the business process archive file.

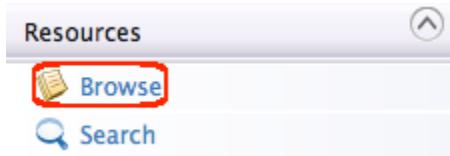


- Select **Add** under the **Human Tasks** menu and upload <APIM\_HOME>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip to BPS. This is the human task archived file.

### **Engaging the WS Workflow Executor in the API Manager**

First, enable the application creation workflow.

- Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



- Go to /\_system/governance/apimgt/applicationdata/workflow-extensions.xml resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```
<WorkFlowExtensions>
...
<ApplicationCreation
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationWSWorkflowExecutor">
    <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationApprovalWorkFlowProcess</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
</ApplicationCreation>
...
</WorkFlowExtensions>
```

**Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

The application creation WS Workflow Executor is now engaged.

- Go to the API Store Web interface, open **My Applications** page and create a new application. It invokes the application creation process and creates a Human Task instance that holds the execution of the BPEL process until some action is performed on it.
- Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
- Log in to the Admin Dashboard (<https://localhost:9443/admin-dashboard>), list all the tasks for

application creation and approve the task. It resumes the BPEL process and completes the application creation.

6. Go back to the **My Applications** page on the API Store and see the created application.

Whenever a user tries to create an application in the API Store, a request is sent to the workflow endpoint. Given below is a sample:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
    <soapenv:Header />
    <soapenv:Body>
        <wor:createApplication
            xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org">
            <wor:applicationName>application1</wor:applicationName>
            <wor:applicationTier>Gold</wor:applicationTier>

            <wor:applicationCallbackUrl>http://webapp/url</wor:applicationCallbackUrl>
                <wor:applicationDescription>Application 1</wor:applicationDescription>
                <wor:tenantDomain>wso2.com</wor:tenantDomain>
                <wor:userName>user1</wor:userName>

            <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExternalRef>

            <wor:callBackURL>https://localhost:8243/services/WorkflowCallbackService</wor:callBackURL>
                </wor:createApplication>
            </soapenv:Body>
        </soapenv:Envelope>
    
```

Elements of the above configuration are described below:

Element	Description
applicationName	Name of the application the user creates.
applicationTier	Throttling tier of the application.
applicationCallbackUrl	When the OAuth2 Authorization Code grant type is applied, this is the endpoint on which the callback needs to happen after the user is authenticated. This is an attribute of the actual application registered on the API Store.
applicationDescription	Description of the application
tenantDomain	Tenant domain associated with the application (domain of the user creating the application).
userName	username of the user creating the application.
workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	At the time of workflow completion, the workflow-completion request is sent to this URL by the workflow engine. This property is configured in the <callBackURL> element in the api-manager.xml.

## Adding an Application Registration Workflow

This section explains how to attach a custom workflow to the application registration operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

### ***Introduction to the application registration workflow***

[Application creation](#) and registration are different workflows. After an application is created, you can subscribe to available APIs, but you get the consumer key/secret and access tokens only after registering the application. There are two types of registrations that can be done to an application: production and sandbox. You change the default application registration workflow in situations such as the following:

1. To issue only sandbox keys when creating production keys is deferred until testing is complete.
2. To restrict untrusted applications from creating production keys. You allow only the creation of sandbox keys.
3. To make API subscribers go through an approval process before creating any type of access token.

 **Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.
- Change the credentials of the workflow configurations in the registry resource \_system/governance/apimgt/applicationdata/workflow-extensions.xml.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Change the user credentials in <APIM\_HOME>/repository/conf/api\_manager.xml file.
- Change the .ht file of the relevant human task.
- Change the allowedRoles parameter in the <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file.

## Configuring the Business Process Server

1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in <BPS\_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

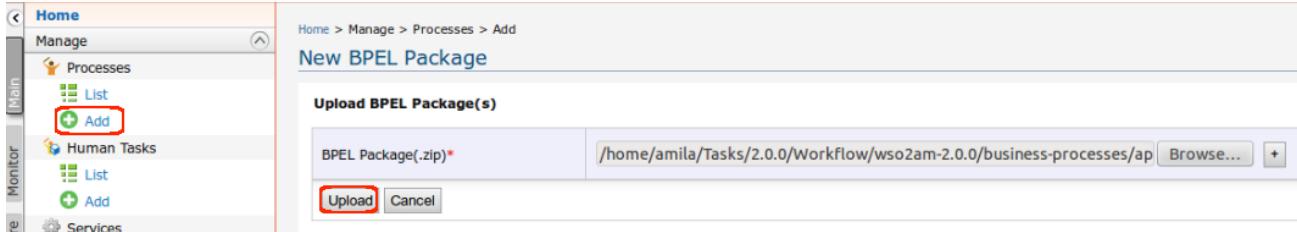
```
<Offset>2</Offset>
```

 **Tip:** If you change the BPS **port offset to a value other than 2 or run the API Manager and BPS on different machines** (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset.)
- Search and replace port 9445 in <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file with the new port (9443 + port offset.)

3. Copy the following from <APIM\_HOME>/business-processes/epr to <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder isn't there, please create it.
  - RegistrationService.epr
  - RegistrationCallbackService.epr
4. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).
5. Select **Add** under **Processes** menu and upload the <APIM\_HOME>/business-processes/application

n-registration/BPEL/ApplicationRegistrationWorkflowProcess\_1.0.0.zip file to BPS. This is the business process archive file.

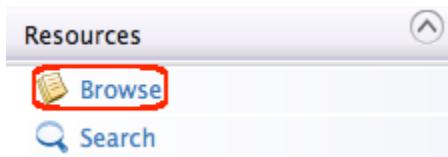


6. Select **Add** under the **Human Tasks** menu and upload <APIM\_HOME>/business-processes/application-registration/HumanTaskBPEL/ApplicationRegistrationTask-1.0.0.zip to BPS. This is the human task archived file.

### **Engaging the WS Workflow Executor in the API Manager**

First, enable the application registration workflow.

1. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



2. Go to /\_system/governance/apimgt/applicationdata/workflow-extensions.xml resource, disable the Simple Workflow Executor and enable WS Workflow Executor:

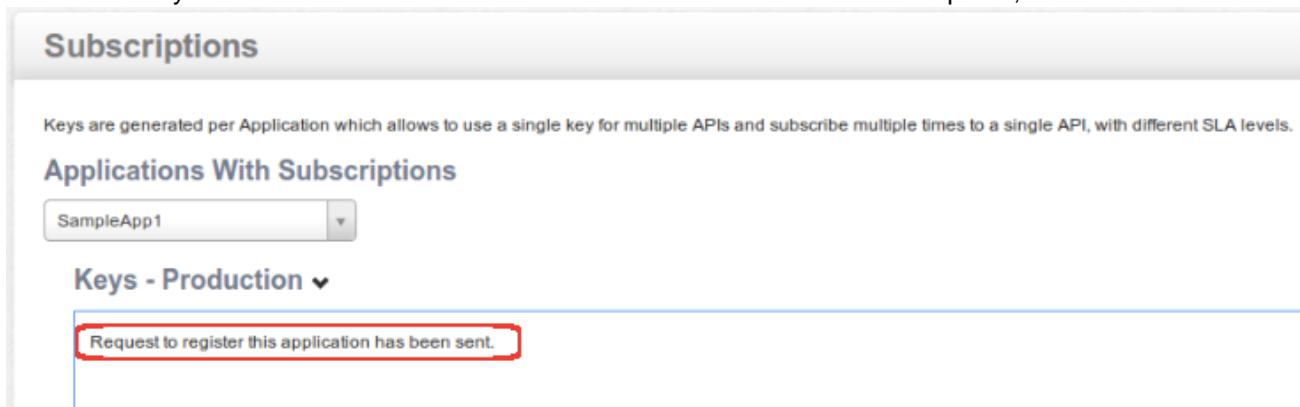
```

<WorkFlowExtensions>
...
    <ProductionApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowE
xecutor">
        <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWork
FlowProcess/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Prope
rty>
    </ProductionApplicationRegistration>
...
    <SandboxApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowE
xecutor">
        <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWork
FlowProcess/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Prope
rty>
    </SandboxApplicationRegistration>
...
</WorkFlowExtensions>

```

 Note that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

3. Go to the API Store Web interface, open **My Subscriptions** page, select an application and click the **Generate** button associated with the production key. It invokes the ApplicationRegistrationWorkflowProcess.bpel that is bundled with ApplicationRegistrationWorkflowProcess\_1.0.0.zip and creates a HumanTask instance that holds the execution of the BPEL process until some action is performed on it.
4. Note a message that appears saying that the request is successfully submitted if the BPEL was invoked correctly. For example,



The screenshot shows the 'Subscriptions' page of the API Store. At the top, there is a dropdown menu set to 'SampleApp1'. Below it, a section titled 'Applications With Subscriptions' shows a single entry 'SampleApp1'. Underneath this, a section titled 'Keys - Production' contains a message: 'Request to register this application has been sent.' This message is highlighted with a red border.

5. Log in to the Admin Dashboard Web application ([https://<Server\\_Host>:9443/admin-dashboard](https://<Server_Host>:9443/admin-dashboard)) and list all the tasks for application registrations. Click **Start** to start the Human Task and then change its

state. Once you approve the task, it resumes the BPEL process and completes the registration.

6. Go back to the **My Subscriptions** page on the API Store and view your application.

It shows the application access token, consumer key and consumer secret. For example,

### Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times to a single API, with different SLA levels.

#### Applications With Subscriptions

The screenshot shows the 'Applications With Subscriptions' page. A dropdown menu is open for 'SampleApp1'. On the right, there is a checkbox labeled 'Show Keys' which is checked. Below the dropdown, a section titled 'Keys - Production' is visible. It contains three fields: 'Access Token' (W3e7f\_43CxslcYC5jOSi58jga), 'Consumer Key' (1KkUVEPiYOCXBWir85frjh3sO54a), and 'Consumer Secret' (NDqep2XaMZl1dQlhCXp8Tz42B54a). There is also a 'Re-generate' button and a 'Token Validity' field set to 3600 seconds. To the right, there is a 'Allowed Domains' section with a dropdown menu containing 'ALL'.

After the registration request is approved, keys are generated by invoking the `APIKeyMgtSubscriber` service hosted in Key Manager nodes. Even when the request is approved, key generation can fail if this service becomes unavailable. To address such failures, you can configure to trigger key generation at a time Key Manager nodes become available again. Given below is the message used to invoke the BPEL process:

```
<applicationregistrationworkflowprocessrequest
  xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org"
  xmlns="http://workflow.application.apimgt.carbon.wso2.org">
  <applicationname>NewApp5</applicationname>
  <applicationtier>Unlimited</applicationtier>
  <applicationcallbackurl></applicationcallbackurl>
  <applicationdescription></applicationdescription>
  <tenantdomain>carbon.super</tenantdomain>
  <username>admin</username>

  <workflowexternalref>4a20749b-a10d-4fa5-819b-4fae5f57ffaf</workflowexternalref>

  <callbackurl>https://localhost:8243/services/WorkflowCallbackService</callbackurl>
  <keytype>PRODUCTION</keytype>
</applicationregistrationworkflowprocessrequest>
```

## Adding an API Subscription Workflow

This section explains how to attach a custom workflow to the API subscription operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflows executors.

Attaching a custom workflow to A PI subscription enables you to add throttling tiers to an API that consumers cannot choose at the time of subscribing. Only admins can set these tiers to APIs. It also allows you to r estrict API consumers to only subscribe to sandbox, and then go through an approval process to go to the next level of subscription.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Give the correct credentials in the `<bps_home>/repository/conf/epr` files.
- Change the credentials of the workflow configurations in the registry resource `_system/governance/apimgt/applicationdata/workflow-extensions.xml`.
- Point the database that has the API Manager user permissions to BPS.

- Share any LDAPs, if exist.
- Change the user credentials in <APIM\_HOME>/repository/conf/api\_manager.xml file.
- Change the .ht file of the relevant human task.
- Change the allowedRoles parameter in the <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file.

## Configuring the Business Process Server

1. Download WSO2 Business Process Server.
2. Set an offset of 2 to the default BPS port in <BPS\_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

```
<Offset>2</Offset>
```

 **Tip:** If you change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset.)
- Search and replace port 9445 in <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file with the new port (9443 + port offset.)

3. Copy the following from <APIM\_HOME>/business-processes/epr to <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder isn't there, please create it.
  - SubscriptionService.epr
  - SubscriptionCallbackService.epr
4. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).
5. Select **Add** under **Processes** menu and upload the <APIM\_HOME>/business-processes/subscription-creation/BPEL/SubscriptionApprovalWorkFlowProcess\_1.0.0.zip file to BPS. This is the business process archive file.

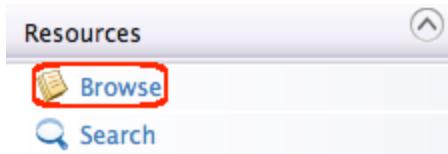


6. Select **Add** under the **Human Tasks** menu and upload <APIM\_HOME>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip to BPS. This is the human task archived file.

## Engaging the WS Workflow Executor in the API Manager

First, enable the API subscription workflow.

1. Log in to APIM admin console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resource**.



2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```
<WorkFlowExtensions>
...
<SubscriptionCreation
executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationWSWorkflowExecutor">
    <Property
name="serviceEndpoint">http://localhost:9765/services/SubscriptionApprovalWorkFlo
wProcess/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Prope
rty>
    </SubscriptionCreation>
...
</WorkFlowExtensions>
```

 **Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

- The application creation WS Workflow Executor is now engaged.
3. Go to the API Store Web interface and subscribe to an API.  
It invokes the API subscription process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.
  4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
  5. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>), list all the tasks for API subscription and approve the task. It resumes the BPEL process and completes the API subscription.
  6. Go back to the API Store and see that the user is now subscribed to the API.

Whenever a user tries to subscribe to an API, a request of the following format is sent to the workflow endpoint:

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wor="http://workfl
ow.subscription.apimgt.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <wor:createSubscription>
      <wor:apiName>sampleAPI</wor:apiName>
      <wor:apiVersion>1.0.0</wor:apiVersion>
      <wor:apiContext>/sample</wor:apiContext>
      <wor:apiProvider>admin</wor:apiProvider>
      <wor:subscriber>subscriber1</wor:subscriber>
      <wor:applicationName>application1</wor:applicationName>
      <wor:tierName>gold</wor:tierName>
      <wor:workflowExternalRef></wor:workflowExternalRef>
      <wor:callBackURL>?</wor:callBackURL>
    </wor:createSubscription>
  </soapenv:Body>
</soapenv:Envelope>

```

Elements of the above configuration are described below:

Element	Description
apiName	Name of the API to which subscription is requested.
apiVersion	Version of the API the user subscribes to.
apiContext	Context in which the requested API is to be accessed.
apiProvider	Provider of the API.
subscriber	Name of the user requesting subscription.
applicationName	Name of the application through which the user subscribes to the API.
tierName	Throttling tiers specified for the application.
workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	The URL to which the Workflow completion request is sent to by the workflow engine, at the time of workflow completion. This property is configured under the callBackURL property in the api-manager.xml.

## Adding a User Signup Workflow

This section explains how to attach a custom workflow to the user signup operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

 **Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.

- Change the credentials of the workflow configurations in the registry resource \_system/governance/apimgt/applicationdata/workflow-extensions.xml.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Change the user credentials in <APIM\_HOME>/repository/conf/api\_manager.xml file.
- Change the .ht file of the relevant human task.
- Change the allowedRoles parameter in the <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file.

## Configuring the Business Process Server

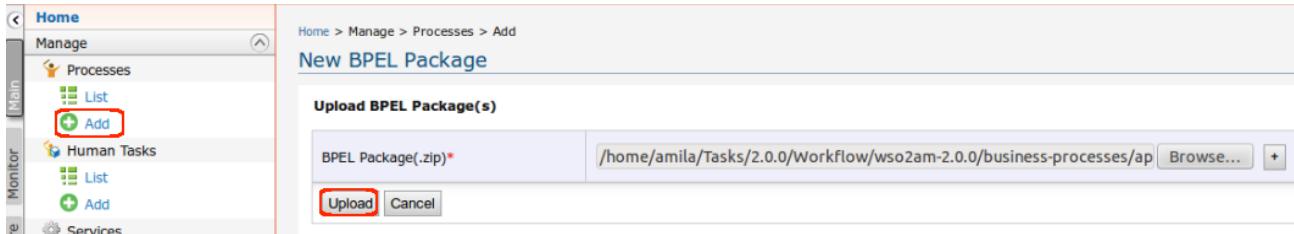
1. Download WSO2 Business Process Server.
2. Set an offset of 2 to the default BPS port in <BPS\_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

```
<Offset>2</Offset>
```

 **Tip:** If you **change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines** (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset.)
- Search and replace port 9445 in <AM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file with the new port (9443 + port offset.)

3. Copy the following from <APIM\_HOME>/business-processes/epr to <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder isn't there, please create it.
  - UserSignupService.epr
  - UserSignupProcess.epr
4. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).
5. Select **Add** under **Processes** menu and upload the <APIM\_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess\_1.0.0.zip file to BPS. This is the business process archive file.



6. Select **Add** under the **Human Tasks** menu and upload <APIM\_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip to BPS. This is the human task archived file.

## Engaging the WS Workflow Executor in the API Manager

First, enable the user signup workflow.

1. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** unde



## Resources .

2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```
<WorkFlowExtensions>
...
<UserSignUp
executor="org.wso2.carbon.apimgt.impl.workflow.UserSignUpWSWorkflowExecutor">
    <Property
name="serviceEndpoint">http://localhost:9765/services/UserSignupProcess/</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
</UserSignUp>
...
</WorkFlowExtensions>
```



**Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

3. Go to the API Store Web interface and sign up. It invokes the signup process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.
4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
5. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) and approve the user signup task. It resumes the BPEL process and completes the signup process.
6. Go back to the API Store and see that the user is now registered.

Whenever a user tries to sign up to the API Store, a request of the following format is sent to the workflow endpoint:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
    <soapenv:Header />
    <soapenv:Body>
        <wor:registerUser
            xmlns:wor="http://workflow.registeruser.apimgt.carbon.wso2.org">
            <wor:userName>sampleuser</wor:userName>
            <wor:tenantDomain>foo.com</wor:tenantDomain>

            <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExternalRef>

            <wor:callbackURL>https://localhost:8243/services/WorkflowCallbackService</wor:callbackURL>
        </wor:registerUser>
    </soapenv:Body>
</soapenv:Envelope>

```

Elements of the above configuration are described below:

Element	Description
userName	The user name requested by the user
tenantDomain	Domain to which the user belongs to
workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	The URL to which the workflow completion request is sent by the workflow engine, at the time of workflow completion. This property is configured under the "callbackURL" property in the api-manager.xml.

### Invoking the API Manager from the BPEL Engine

Once the workflow configurations are finalized at the BPEL, the call-back URL of the APIM, which is originally configured in the <APIM\_HOME>/repository/conf/api-manager.xml file and sent to the BPEL engine in the outflow will be called to progress the workflow. In APIM, the endpoint is available in both SOAP and REST variants as follows:

Type	URI
SOAP	<a href="https://localhost:8243/services/WorkflowCallbackService">https://localhost:8243/services/WorkflowCallbackService</a> WSDL Location : <a href="http://localhost:8280/services/WorkflowCallbackService?wsdl">http://localhost:8280/services/WorkflowCallbackService?wsdl</a>
REST	<a href="https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag">https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag</a>

Both the endpoints are secured via basic authentication. Therefore, when you invoke either endpoint, you need to include an authorization header with a base64-encoded value of the username and password with the request. E.g., Authorization: Basic <base64 encoded `username:password`> .

The endpoint expects the following list of parameters:

Parameter	Description	Mandatory
workflowReference	The unique identifier sent to the BPEL against which the workflow is tracked in API Manager	YES

status	The next status to which the workflow needs to be promoted to.	YES
description	Notes, that may need to be persisted against a particular workflow.	NO

A sample curl request for invoking the REST endpoint is as follows:

```
curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X POST
http://localhost:9763/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag -d
'workflowReference=b530be39-9174-43b3-acb3-2603a223b094&status=APPROVED&description=DESCRIPTION'
```

A sample SOAP request is given below:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:cal="http://callback.workflow.apimgt.carbon.wso2.org">
<soapenv:Header/>
<soapenv:Body>
<cal:resumeEvent>

<cal:workflowReference>b530be39-9174-43b3-acb3-2603a223b094</cal:workflowReference>
<cal:status>APPROVED</cal:status>
<cal:description>DESCRIPTION</cal:description>
</cal:resumeEvent>
</soapenv:Body>
</soapenv:Envelope>
```

## Customizing a Workflow Extension

Each workflow executor in the WSO2 API Manager is inherited from the `org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor` abstract class, which has two abstract methods:

- `execute`: contains the implementation of the workflow execution
- `complete`: contains the implementation of the workflow completion
- `getWorkflowType`: abstract method that returns the type of the workflow as a String
- `getWorkflowDetails(String workflowStatus)`: abstract method that returns a list of `WorkflowDTO` objects. This method is not used at the moment and it returns null for the time being.

To customize the default workflow extension, you override the `execute()` and `complete()` methods with your custom implementation. For example, the following class is a sample implementation of the Subscription Creation workflow. It returns an email to an address provided through the configuration on each subscription creation:

```
package org.wso2.sample.workflow;

import java.util.List;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import org.wso2.carbon.apimgt.api.APIManagementException;
```

```

import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO;
import org.wso2.carbon.apimgt.impl.dto.SubscriptionWorkflowDTO;
import org.wso2.carbon.apimgt.impl.dto.WorkflowDTO;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowConstants;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowException;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowStatus;

public class SubsCreationEmailSender extends WorkflowExecutor {
    private String adminEmail;
    private String emailAddress;
    private String emailPassword;

    @Override
    public List<WorkflowDTO> getWorkflowDetails(String arg0)
        throws WorkflowException {
        return null;
    }

    @Override
    public String getWorkflowType() {
        return WorkflowConstants.WF_TYPE_AM_SUBSCRIPTION_CREATION;
    }

    @Override
    public void execute(WorkflowDTO workflowDTO) throws WorkflowException{
        SubscriptionWorkflowDTO subsCreationWFDTO =
(SubscriptionWorkflowDTO)workflowDTO;

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props,
            new javax.mail.Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication(emailAddress,
                        emailPassword);
                }
            });
    }

    try {

        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(emailAddress));
        message.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(adminEmail));
        message.setSubject("Subscription Creation");
        message.setText("Subscription created for API " +
subsCreationWFDTO.getApiName() +
                    " using Application " +
subsCreationWFDTO.getApplicationName() +
                    " by user " + subsCreationWFDTO.getSubscriber()));

        Transport.send(message);
        System.out.println("Sent email to notify subscription creation");
    }
}

```

```

        //Call the execute method of the parent class. This will create a
reference for the
        //workflow execution in the database.
        super.execute(workflowDTO);
        //Set the workflow Status to APPROVED and Immediately complete the
workflow since we
        //are not waiting for an external party to complete this.
        workflowDTO.setStatus(WorkflowStatus.APPROVED);
        complete(workflowDTO);

    } catch (MessagingException e) {
        e.printStackTrace();
        throw new WorkflowException(e.getMessage());
    } catch (Exception e){
        e.printStackTrace();
        throw new WorkflowException(e.getMessage());
    }
}

@Override
public void complete(WorkflowDTO workflowDTO) throws WorkflowException{
    workflowDTO.setUpdatedTime(System.currentTimeMillis());
    super.complete(workflowDTO);
    ApiMgtDAO apiMgtDAO = new ApiMgtDAO();
    try {
        apiMgtDAO.updateSubscriptionStatus(
            Integer.parseInt(workflowDTO.getWorkflowReference()),
            APIConstants.SubscriptionStatus.UNBLOCKED);
    } catch (APIManagementException e) {
        throw new WorkflowException(
            "Could not complete subscription creation workflow", e);
    }
}
public String getAdminEmail() {
    return adminEmail;
}
public void setAdminEmail(String adminEmail) {
    this.adminEmail = adminEmail;
}
public String getEmailAddress() {
    return emailAddress;
}
public void setEmailAddress(String emailAddress) {
    this.emailAddress = emailAddress;
}
public String getEmailPassword() {
    return emailPassword;
}
public void setEmailPassword(String emailPassword) {

```

```

        this.emailPassword = emailPassword;
    }
}

```

Note the following regarding the above sample:

- The `execute()` method takes in a `WorkflowDTO` object (**SubscriptionWorkflowDTO** class) that contains information about the subscription that is being created.
- The `adminEmail`, `emailAddress` and `emailPassword` are private String variables with public getter and setter methods. The values for these variables are populated through the server configuration.
- After sending the email, a call is made to the super class's `execute()` method in order to create a reference entry in the database. This entry is generally used to look up the workflow when the workflow happens asynchronously (via a human approval).
- The `complete()` method contains the code to mark the subscription active. Until then, the subscription is in `ON_HOLD` state.
- In this sample, the `complete()` method is called immediately to make the subscription active instantly. If the completion of your workflow happens asynchronously, you must not call the `complete()` method from the `execute()` method.
- The `WorkflowException` is thrown to roll back the subscription in case of a failure.

After the implementation of the class is done, follow the steps below to implement the new workflow extension in the API Manager:

1. Compile the class and export it as a JAR file. Make sure you have the following JARs in the classpath before compilation.
  - <AM\_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.impl\_1.2.1.jar
  - <AM\_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.api\_1.2.1.jar
  - javax.mail.jar: see <https://java.net/projects/javamail/pages/Home> to download the JAR
2. After exporting the JAR, copy it to <AM\_HOME>/repository/components/lib.
3. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **R e s o u r c e s .**



4. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication). For example:

```

<WorkFlowExtensions>
...
    <!--SubscriptionCreation
executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationSimpleWorkflow
Executor"-->
    <SubscriptionCreation
executor="org.wso2.sample.workflow.SubsCreationEmailSender">
        <Property name="adminEmail">to_user@email.com</Property>
        <Property name="emailAddress">from _user@email.com</Property>
        <Property name="emailPassword">from_user_password</Property>
    </SubscriptionCreation>
...
</WorkFlowExtensions>

```

Note that the `adminEmail`, `emailAddress` and `emailPassword` properties will be assigned to the appropriate variables defined in the class through the public setter methods of those variables.

**⚠** If you use the same or similar sample to return an email, you must remove the `org.jaggeryjs.hostobj
ects.email_0.9.0.ALPHA4_wso2v1.jar` file from `<AM_HOME>/repository/components/plugins` directory. Removing it results in a `ClassNotFoundException` thrown at server startup, but it does not affect the server's functionality.

## Configuring Workflows for Tenants

Using the API Manager, you can configure custom workflows that get invoked at the event of a user signup, application creation, registration, subscription etc. You do these configurations in the `api-manager.xml` as described in the previous sections.

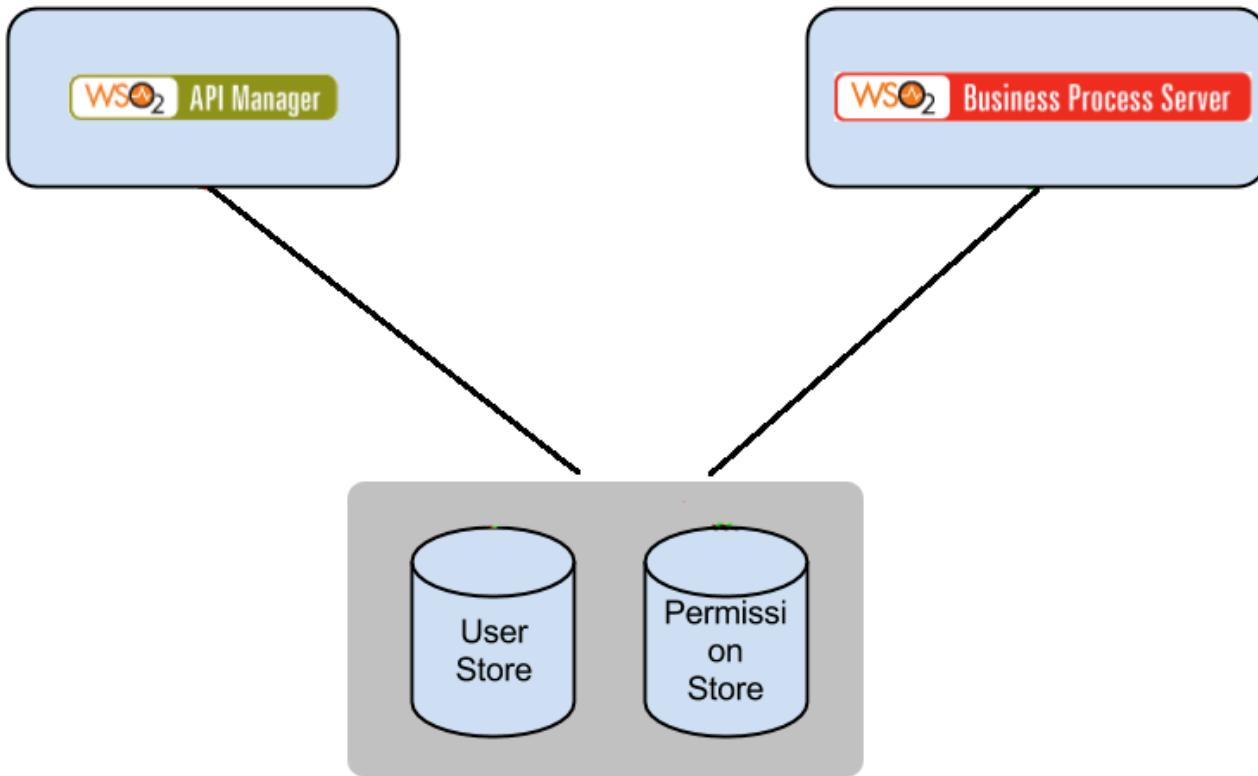
However, in a multi-tenant API Manager setup, not all tenants have access to the file system and not all tenants want to use the same workflow that the super admin has configured in the `api-manager.xml` file. For example, different departments in an enterprise can act as different tenants using the same API Manager instance and they can have different workflows. Also, an enterprise can combine WSO2 API Manager and WSO2 Business Process Server (BPS) to provide API Management As a Service to the clients. In this case, each client is a separate enterprise represented by a separate tenant. In both cases, the authority to approve business operations (workflows) resides within a tenant's space.

To allow different tenants to define their own custom workflows without editing configuration files, the API Manager provides configuration in tenant-specific locations in the registry, which you can access through the UI.

The topics below explain how to deploy a BPEL/human task using WSO2 BPS and how to point them to services deployed in the tenant spaces in the API Manager.

### Deploying a BPEL and a HumanTask for a tenant

Only the users registered in the BPS can deploy BPELs and human tasks in it. Registration adds you to the user store in the BPS. In this guide, the API Manager and BPS use the same user store and all the users present in the BPS are visible to the API Manager as well. This is depicted by the diagram below:



**Figure:** API Manager and BPS share the same user and permission store

**!** If you are using WSO2 BPS 3.2.0, please copy the `<APIM_HOME>/repository/components/patches/patch0009` folder to the `<BPS_HOME>/repository/components/patches` folder and restart the BPS server for the patch to be applied. This patch has a fix to a bug that causes the workflow configurations to fail in multi-tenant environments.

This patch is built into the BPS version 3.5.0 onwards.

Follow the steps below to deploy a BPEL and a human task for a tenant in the API Manager:

#### **Sharing the user/permission stores with the BPS and API Manager**

1. Create a database for the shared user and permission store as follows:

```
mysql> create database workflow_ustore;
Query OK, 1 row affected (0.00 sec)
```

**✓** Make sure you copy the database driver (in this case, mysql driver) to the `/repository/components/lib` folder before starting each server.

2. Run the `<APIM_HOME>/dbscripts/mysql.sql` script (the script may vary depending on your database type) on the database to create the required tables.
3. Open the `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` and create a datasource pointing to the newly created database. For example,

```

<datasource>
    <name>USTORE</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/ustore</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>

<url>jdbc:mysql://127.0.0.1:3306/workflow_ustore?autoReconnect=true&relaxAutoCommit=true</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

4. Repeat step 3 in the BPS as well.
5. Point the datasource name in <APIM\_HOME>/repository/conf/user-mgt.xml to the new datasource. (note that the user store is configured using the <UserStoreManager> element).

 If you already have a user store such as the IDAP in your environment, you can point to it from the user-mgt.xml file, instead of the user store that we created in step1.

In the following example, the same JDBC user store (that is shared by both the API Manager and the BPS) is used as the permission store as well:

```

<Configuration>
    <AddAdmin>true</AddAdmin>
    <AdminRole>admin</AdminRole>
    <AdminUser>
        <UserName>admin</UserName>
        <Password>admin</Password>
    </AdminUser>
    <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role sees the registry root -->
    <Property name="dataSource">jdbc/ustore</Property>
</Configuration>

```

6. Repeat step 5 in the BPS as well.

#### ***Sharing the data in the registry with the BPS and API Manager***

To deploy BPELs in an API Manager tenant space, the tenant space should be accessible by both the BPS and API Manager, and certain tenant-specific data such as key stores needs to be shared with both products. Follow the steps below to create a registry mount to share the data stored in the registry:

1. Create a separate database for the registry:

```
mysql> create database workflow_regdb;
Query OK, 1 row affected (0.00 sec)
```

2. Run the <APIM\_HOME>/dbscripts/mysql.sql script (the script may vary depending on your database type) on the database to create the required tables.
3. Create a new datasource in <APIM\_HOME>/repository/conf/datasources/master-datasources.xml as done before:

```
<datasource>
    <name>REG_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/regdb</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>

            <url>jdbc:mysql://127.0.0.1:3306/workflow_regdb?autoReconnect=true&relaxAutoCommit=true</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

4. Add the following entries to <APIM\_HOME>/repository/conf/registry.xml:

```

<dbConfig name="sharedregistry">
    <dataSource>jdbc/regdb</dataSource>
</dbConfig>

<remoteInstance url="https://localhost:9443/registry">
    <id>mount</id>
    <dbConfig>sharedregistry</dbConfig>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>
<!-- This defines the mount configuration to be used with the remote instance
and the target path for the mount --&gt;
&lt;mount path="/_system/config" overwrite="true"&gt;
    &lt;instanceId&gt;mount&lt;/instanceId&gt;
    &lt;targetPath&gt;/_system/nodes&lt;/targetPath&gt;
&lt;/mount&gt;
&lt;mount path="/_system/governance" overwrite="true"&gt;
    &lt;instanceId&gt;mount&lt;/instanceId&gt;
    &lt;targetPath&gt;/_system/governance&lt;/targetPath&gt;
&lt;/mount&gt;
</pre>

```

5. Repeat the above three steps in the BPS as well.

### **Creating a BPEL**

In this section, you create a BPEL that has service endpoints pointing to services hosted in the tenant's space. This example uses the [Application Creation](#) workflow.

1. Set a port offset of 2 to the BPS using the `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents any port conflicts when you start more than one WSO2 products on the same server.
2. Log in to the API Manager's management console (<https://localhost:9443/carbon>) and create a tenant using the **Configure** -> **Multitenancy** menu.

The screenshot shows the WSO2 API Manager's management console interface. The left sidebar has tabs for Main, Monitor, Configure, and Tools. Under the Configure tab, there is a 'Multitenancy' section with two options: 'Add New Tenant' and 'View Tenants'. The 'Add New Tenant' option is highlighted with a red box. The main content area is titled 'Register A New Organization'. It contains fields for 'Domain' (set to 'acme.com'), 'Usage Plan Information' (set to 'Demo'), 'Tenant Admin' (with fields for First Name, Last Name, Admin Username, Admin Password, Admin Password (Repeat), and Email), and a 'Contact Details' section (with an Email field set to 'ss@cc.com'). A 'Save' button is at the bottom.

3. Create a copy of the BPEL located in <APIM\_HOME>/business-processes/application-creation/BPEL.
4. Extract the contents of the new BPEL archive.
5. Copy ApplicationService.epr and ApplicationCallbackService.epr from <APIM\_HOME>/business-processes/epr folder to the folder extracted before. Then, rename the two files as ApplicationService-Tenant.epr and ApplicationCallbackService-Tenant.epr respectively.
6. Open ApplicationService-Tenant.epr and change the wsa:Address to http://localhost:9765/services/t/<tenant domain>/ApplicationService.
7. Point the deploy.xml file of the extracted folder to the new .epr files provided in the BPEL archive. For example,

```

<invoke partnerLink="AAPL">
    <service name="applications:ApplicationService" port="ApplicationPort">
        <endpoint xmlns="http://wso2.org/bps/bpel/endpoint/config"
        endpointReference="ApplicationService-Tenant.epr"></endpoint>
    </service>
</invoke>

<invoke partnerLink="CBPL">
    <service
    name="callback.workflow.apimgt.carbon.wso2.org:WorkflowCallbackService"
    port="WorkflowCallbackServiceHttpsSoap11Endpoint">
        <endpoint xmlns="http://wso2.org/bps/bpel/endpoint/config"
        endpointReference="ApplicationCallbackService-Tenant.epr"></endpoint>
    </service>
</invoke>

```

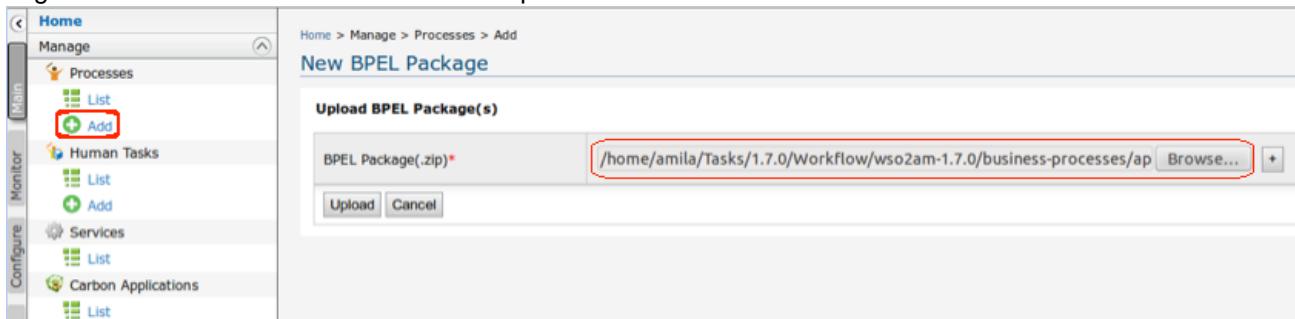
8. Zip the content and create a BPEL archive in the following format:

```

ApplicationApprovalWorkFlowProcess_1.0.0-Tenant.zip
|_ApplicationApprovalWorkFlowProcess.bpel
|_ApplicationApprovalWorkFlowProcessArtifacts.wsdl
|_ApplicationCallbackService-Tenant.epr
|_ApplicationService-Tenant.epr
|_ApplicationsApprovalTaskService.wsdl
|_SecuredService-service.xml
|_WorkflowCallbackService.wsdl
|_deploy.xml

```

9. Log into the BPS as the tenant admin and upload the BPEL.



### ***Creating a human task***

Similar to creating a BPEL, create a HumanTask that has service endpoints pointing to services hosted in the tenant's space.

1. Create a copy of the HumanTask archive in <APIM\_HOME>/business-processes/application-creation/HumanTask and extract its contents.
2. Edit the SOAP service port-bindings in ApplicationApprovalTaskService.wsdl. For example,

```
<wsdl:service name="ApplicationService">
  <wsdl:port name="ApplicationPort" binding="tns:ApplicationSoapBinding">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="ApplicationReminderService">
  <wsdl:port name="ApplicationReminderPort"
binding="tns:ApplicationSoapBindingReminder">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationReminderService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="ApplicationServiceCB">
  <wsdl:port name="ApplicationPortCB" binding="tns:ApplicationSoapBindingCB">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationServiceCB" />
  </wsdl:port>
</wsdl:service>
```

3. Create the HumanTask archive by zipping all the extracted files.
4. Log into the BPS as the tenant admin and upload the HumanTask.
5. Log into the API Manager's management console as the tenant admin and select **Resources > Browse** menu.
6. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` in the registry and change the **service endpoint as a tenant-aware service URL** (e.g., `http://localhost:9765/services/t/<tenant_domain>/ApplicationApprovalWorkFlowProcess`). Also set the **credentials**

als as the **tenant admin's credentials** of the ApplicationCreationWSWorkflowExecutor file. For example,

The screenshot shows the WSO2 API Manager Governance interface. On the left, there is a sidebar with various management options like Identity Providers, Applications, Service Bus, and Resources. Under the 'Resources' section, the 'Browse' option is highlighted with a red box. The main content area is titled 'Browse' and shows the XML configuration for the 'ApplicationCreationWSWorkflowExecutor'. The XML code includes properties for service endpoint, username, password, and callback URL. At the bottom of the editor, there are 'Save Content' and 'Cancel' buttons.

**⚠️** Be sure to disable the SimpleWorkflowExecutor and enable the ApplicationCreationWSWorkflowExecutor.

## Testing the workflow

You have now completed configuring the Application Creation workflow for a tenant. Whenever a tenant user logs in to the tenant store and create an application, the workflow will be invoked. You log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) as the tenant admin and browse **Application Creation** menu to see all approval tasks have been created for newly created applications.

The screenshot shows the WSO2 Admin Dashboard. The left sidebar has a 'Tasks' section with 'User Creation', 'Application Creation' (which is selected and highlighted in blue), 'Subscriptions Creation', 'Application Registration', and 'Settings'. The main content area is titled 'Approval Tasks' and shows a table with one row. The table columns are 'ID', 'Description', 'Status', 'Created On', and 'Action'. The single task listed is '451 Approve application [newApplication] creation request from application creator - admin with throttling tier - Unlimited' with status 'RESERVED' and created on '2014-12-12 - 15:29:27.815+05:30'. There is a 'Start' button next to the last column.

## Configuring Workflows in a Cluster

If you are working in a clustered API Manager setup with the API Store, Publisher, Gateway and Key Manager in separate servers, do the workflow configurations that are discussed in the previous topics in the **API Store node**. In addition, do the following configurations.

In this guide, you access the Admin Dashboard (<https://<Server Host>:9443/admin-dashboard>) Web application using the same node as the API Publisher. This is recommended because workflow management is an administrative task and is meant to reside within a private network as the Publisher. Typically, the Admin

Dashboard from the same user store as the API Manager. Therefore, you can use the Admin Dashboard residing in the Publisher node instead of having it separately. This eliminates the need for a dedicated workflow management node. You need a dedicated node if the Admin Dashboard users reside in a separate user store.

1. If you want to change the user roles that can access the Admin Dashboard, open the <APIM\_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json file that is in the node from where you access the Admin Dashboard (the API Publisher node in this example) and change its Allowed Roles parameter. You can add multiple user roles as a comma-separated list.
2. By default, workflow related configuration files have the port of the Business Process Server with an offset of 2. If you set up the BPS with a different port offset, change the workflow server URLs in the site.json file accordingly.
3. Point the <Address> sub element of the <endpoint> element to the API Store node in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml file of the API Store node.

```
<endpoint>
  <address
    uri="https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag" format="rest"/>
</endpoint>
```

4. Add the IP address and the port of the API Store to the <Address> element of the .epr file of the workflow that you configure. You can find the .epr file by the name of the workflow in the <APIM\_HOME>/business-processes/epr folder.
5. Go to the <APIM\_HOME>/business-processes/<workflow\_name>/BPEL folder and unzip the file that is there by the name of the workflow. For example, <APIM\_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess\_1.0.0.zip.
6. Go inside the unzipped folder and do the following:

Action	Example
Open the ApprovalTask WSDL file and point the address elements of the server where the BPEL runs.	<p>In the &lt;APIM_HOME&gt;/business-processes/user-signup/BPEL/UserSignup.epr:</p> <pre>&lt;wsdl:service name="UserApprovalService"&gt;   &lt;wsdl:port name="UserApprovalPort" binding="tns:UserApprovalBinding"&gt;     &lt;soap:address location="http://localhost:9765/services/UserApprovalService"&gt;       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;           &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;             &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;               &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                 &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                   &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                     &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                           &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                             &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                               &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                 &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                   &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                     &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                           &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                             &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                               &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                 &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                   &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                     &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                           &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                             &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                               &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                 &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                   &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                     &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                           &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                             &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                               &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                                 &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                                   &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                                     &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;                                                                                       &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"&gt;                                                                                         &lt;soap:address location="http://localhost:9765/services/UserApprovalServiceCB"&gt;               &lt;/wsdl:port&gt;             &lt;/wsdl:port&gt;           &lt;/wsdl:port&gt;         &lt;/wsdl:port&gt;       &lt;/wsdl:port&gt;     &lt;/wsdl:port&gt;   &lt;/wsdl:port&gt; &lt;/wsdl:service&gt;</pre> <p><b>Note</b> that all workflow process services of the BPS run on port 9765 because you have specified the port number in the &lt;Address&gt; element of the .epr file.</p>

Open the CallbackService WSDL file and point the address elements to the API Store node in NIO port.

In the <APIM\_HOME>/business-processes/user-signup/BPEL/UserSignup.wsdl file:

```
<wsdl:service name="WorkflowCallbackService">
    <wsdl:port name="WorkflowCallbackServiceHttpsSoap11Endpoint"
binding="ns:WorkflowCallbackServiceSoap11Binding">
        <soap:address
location="https://localhost:8243/services/WorkflowCallbackService."
        </wsdl:port>
        <wsdl:port name="WorkflowCallbackServiceHttpSoap11Endpoint"
binding="ns:WorkflowCallbackServiceSoap11Binding">
            <soap:address
location="http://localhost:8280/services/WorkflowCallbackService.W
            </wsdl:port>
            <wsdl:port name="WorkflowCallbackServiceHttpsSoap12Endpoint"
binding="ns:WorkflowCallbackServiceSoap12Binding">
                <soap12:address
location="https://localhost:8243/services/WorkflowCallbackService."
                </wsdl:port>
                <wsdl:port name="WorkflowCallbackServiceHttpSoap12Endpoint"
binding="ns:WorkflowCallbackServiceSoap12Binding">
                    <soap12:address
location="http://localhost:8280/services/WorkflowCallbackService.W
                    </wsdl:port>
                    <wsdl:port name="WorkflowCallbackServiceHttpsEndpoint" bind
                        <http:address
location="https://localhost:8243/services/WorkflowCallbackService."
                        </wsdl:port>
                        <wsdl:port name="WorkflowCallbackServiceHttpEndpoint" bind
                            <http:address
location="http://localhost:8280/services/WorkflowCallbackService.W
                            </wsdl:port>
</wsdl:service>
```

7. Go to the <APIM\_HOME>/business-processes/<workflow\_name>/HumanTask folder and unzip the file that is there by the name of the workflow. For example, <APIM\_HOME>/business-processes/user-signup/HumanTask/**UserApprovalTask-1.0.0.zip**.
8. Go inside the unzipped folder and do the following:

Action	Example
If you changed the default admin role, open the ApprovalTask HT file and apply the changes there.	<p>Change the admin instances in &lt;APIM_HOME&gt;/business-processes/user-signup,/UserApprovalTask-1.0.0/<b>UserApprovalTask.ht</b> file. Here's an example, assume new admin role is apimadmin.</p> <pre>&lt;htd:peopleAssignments&gt;     &lt;htd:potentialOwners&gt;         &lt;htd:from logicalPeopleGroup="admin"&gt;             &lt;htd:argument name="role"&gt;apimadmin&lt;/htd:argument&gt;         &lt;/htd:from&gt;     &lt;/htd:potentialOwners&gt; &lt;/htd:peopleAssignments&gt;</pre>

Open the ApprovalTask WS DL file and point the two address elements to the Business Process Server node.

In the <APIM\_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask.wsdl file:

```
<wsdl:service name="UserApprovalService">
    <wsdl:port name="UserApprovalPort"
binding="tns:UserApprovalBinding">
        <soap:address
location="http://localhost:9765/services/UserApprovalService" />
    </wsdl:port>
</wsdl:service>
<wsdl:service name="UserApprovalServiceCB">
    <wsdl:port name="UserApprovalPortCB"
binding="tns:UserApprovalBindingCB">
        <soap:address
location="http://localhost:9765/services/UserApprovalServiceCB" />
    </wsdl:port>
</wsdl:service>
```

 **Note** that all workflow process services of the BPS run on port 9765 because you changed the default port (9763) with an offset of 2.

## Changing the Default User Role in Workflows

The default user role in the workflow configuration files is the admin role. If you change this to something else, you need to change the following files:

1. Change the credentials in the .epr files of the <BPS\_HOME>/repository/conf/epr folder.
2. Change the credentials in work-flow configurations in API Manager Registry (\_system/governance/apimgt/applicationdata/workflow-extensions.xml.)
3. Point the same database that has the permissions used by the API Manager to the BPS.
4. Share the LDAP, if it exists.
5. Change the credentials in <APIM\_HOME>/repository/conf/api\_manager.xml file.
6. If you change the default user role,
  - a. change the .ht file of the relevant human task
  - b. change the allowedRoles attribute in <APIM\_HOME>/repository/deployment/server/jaggery\_apps/admin-dashboard/site/conf/site.json file

## Adding new Throttling Tiers

API Manager admins can add new throttling tiers and define extra properties to throttling tiers using the management console as discussed below. For a description of throttling tiers, see [API-level throttling](#).

1. Log in to the API Manager's Management Console and select **Browse** under **Resources** menu.
2. Select the file: /\_system/governance/apimgt/applicationdata/tiers.xml.

**Browse**

The screenshot shows the 'Browse' interface of the WSO2 API Manager. At the top, there's a 'Location' input field with a '/'. To its right is a 'Go' button. Below this is a navigation bar with 'Tree view' and 'Detail view' buttons, where 'Tree view' is selected. The main area displays a hierarchical tree structure under 'Root'. The nodes are: /, \_system, config, governance, apimgt, applicationdata, provider, tiers.xml (which is highlighted with a red box), event, and permission.

3. In the **Contents** panel, click **Edit as text** link and the throttling policy opens.
4. You can add a new policy configuration by editing the XML code. For example, we have added a new tier called `Platinum` by including the following XML code block soon after the `<throttle:MediatorThrottle Assertion>` element.

**Tier DisplayName :** You can add this **optional** attribute to each throttle ID of `tiers.xml` file in order to decouple the throttle policy name defined in `tiers.xml` from the tier name showing in APIPublisher/Store UIs. That is, a user can add a different throttle display name to appear in APIPublisher/Store UIs without changing the throttle ID policy name. The configuration below has a `displayName` as `platino` for the throttle value `platinum`. This value is displayed in APIPublisher/Store apps.

**Tier Attributes :** In the configuration below, there's a commented out XML section starting from the XML tag `<throttle:Attributes>`. You can use it to define additional attributes related to each throttling tier definition. For example, if the throttling tier `Platinum` has attributes called `PaymentPlan` and `Availability`, first uncomment the `<throttle:Attributes>` section and then define the new attributes as follows:

```

<wsp:Policy>
  <throttle:ID throttle:type="ROLE"
  throttle:displayName="platino">Platinum</throttle:ID>
    <wsp:Policy>
      <throttle:Control>
        <wsp:Policy>
          <throttle:MaximumCount>50</throttle:MaximumCount>
          <throttle:UnitTime>60000</throttle:UnitTime>
          <!--It's possible to define tier level attributes as below for
each tier level.For eg:Payment Plan for a tier-->
          <wsp:Policy>
            <throttle:Attributes>
              <!--throttle:Attribute1>xxxx</throttle:Attribute1-->
              <!--throttle:Attribute2>xxxx</throttle:Attribute2-->
              <throttle:PaymentPlan>monthly</throttle:PaymentPlan>
              <throttle:Availability>FullTime</throttle:Availability>
            </throttle:Attributes>
          </wsp:Policy>
        </wsp:Policy>
      </throttle:Control>
    </wsp:Policy>
  </wsp:Policy>

```

- After the edits, click **Save Content**. Your new throttling policy (Platinum) is now successfully saved in the Repository used by WSO2 API Manager. You can view this new throttle tier available for selection when creating a new API through the API Publisher.

## Adding a Reverse Proxy Server

A reverse proxy server retrieves information from a server and sends it to a client as though the information originated from the reverse proxy sever rather than the actual server. You can use a reverse proxy server to block access to selected applications in a server. For example, this is useful when you want to expose the token API in such a way that the clients can authenticate against OAuth2 using the same port that their API's are on.

The API Manager comes with two Web applications as the Publisher and Store. You can route the requests that come to them through a proxy server by editing the `<AM_HOME>/repository/deployment/server/jaggeryapps/store(/publisher)/site/conf/site.json` file. For example, to use a reverse proxy server for the API Store, edit the `<AM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file with the context and request URL as shown below.

```

"context" : "/public/store",
"request_url": "https://localhost/public/store/",

```

If you set up the reverse proxy server correctly, when you access the URL <https://localhost/public/store>, you will be directed to the API Store.

To do the same for the API Publisher, edit the `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file.

Also note that if you want to change all the default API Manager ports, you do so by editing the `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` file.

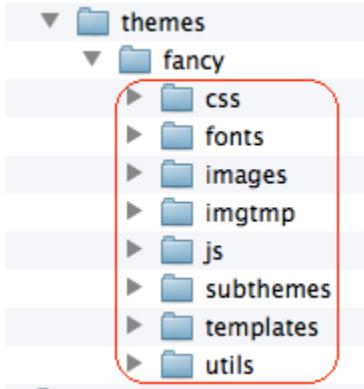
## Adding a new API Store Theme

A **theme** consists of UI elements such as logos, images, copyrights messages, landing page text, background colors etc. WSO2 API Store comes with a default theme.

## The folder structure of the API Store themes

The default theme of the API Store is called **Fancy**. You find it inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/fancy` folder. If you do not have access to the file system, download the default theme from [here](#).

The easiest way to create a new theme is to copy the files of an existing theme to a folder by the name of your new theme, and do the modifications you want to the files inside it. All themes have the same folder structure as shown below:



You can add a new theme as a main theme or a sub theme.

- **A main theme** is saved inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` folder
- **A sub theme** is saved inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/<folder of the main theme>/subthemes` folder.

Because a sub theme is saved inside a main theme, it needs to contain only the files that are different from the main theme. Any file you add inside the sub theme will override the corresponding files in the main theme. The rest of the files will be inherited from the main theme.

Let's see how to create a new theme and set it to the API Store:

- Writing a sub theme of the main theme
- Setting the new theme as the default theme
- Adding the new theme to the Themes menu

### ***Writing a sub theme of the main theme***

Because a main theme already has most of the UIs and the syntax and logic of Jaggery code defined, in a typical scenario, you do not have to implement a theme from scratch. Rather, you just add in your edits as a sub theme of the existing main theme as given below:

1. Download the default main theme [from here](#), unzip it and rename the folder according to the name of your new theme (e.g., `ancient`). Let's call this folder the `<THEME_HOME>`.
2. To change the logo of the API Store, replace the `logo.png` file inside the `<THEME_HOME>/images` folder with [this logo](#) (or anything else of your choice.)
3. To change the copyrights note in the footer, open the `<THEME_HOME>/templates/page/base/template.jag` file using a text editor, search for the word "Copyright" and change the text. For example, let's add our company name as "copyright", "&copy; Copyright 2011 &ndash; 2014 **My Company**."
4. To change the header's background color, open the `<THEME_HOME>/css/styles-layout.css` file using a text editor and add the following CSS rule to the end of the file. It changes the header color.

```
.header{
    background:#7e3330;
}
```

- As you plan to upload this as a sub theme of the default main theme, delete all the files in your <THEME\_HOME> folder except the ones that you edited. The rest of the files will be automatically applied from the main theme.

#### **Setting the new theme as the default theme**

You can set your new theme as the default theme in two ways:

- Saving directly in the file system
- Uploading through the Admin Dashboard

#### **Saving directly in the file system**

If you have access to the file system, do the following:

- Save the <THEME\_HOME> folder inside the <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/fancy/subthemes folder. This will make your new theme a sub theme of fancy.
- Open the <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and add the following code to it. It specifies the base theme as fancy, which is overridden by the sub theme ancient.

```
"theme" : {
    "base" : "fancy",
    "subtheme" : "ancient"
}
```

- Open the API Store and note the new theme applied to it.

The screenshot shows the WSO2 API Manager Admin Dashboard interface. At the top, there is a navigation bar with tabs: 'My API STORE' (selected), 'APIs' (highlighted in blue), 'Prototyped APIs', 'My Applications', and 'My Subscriptions'. Below the navigation bar, there is a search bar labeled 'Search API'. On the left, a sidebar titled 'Recently Added' shows no recent activity. The main content area is titled 'APIs' and contains the message 'No APIs published yet'.

#### **Uploading through the Admin Dashboard**

If you do not have access to the file system, you can upload the theme through the Admin Dashboard Web application as shown below:

- Go inside the <THEME\_HOME> folder, select all the folders inside it and right click to archive all the selected files and folders together. Then rename the archive files to `ancient.zip`.
- Log in to WSO2 Admin Dashboard Web application using the URL `https://<Server>`

**Host>:9443/admin-dashboard**

For example, if you are a WSO2 Cloud user and want to upload a new theme, log in to the URL [api.cloud.wso2.com/admin-dashboard](http://api.cloud.wso2.com/admin-dashboard) with the user name as email@domain with the @ in the email replaced by a dot (e.g., john.gmail.com@MyCompany).

- Click the **Upload Tenant Theme** menu and upload your zip file.

The screenshot shows the WSO2 Admin Dashboard with a dark header bar containing the text "WSO2 ADMIN DASHBOARD". Below it is a sidebar titled "Tasks" with several options: "User Creation" (highlighted in light blue), "Application Creation", "Subscriptions Creation", "Application Registration", "Settings", and "Upload Tenant Theme" (which has a red border around it). To the right of the sidebar is the main content area. The title "Upload Tenant Theme" is at the top. A sub-header "Tasks / User Creation" is visible above the main form. A message box contains the text: "The theme should be a zip file containing css and images which is compliant to API Manager theme format". Below this, there is a label "Please select the theme" followed by a "Choose File" button with the path "ancient.zip". At the bottom right of the form is a large blue "Upload" button.

- Open the API Store and note the new theme applied to it.

The screenshot shows the "My API STORE" interface. At the top, there is a navigation bar with tabs: "APIs" (highlighted in light blue), "Prototyped APIs", "My Applications", and "My Subscriptions". Below the navigation bar is a search bar labeled "Search API". The main content area is divided into two sections: "Recently Added" (on the left) and "APIs" (on the right). The "APIs" section displays the message "No APIs published yet".

#### ***Adding the new theme to the Themes menu***

Once you are done modifying the new theme, add it to the **Themes** menu in the API Store along with a thumbnail image as follows:

- Open the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/fancy/templates/user/login/template.jag` file and find the HTML table that defines the theme thumbnails.
- Add a new row under the `<table>` element with the following code. It adds `thumb-ancient.png` as the thumbnail image of our theme. Be sure save the image in the `...fancy/images` folder.

```

<td>
    <div class="thumbnail <% if(jagg.getUserTheme().base == "fancy" &&
jagg.getUserTheme().subtheme == "ancient") { %>currentTheme<% } %>">
        <a data-theme="fancy" data-subtheme="ancient" class="badge themeLabel"
onclick="applyTheme(this)">
            " />
            <br /><div class="themeName">Ancient</div>
        </a>
    </div>
</td>

```

## Working with Security

After you install the APIM, it is recommended to change the default security settings according to the requirements of your production environment. As the APIM is built on top of the WSO2 Carbon platform, some security configurations are inherited from WSO2 Carbon.

The following topics explain the WSO2 Carbon platform-based, and product-specific configurations:

- [WSO2 Carbon platform-based security configurations](#)
- [APIM-specific security configurations](#)

### WSO2 Carbon platform-based security configurations

The following security configurations are common to all WSO2 products that are built on top of the WSO2 Carbon platform.

Configuration	Description
Configuring transport-level security	<p>WSO2 products support a variety of transports that make them capable of receiving and sending messages over a multitude of transport and application-level protocols. By default, all WSO2 products come with the HTTP transport. The transport receiver implementation of the HTTP transport is available in Carbon. The transport sender implementation comes from the Tomcat HTTP connector, which is configured in the <code>&lt;APIM_HOME&gt;/repository/conf/tomcat/catalina-server.xml</code> file.</p> <p>For more information on securing the HTTP transport, see <a href="#">Configuring transport level security</a> in the WSO2 Carbon documentation.</p>
Configuring keystores	<p>A keystore is a repository that stores artifacts such as cryptographic keys and certificates. These artifacts are used for encrypting sensitive information and establishing trust between your server and outside parties that connect to your server.</p> <p>All WSO2 products come with a default keystore (<code>wso2carbon.jks</code>). In a production environment, it is recommended to replace it with one or more keystores.</p> <p>See the following in the WSO2 Carbon documentation:</p> <ul style="list-style-type: none"> <li>• <a href="#">How public key encryption and keystores are used.</a></li> <li>• <a href="#">How to create new keystores and replace the default one.</a></li> <li>• <a href="#">How configuration files should be updated to use the relevant keystore for different purposes.</a></li> </ul>

Securing sensitive passwords	<p>As a secure vault implementation is available in all WSO2 products, you can encrypt the sensitive data such as passwords in configuration files using the Cipher tool.</p> <p>See the following in the WSO2 Carbon documentation:</p> <ul style="list-style-type: none"> <li>• <a href="#">How the secure vault is implemented in WSO2 products.</a></li> <li>• <a href="#">How to encrypt passwords using the Cipher tool.</a></li> <li>• <a href="#">How to resolve encrypted passwords.</a></li> </ul> <p>Also see how to encrypt secure endpoint passwords.</p>
Enabling JAVA security manager	See <a href="#">Enabling JAVA security manager</a> in the WSO2 Carbon documentation on how to prevent untrusted code from manipulating your system.

### APIM-specific security configurations

See the following topics:

- [Passing Enduser Attributes to the Backend Using JWT](#)
- [Encrypting Secure Endpoint Passwords](#)
- [Maintaining Logins and passwords](#)
- [Saving Access Tokens in Separate Tables](#)
- [Configuring WSO2 Identity Server as the Key Manager](#)
- [Configuring a Third-Party Key Manager](#)
- [Block Subscription to an API](#)
- [Pass a Custom Authorization Token to the Backend](#)

## Passing Enduser Attributes to the Backend Using JWT

**JSON Web Token (JWT)** is used to represent claims that are transferred between two parties such as the enduser and the backend.

A claim is an attribute of the user that is mapped to the underlying user store. It is encoded as a JavaScript Object Notation (JSON) object that is used as the payload of a JSON Web Signature (JWS) structure, or as the plain text of a JSON Web Encryption (JWE) structure. This enables claims to be digitally signed.

A set of claims is called a dialect (e.g., <http://wso2.org/claims>). The general format of a JWT is {token info}.{claims list}.{signature}. The API implementation uses information such as logging, content filtering and authentication/authorization that is stored in this token. The token is Base64-encoded and sent to the API implementation in a HTTP header variable. The JWT is self-contained and is divided into three parts as the header, the payload and the signature. For more information on JWT, see <http://openid.net/specs/draft-jones-json-web-token-07.html#anchor3>.

To authenticate endusers, the API Manager passes attributes of the API invoker to the backend API implementation using JWT. In most production deployments, service calls go through the API Manager or a proxy service. If you enable JWT generation in the API Manager, each API request will carry a JWT to the back-end service. When the request goes through the API manager, the JWT is appended as a transport header to the outgoing message. The back-end service fetches the JWT and retrieves the required information about the user, application, or token.

An example of a JWT is given below:

```
{
    "typ": "JWT",
    "alg": "NONE"
} {
    "iss": "wso2.org/products/am",
    "exp": 1345183492181,
    "http://wso2.org/claims/subscriber": "admin",
    "http://wso2.org/claims/applicationname": "app2",
    "http://wso2.org/claims/apicontext": "/placeFinder",
    "http://wso2.org/claims/version": "1.0.0",
    "http://wso2.org/claims/tier": "Silver",
    "http://wso2.org/claims/enduser": "sumedha"
}
```

The above token contains,

- Token expiration time ("exp")
- Subscriber to the API, usually the app developer ("http://wso2.org/claims/subscriber")
- Application through which API invocation is done ("http://wso2.org/claims/applicationname")
- Context of the API ("http://wso2.org/claims/apicontext")
- API version ("http://wso2.org/claims/version")
- Tier/price band for the subscription ("http://wso2.org/claims/tier")
- Enduser of the app who's action invoked the API ("http://wso2.org/claims/enduser")

Let's see how to enable and pass information in the JWT or completely alter the JWT generation logic in the API Manager:

- [Configuring JWT](#)
- [Customize the JWT generation](#)

#### **Configuring JWT**

Before passing enduser attributes, you enable and configure the JWT implementation in the <APIM\_HOME>/repository/conf/api-manager.xml file. The relevant elements are described below. If you do not configure these elements, they take their default values.

Element	Description
<EnableTokenGeneration>	Set this value to <b>true</b> to enable JWT. <b>Note</b> that if you publish APIs before J
<SecurityContextHeader/>	The name of the HTTP header to which the JWT is attached.

<ClaimsRetrieverImplClass/>	<p>By default, the &lt;ClaimsRetrieverImplClass&gt; parameter is commented in the JWT token:</p> <pre>&lt;ClaimsRetrieverImplClass&gt;org.wso2.carbon.apimgt.impl.toke</pre> <p>By default, the following are encoded to the JWT:</p> <ul style="list-style-type: none"> <li>• subscriber name</li> <li>• application name</li> <li>• API context</li> <li>• API version</li> <li>• authorized resource owner name</li> </ul> <p>In addition, you can also write your own class by extending the interface or implementing the following methods of the interface:</p>										
	<table border="1"> <thead> <tr> <th>Method</th><th>Description</th></tr> </thead> <tbody> <tr> <td>void init() throws APIManagementException;</td><td>Used to perform initialization tasks. Is executed when the API token is generated.</td></tr> <tr> <td>SortedMap&lt;String, String&gt; getClaims(String endUserName) throws APIManagementException;</td><td>Returns a sorted map of claims. The key indicates the corresponding user attribute and depends on the ordering defined by the attribute.</td></tr> <tr> <td>String getDialectURI(String endUserName);</td><td>The dialect URI to which the attribute name example, if the getClaims method returns {email : "male", "http://wso2.org/claims": "http://wso2.org/cla:}</td></tr> <tr> <td></td><td>The default implementation (org.wso2.carbon.apimgt.impl.ClaimsRetriever) returns the user's attributes defined under the consumer dialect and also be encoded with the same dialect of the attributes. If no value is specified, the attributes.</td></tr> </tbody> </table>	Method	Description	void init() throws APIManagementException;	Used to perform initialization tasks. Is executed when the API token is generated.	SortedMap<String, String> getClaims(String endUserName) throws APIManagementException;	Returns a sorted map of claims. The key indicates the corresponding user attribute and depends on the ordering defined by the attribute.	String getDialectURI(String endUserName);	The dialect URI to which the attribute name example, if the getClaims method returns {email : "male", "http://wso2.org/claims": "http://wso2.org/cla:}		The default implementation (org.wso2.carbon.apimgt.impl.ClaimsRetriever) returns the user's attributes defined under the consumer dialect and also be encoded with the same dialect of the attributes. If no value is specified, the attributes.
Method	Description										
void init() throws APIManagementException;	Used to perform initialization tasks. Is executed when the API token is generated.										
SortedMap<String, String> getClaims(String endUserName) throws APIManagementException;	Returns a sorted map of claims. The key indicates the corresponding user attribute and depends on the ordering defined by the attribute.										
String getDialectURI(String endUserName);	The dialect URI to which the attribute name example, if the getClaims method returns {email : "male", "http://wso2.org/claims": "http://wso2.org/cla:}										
	The default implementation (org.wso2.carbon.apimgt.impl.ClaimsRetriever) returns the user's attributes defined under the consumer dialect and also be encoded with the same dialect of the attributes. If no value is specified, the attributes.										
<ConsumerDialectURI/>	<p>The dialect URI under which the user's claims are to be looked for. Only works if defined above.</p> <p>The JWT token contains all claims defined in the &lt;ConsumerDialectURI&gt; element. To get a list of users to be included in the JWT, simply uncomment &lt;claims&gt; to the JWT token.</p>										
<SignatureAlgorithm/>	<p>The signing algorithm used to sign the JWT. The general format of the JWT is: NONE is specified as the algorithm, signing is turned off and the JWT looks like: by a period and a period at the end.</p> <p>This element can have only two values- the default value, which is SHA256V</p>										

### Customize the JWT generation

The JWT that is generated by default (see example above) has predefined attributes that are passed to the backend. These include basic application-specific details, subscription details, and user information that are defined

in the JWT generation class that comes with the API Manager by the name `org.wso2.carbon.apimgt.impl.token.JWTGenerator`. If you want to pass additional attributes to the backend with the JWT or completely change the default JWT generation logic, do the following:

1. Write your own custom JWT implementation class by extending the default `JWTGenerator` class. A typical example of implementing your own claim generator is given below. It implements the `populateCustomClaims()` method to generate some custom claims and adds them to the JWT.

```

import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dto.APIKeyValidationInfoDTO;
import org.wso2.carbon.apimgt.impl.token.JWTGenerator;
import org.wso2.carbon.apimgt.api.*;

import java.util.Map;

public class CustomTokenGenerator extends JWTGenerator {

    public Map populateStandardClaims(APIKeyValidationInfoDTO keyValidationInfoDTO, String apiContext, String version)
            throws APIManagementException {
        Map claims = super.populateStandardClaims(keyValidationInfoDTO, apiContext, version);
        boolean isApplicationToken =
                keyValidationInfoDTO.getUserType().equalsIgnoreCase(APIConstants.ACCESS_TOKEN_USE_R_TYPE_APPLICATION) ? true : false;
        String dialect = getDialectURI();
        if (claims.get(dialect + "/enduser") != null) {
            if (isApplicationToken) {
                claims.put(dialect + "/enduser", "null");
                claims.put(dialect + "/enduserTenantId", "null");
            } else {
                String enduser = claims.get(dialect + "/enduser");
                if (enduser.endsWith("@carbon.super")) {
                    enduser = enduser.replace("@carbon.super", "");
                    claims.put(dialect + "/enduser", enduser);
                }
            }
        }
        return claims;
    }

    public Map populateCustomClaims(APIKeyValidationInfoDTO keyValidationInfoDTO, String apiContext, String version, String accessToken)
            throws APIManagementException {
        Long time = System.currentTimeMillis();
        String text = "This is custom JWT";
        Map map = new HashMap();
        map.put("current_timestamp", time.toString());
        map.put("messge" , text);
        return map;
    }
}

```

2. Build your class and add the JAR file to `<APIM_HOME>/repository/components/lib` directory.

- Add your class in the <TokenGeneratorImpl> element of the <APIM\_HOME>/repository/conf/api-manager.xml file.

```
<APIConsumerAuthentication>
  ...
<TokenGeneratorImpl>org.wso2.carbon.test.CustomTokenGenerator</TokenGeneratorImpl>
  ...
</APIConsumerAuthentication>
```

- Set the <EnableTokenGeneration> element to **true** in the api-manager.xml file.
- Restart the server.

## Encrypting Secure Endpoint Passwords

When creating an API using the API Publisher, you specify the endpoint of its backend implementation in the **Implement** tab. If you select the endpoint as secured, you are prompted to give credentials in plain-text.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like APIs, Browse, Add, All Statistics, My APIs, Subscriptions, and Statistics. The 'My APIs' option is currently selected. In the main area, there's a navigation bar with three steps: Design, Implement (which is highlighted with a red box), and Manage. Below this, the API details for 'test : /test/1.0.0' are shown. Under the 'Endpoints' section, the 'Endpoint Type' is set to 'HTTP Endpoint'. The 'Production Endpoint' field contains 'http://appserver/resource'. Below it, the 'Sandbox Endpoint' field is empty. The 'Endpoint Security Scheme' dropdown is set to 'Secured' and is also highlighted with a red box. The 'Credentials' input fields, which are used for storing plain-text passwords, are also highlighted with a red box. There are 'Advanced Options' and 'Test' buttons for both the production and sandbox endpoints.

The steps below show how to secure the endpoint's password that is given in plain-text in the UI.

- Shut down the server if it is already running and set the element <EnableSecureVault> in <APIM\_HOME>/repository/conf/api-manager.xml to true. By default, the system stores passwords in configuration files in plain text because this value is set to false.
- Define synapse property in the synapse.properties file as follows: synapse.xpath.func.extensions=org.wso2.carbon.mediation.security.vault.xpath.SecureVaultLookupXPathFunctionProv

- der.
- Run the cipher tool available in <APIM\_HOME>/bin. If on windows, the file is `ciphertool.bat`. If you are using the default keystore, give `wso2carbon` as the primary keystore password when prompted.

```
sh ciphertool.sh -Dconfigure
```

## Maintaining Logins and passwords

This section covers the following topics:

- Changing the super admin password
- Recovering a password
- Logging in via multiple user store attributes
- Setting up primary and secondary logins
- Setting up an e-mail login
- Setting up a social media login

### ***Changing the super admin password***

To change the default admin password, log in to the management console with admin/admin credentials and use the "Change my password" option. After changing the admin credentials, do the following:

1. Change the admin credentials in the <APIM\_HOME>/repository/conf/user-mgt.xml file:

```
<UserManager>
  <Realm>
    <Configuration>
      ...
      <AdminUser>
        <UserName>admin</UserName>
        <Password>admin</Password>
      </AdminUser>
      ...
    </Realm>
  </UserManager>
```

2. Go to the **Resources -> Browse** menu in the management console to open the registry and update the credentials in `/_system/governance/apimgt/applicationdata/sign-up-config.xml` registry location.

#### **Do you have any special characters in passwords?**

If you specify passwords inside XML files, take care when giving special characters in the user names and passwords. According to XML specification (<http://www.w3.org/TR/xml/>), some special characters can disrupt the configuration. For example, the ampersand character (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA ([http://www.w3schools.com/xml/xml\\_cdata.asp](http://www.w3schools.com/xml/xml_cdata.asp)) as shown below or remove the character:

```
<Password>
  <! [ CDATA[xnvYh?@VHAKc?qZ%Jv855&A4a,%M8B@h] ]>
</Password>
```

## **Recovering a password**

See [How can I recover the admin password used to log in to the management console?](#)

## **Logging in via multiple user store attributes**

See [Authentication using Attributes](#) in the WSO2 IS documentation.

## **Setting up primary and secondary logins**

In a standalone deployment of the API Manager instance, users of the API Store can have a secondary login name in addition to the primary login name. This gives the user flexibility to provide either an email or a user name to log in. You can configure the API Store to treat both login names as belonging to a single user. Users can invoke APIs with the same access token without having to create a new one for the secondary login.

You can configure this capability using the steps below.

1. Configure user login under the `<OAuth>` element in `<APIM_HOME>/repository/conf/identity.xml` file.
  - a. Mention your primary and secondary login names. Set the `primary` attribute of the primary login to `true` and the `primary` attribute of the secondary login to `false`.
  - b. Primary login doesn't have a `ClaimUri`. Leave this field empty.
  - c. Provide the correct `ClaimUri` value for the secondary login

An example is given below:

```
<OAuth>
    .... .
    <LoginConfig>
        <UserIdLogin primary="true">
            <ClaimUri></ClaimUri>
        </UserIdLogin>
        <EmailLogin primary="false">
            <ClaimUri>http://wso2.org/claims/emailaddress</ClaimUri>
        </EmailLogin>
    </LoginConfig>
</OAuth>
```

2. In the API Store of a distributed setup, the `serverURL` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file should point to the key manager instance's service endpoint. This allows users to connect to the key manager's user store to perform any operations related to API Store such as login, access token generation etc. For example,

```
<AuthManager>
    <!--Server URL of the Authentication service -->
    <ServerURL>https://localhost:9444/services/</ServerURL>

    <!-- Admin username for the Authentication manager. -->
    <Username>admin</Username>

    <!-- Admin password for the Authentication manager.-->
    <Password>admin</Password>
</AuthManager>
```

 **Tip:** In a distributed setup, the API Store's user store needs to point to the key manager user store.

 **Tip:** Be sure to keep the secondary login name unique to each user.

### Setting up an e-mail login

See [Email Authentication](#) in the WSO2 IS documentation.

### Setting up a social media login

You can auto provision users based on a **social network login** by integrating the API Manager with WSO2 Identity Server. But, this is not supported in a **multi-tenant environment**.

In a multi-tenant environment, the system cannot identify the tenant domain in the login request that comes to API Manager's Publisher/Store. Therefore, the service provider is registered as a SaaS application within the super tenant's space. Configuring user provisioning is part of creating the service provider. In order to authenticate the user through a third party identity provider such as a social network login, you must enable identity federation. As the service provider is created in the super tenant's space, the provisioned user is also created within the super tenant's space. As a result, it is not possible to provision the user in the tenant's space.

To overcome this limitation, you can write a custom authenticator to retrieve the tenant domain of the user and write a custom login page where the user can enter the tenant domain, which is then added to the authenticator context. Then, write a custom provisioning handler to provision the user in the tenant domain that maintained in the context.

- For information on writing a custom authenticator, see [Creating Custom Authenticators](#) in the WSO2 IS documentation.
- For information on writing a custom login page, see [Customizing Login Pages](#) in the WSO2 IS documentation.

## Saving Access Tokens in Separate Tables

You can configure the API Manager instances to store access tokens in different tables according to their user store domains. This is referred to as **user token partitioning** and it ensures better security when there are multiple user stores configured in the system. To configure user stores other than the default one, see [Configuring Secondary User Stores](#).

The following topics explain how to enable user token partitioning:

- Enabling assertions
- Storing keys in different tables

### Enabling assertions

You use assertions to embed parameters into tokens and generate a strong access token. You can also use these parameters later for other processing. At the moment, the API Manager only supports UserName as an assertion.

By default, assertions are set to `false` in `<APIM_HOME>/repository/conf/identity.xml`. To enable it, set the `<UserName>` element to `true`. You can add a user name to an access token when generating the key, and verify it by encoding the retrieved access token with Base64.

#### <APIM\_HOME>/repository/conf/identity.xml

```
<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>
```

### Storing keys in different tables

1. If the `<UserName>` assertion is enabled, set the `<EnableAccessTokenPartitioning>` element in `<APIM_HOME>/repository/conf/identity.xml` file to `true`. It determines whether you want to store the keys in different tables or not.

```
<EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>
```

2. Set the user store domain names and mappings to new table names. For example,

- if userId = [foo.com/admin](#) where 'foo.com' is the user store domain name, then a 'mapping:domain' combo can be defined as 'A:foo.com'
- 'A' is the mapping for the table that stores tokens relevant to users coming from 'foo.com' user store

In this case, the actual table name is `IDN_OAUTH2_ACCESS_TOKEN_A`. We use a mapping simply to prevent any issues caused by lengthy table names when lengthy domain names are used. You must manually create the tables you are going to use to store the access tokens in each user store (i.e., manually create the tables `IDN_OAUTH2_ACCESS_TOKEN_A` and `IDN_OAUTH2_ACCESS_TOKEN_B` according to the following defined domain mapping). This table structure is similar to the `IDN_OAUTH2_ACCESS_TOKEN` table defined in the `api-manager dbscript`, which is inside the `<APIM_HOME>/dbscripts/apimgt` directory.

You can provide multiple mappings separated by commas as follows. Note that the domain names need to be specified in upper case.

```
<AccessTokenPartitioningDomains>A:FOO.COM,  
B:BAR.COM</AccessTokenPartitioningDomains>
```

3. According to the information given above, change the `<OAuth>` element in the `<APIM_HOME>/repository/conf/identity.xml` file as shown in the following example:

```
<APIM_HOME>/repository/conf/identity.xml

<!-- Assertions can be used to embed parameters into access token.-->
<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>

<!-- This should be set to true when using multiple user stores and keys should
     saved into different tables according to the user store. By default all the
     application keys are saved in to the same table. UserName Assertion should be
     'true' to use this.-->
<AccessTokenPartitioning>
    <EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>
        <!-- user store domain names and mappings to new table names. eg: if you
             provide 'A:foo.com', foo.com should be the user store domain
             name and 'A' represent the relavant mapping of token storing table, i.e.
             tokens relevant to the users comming from foo.com user store
             will be added to a table called IDN_OAUTH2_ACCESS_TOKEN_A. -->
    <AccessTokenPartitioningDomains>A:foo.com,
    B:bar.com</AccessTokenPartitioningDomains>
</AccessTokenPartitioning>
```

## Configuring WSO2 Identity Server as the Key Manager

The **Key Manager** handles all clients, security and access token-related operations. For more information, see [Key Manager](#).

To configure WSO2 Identity Server as the Key Manager of the API Manager, see [Configuring WSO2 Identity Server](#)

as the Key Manager in [WSO2 API Manager](#) in the WSO2 Clustering documentation.

## Configuring a Third-Party Key Manager

The **Key Manager** handles all clients, security and access token-related operations. In a typical API Manager production deployment, different components talk to the Key Manager component for achieving different tasks. The API Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions and API invocations. When a subscriber generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth App and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database. For more information, see [Key Manager](#).

The Key Manager decouples OAuth client and access token management from the rest of its operations, so that you can plug in a third-party OAuth provider for managing OAuth clients and access tokens. Let's see what basic steps to follow when writing a Key Manager implementation that acts as the bridge between a third-party OAuth provider and the API Manager.

In this guide, we use the **Surf OAuth Authorization Server** for managing OAuth clients and tokens required by the API Manager. We have a sample client implementation that consumes APIs exposed by Surf OAuth. You find a live version of Surf OAuth [here](#).

### **Starting the authorization server**

1. Download the binary located [here](#) and deploy it in a tomcat server. Alternatively, you can build the OAuth Server from scratch and start the server by issuing the `mvn jetty:run` command in the `api-authorization-server-war` folder.



**Tip:** We have done the following changes to the Web application you just downloaded:

- `apis.application.properties` file is copied to the classpath.
- All the URLs starting with `localhost` are replaced by the loop-back IP (`127.0.0.1`)
- `org.surfnet.oaaas.noop.NoopAuthenticator` authenticator is set as the default authenticator.
- Token expiry time is increased to 99999 seconds. This ensures that the tokens issued for the Web client lasts several months.

2. Move the Web application to the ROOT context to ensure that the Surf OAuth Web applications works on Tomcat.

```
rm -rf tomcat7/webapps/ROOT
mv tomcat7/webapps/surf-oauth tomcat7/webapps/ROOT
```

3. Access <http://127.0.0.1:8080/> to see the following page:

The screenshot shows the Surf OAuth 2.0 UI homepage. The top navigation bar has the title "SURF OAUTH". The left sidebar contains four links: "Resource Servers", "Client Applications", "Access Tokens", and "Statistics". The main content area features a large heading "OAuth 2.0. But dead simple." followed by a descriptive text: "Can you imagine getting an OAuth2 compliant Authorization Server (and this client apparently;-) up in a matter of minutes? Wait and see. By the way, the Apis Authorization Server lets you authenticate against any possible backend of your choice and is totally agnostic as it comes to the flavor of your Resource Server." Below the text is a blue "Login" button.

The server is now up and running. Next, let's create a Resource Server and an OAuth Client.

4. In Surf OAuth UI, click the **Resource Servers** link where all the OAuth clients are grouped together, and

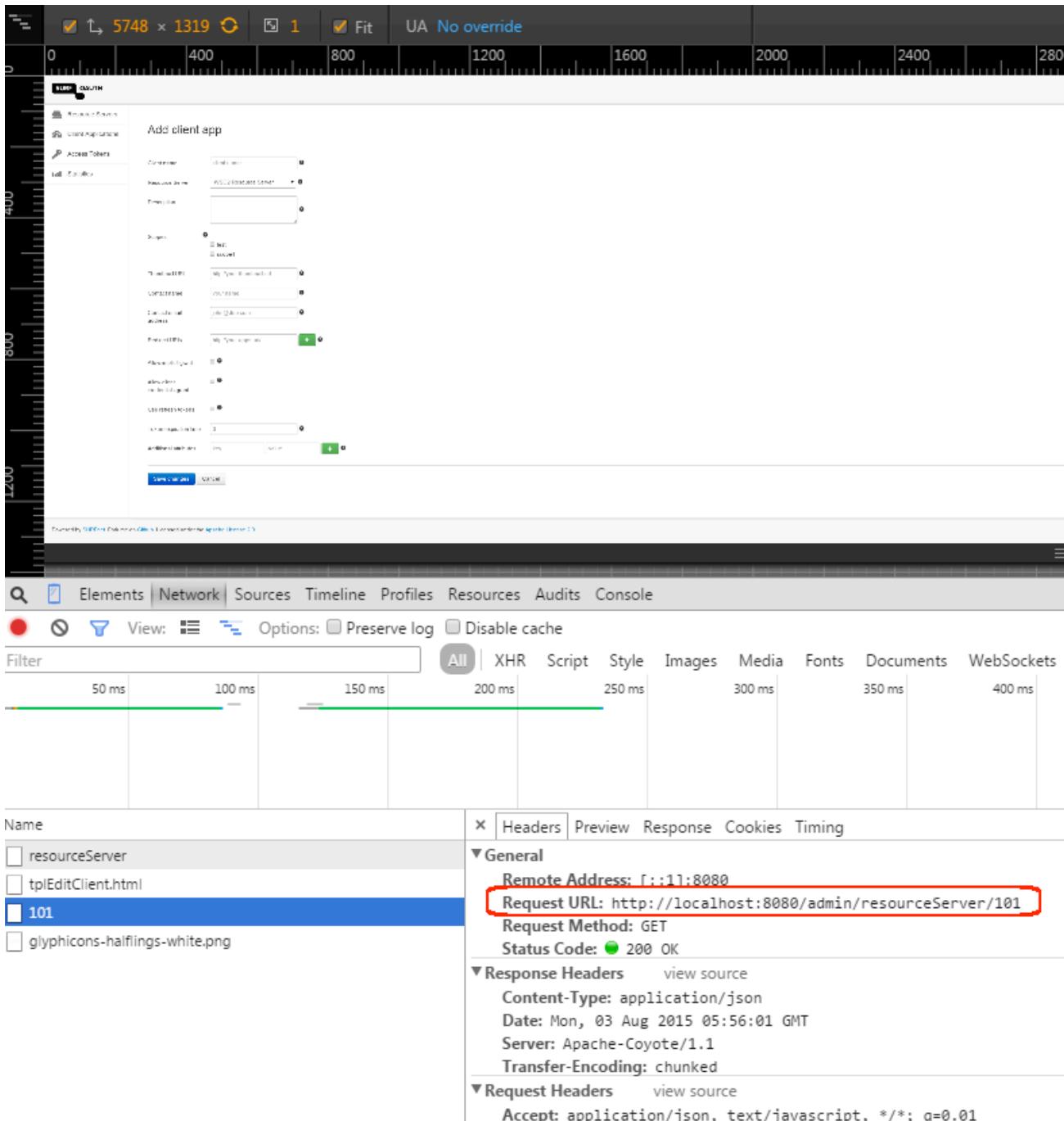
register a resource server representing WSO2 API Manager. Also, add two scopes named test and scope1. You will use them when creating clients.

The front end is now registered as a distinct client with the authorization server.

- Click the **Access Token** link and note all the tokens issued for the Web client. These tokens are obtained at the time you log in, by a Javascript client running on the browser. The same token is then used for subsequent operations.

Token	Resource server	Client	Scopes	Resource owner ID	Issue date	Valid until	Actions
57e2174f-a133-4d16-a821-b20e83927cf6	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 6:42:16 AM	5/6/2015, 10:28:54 AM	<button>Delete</button>
06d66602-6af2-4d70-addf-f22d766d3b46	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 10:53:46 AM	5/6/2015, 2:40:25 PM	<button>Delete</button>
0df05cb9-8c7b-4172-a5a8-7520bdbd5418	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 11:09:34 AM	5/6/2015, 2:56:13 PM	<button>Delete</button>
4154f2a7-b2a6-45af-aed7-35633b1c91aa	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 2:03:59 PM	5/6/2015, 5:50:38 PM	<button>Delete</button>

- Pick an active access token from the above list. You use it to create clients through the API Manager.
- Get a registration endpoint to register the client with. As Surf OAuth doesn't support a spec-compliant client registration yet, you can use an endpoint with similar capabilities. For example, you can enable Developer Tools in Google Chrome to see the URL and the request being sent as shown below:



### Configuring the API Manager

1. Build the demo.client available at <https://github.com/jaadds/surf-oauth-demo.git> and copy the built JAR to <KM\_HOME>/repository/components/lib folder. Note that <KM\_HOME> is the API Manager distribution folder where the Key Manager is set up.
2. Uncomment the <APIKeyManager> element in <KM\_HOME>/repository/conf/api-manager.xml file and change the values according to your third-party implementation.

**Tip:** Be sure to replace the <RegistrationEndpoint> and <AccessToken> with the client registration endpoint and the access token you obtained earlier in step 7 and 6. ConsumerKey and Secret should be that of the created Resource Server. Also change the <hostname> in the <IntrospectionURL> accordingly.

```
<APIKeyManager>

<KeyManagerClientImpl>n1.surfnet.demo.SurfOAuthClient</KeyManagerClientImpl>
    <Configuration>
        <RegistrationEndpoint><Give the client registration endpoint you got in step 7></RegistrationEndpoint>
            <AccessToken><Give the access token you got in step 6></AccessToken>

        <IntrospectionURL>http://<hostname>:port/v1/tokeninfo</IntrospectionURL>
            <ConsumerKey>xxx</ConsumerKey>
            <ConsumerSecret>xxx</ConsumerSecret>
    </Configuration>
</APIKeyManager>
```



**Tip:** See the [WSO2 default Key Manager implementation](#) for a sample Key Manager implementation.

### Running the sample

You have connected the API Manager with a third-part authorization server. Let's see how the API Manger creates OAuth clients at Surf OAuth when applications are registered in the API Store. In this guide, we use the [Published APIs](#) to test invoke this process.

1. Start the API Manager.
2. Log in to the API Store and create an application.

```
curl -k -X POST -c cookies
https://localhost:9443/store/site/blocks/user/login/ajax/login.jag -d
'action=login&username=admin&password=admin'
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/application/application-add/ajax/application-add.jag -d
'action=addApplication&application=SurfClientApp&tier=Unlimited&description=&callbackUrl='
```

3. Register an OAuth client of type PRODUCTION the authorization server. Note that you are sending the specific parameters required by the OAuth Server in JSON.

```
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d
'action=generateApplicationKey&application=SurfClientApp&authorizedDomains=ALL&keyType=PRODUCTION&validityTime=3600&jsonParams={ "scopes": [ "test" ], "contactName": "John Doe", "contactEmail": "john@doe.com" }'
```

4. Go to the **Client Applications** link in the Surf OAuth UI and note the newly created client listed there.

The screenshot shows the SURF OAuth interface. On the left, there's a sidebar with links: 'Resource Servers' (selected), 'Client Applications' (highlighted with a red box), 'Access Tokens', and 'Statistics'. The main area is titled 'Resource servers' and shows a table with one row for 'WSO2 Resource Server'. The table columns are 'Icon', 'Resource server', 'Client apps', 'Scopes', 'About', and 'Actions'. The 'About' section contains fields for 'Contact' (John Doe), 'Key' (6b3dce67-dfc4-44fd-8165-007cb19e4272), and 'Secret' (9358b2b6-8e17-451e-860d-973a5e54a91f). A 'Delete' button is in the 'Actions' column. Below this is a section titled 'Client applications' with a table containing one row for 'SurfClientApp\_PRODUCTION'. The table columns are 'Icon', 'Name', 'Client ID', 'Scopes', 'Credentials', and 'Actions'. The 'Credentials' section shows 'Client ID' (surfclientapp\_production) and 'Secret' (dd32e25b-87fc-411c-97f4-6c9a3205274e). A 'Delete' button is in the 'Actions' column. There are 'Add another server' and 'Add another client app' buttons at the top right of their respective tables.

You have now created an application and registered an OAuth Client corresponding to it. Let's see how to validate tokens by subscribing to a SurfClient application and obtaining a token.

5. Log in to the API Publisher and deploy the sample API (WeatherAPI) if you haven't done so already.

The screenshot shows the WSO2 API PUBLISHER interface. On the left, there's a sidebar with links: 'APIs' (selected), 'Browse' (highlighted with a red box), 'Add', 'All Statistics', 'My APIs', 'Subscriptions', and 'Statistics'. The main area is titled 'APIs / All' and shows a large button labeled 'All APIs' with the sub-instruction 'No APIs created yet. Click one of below buttons to get started.' Below this are two buttons: 'New API...' and 'Deploy Sample API' (highlighted with a red box).

6. Assuming you still have the OAuth client created earlier, subscribe to this API as follows:

```
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d
'action=addAPISubscription&name=WeatherAPI&version=1.0.0&provider=admin&tier=Unlimited&applicationName=SurfClientApp'
```

Let's obtain a token from the OAuth Provider.

7. Go to the **Edit** view of the OAuth client and make sure the `client_credentials` grant type is enabled, and a token expiration time is specified.

**Scopes**

test  
 openid

**Thumbnail URL**

**Contact name**

**Contact email address**

**Redirect URIs**  -  
 + ?

**Allow implicit grant**  ?

**Allow client credentials grant**  ?

**Use refresh tokens**  ?

**Token expiration time**

**Additional attributes**   + ?

---

Save changes

Cancel

#### 8. Obtain a token.

```
curl -k -d "grant_type=client_credentials&scope=test" -H "Authorization: Basic bWFwcGVkY2xpZW50OjIxOTQ0Y2Y3LTEyOWMtNDY4OC05Y2E0LTJkMGFjMGY3NWU1Nw==, Content-Type: application/x-www-form-urlencoded" http://localhost:8080/v1/token
```

#### 9. Invoke the API using the token obtained.

```
curl -k -H "Authorization: Bearer 2f2a7542-4001-4491-a415-e151cbb9f45e" http://localhost:8280/weatherapi/1.0.0
```

## Working with Statistics

The following topics describe how to monitor API invocations and how to collect and summarize statistics in order to monetize API usage.

- Publishing API Runtime Statistics
- Integrating with Google Analytics
- Viewing API Statistics

## Publishing API Runtime Statistics

In this section, we explain how to set up **WSO2 Business Activity Monitor** (**version 2.5.0** is used here) to collect and analyze runtime statistics from the API Manager. To publish data from the API Manager to BAM, the Thrift protocol is used. Information processed in BAM is stored in a database from which the API Publisher retrieves information before displaying it in the corresponding UI screens.

By default, `org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataPublisher` is configured to push data events to WSO2 BAM. If you use a product other than WSO2 BAM to collect and analyze runtime statistics, you write a new data publishing agent by extending `APIMgtUsageDataPublisher`. Find the API templates inside `<APIM_HOME>/repository/resources/api_templates`. When writing a new data publishing agent, make sure the data publishing logic has a minimal impact to API invocation.

 **Tip:** The datasource and database names used here are just examples. They may vary depending on your configurations.

- Prerequisites
- Configure WSO2 BAM
- Configure WSO2 API Manager
- Change the statistics database
- Troubleshoot common issues

### Prerequisites

- JDK 1.6.\* or 1.7

 If you install JDK in the `Program Files` folder in **Windows**, avoid the space by using `PROGRA~1` in the `JAVA_HOME` and `PATH` environment variables. Else, the server throws an exception.

- Cygwin (<http://www.cygwin.com>): Required **only if you are using Windows**. WSO2 BAM analytics framework depends on Apache Hadoop, which requires Cygwin in order to run on Windows. Install at least the basic net (OpenSSH,tcp\_wrapper packages) and security related Cygwin packages. After Cygwin installation, update the `PATH` variable with `C:/cygwin/bin` and restart BAM.

Next prepare BAM to collect and analyze statistics from the API Manager.

### Configure WSO2 BAM

1. Download WSO2 BAM 2.5.0 from location: <http://wso2.com/products/business-activity-monitor>.
2. Apply an offset of 3 to the default BAM port by editing the `<BAM_HOME>/repository/conf/carbon.xml` file.

```
<Offset>3</Offset>
```

This increments all ports used by the server by 3, which means the BAM server will run on port 9446. Port offset is used to increment the default port by a given value. It avoids possible port conflicts when multiple WSO2 products run on same host.

3. Install MySQL server and a suitable client like the MySQL Workbench. You can get the instructions manual from <http://dev.mysql.com/doc/>.
4. Go to the command-line and issue the following commands to connect to the MySQL server and create a

database (e.g., TestStatsDB). This database is used to save the statistical data collected by the BAM. You do not need to create any tables in it.

```
mysql -u <username> -p <password> -h <host_name or IP>;
CREATE DATABASE TestStatsDB;
```

5. Save the MySQL connector JAR inside both <APIM\_HOME>/repository/components/lib and <BAM\_HOME>/repository/components/lib folders.
6. Give the datasource definition under the <datasource> element in the <BAM\_HOME>/repository/conf/datasources/master-datasources.xml file. The tables are created automatically when the Hive script runs. You just need to create the schema. WSO2AM\_STATS\_DB is used to fetch analytical data from the database. The example below connects to a MySQL instance:

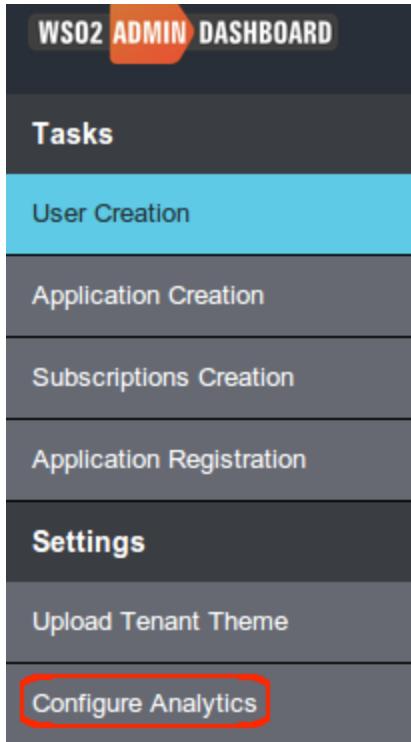
```
<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for getting statistics to API Manager</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/TestStatsDB</url>
            <username>db_username</username>
            <password>db_password</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

 **Tip:** If you are using **BAM 2.4.1**, be sure to uncomment the <thriftDataReceiver><hostName> element in the <BAM\_HOME>/repository/conf/data-bridge/data-bridge-config.xml file and give the BAM host IP there.

7. Restart BAM server by running <BAM\_HOME>/bin/wso2server.[sh/bat].

#### Configure WSO2 API Manager

1. Start the API Manager and log in to its Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>).
2. Click the **Configure Analytics** menu.



3. Select the **Enable** check-box to enable statistical data publishing and do the rest of the configurations using the information given below:

<b>Event Receiver Configurations</b>	<p>Set the URL_to <code>tcp://&lt;BAM_HOST_IP&gt;:7614/</code> where <code>&lt;BAM_HOST_IP&gt;</code> is the machine's IP address. Do not use localhost unless you're in a disconnected mode.</p> <p>You can define multiple event receiver groups, each with one or more receivers separated by commas. For an example <code>tcp://localhost:7612/, tcp://localhost:7613/</code>. This helps you <b>manage failover</b>. If the BAM server in the first URL fails, the request will be routed to the second one.</p> <p>Event receivers refer to the endpoint to which events are published from the API Gateway. Because you apply an offset to the default BAM port later in this guide, you need to apply the same offset to the default Thrift port. The API Manager then pushes the data to BAM through port 7614, using the Thrift protocol.</p>
<b>Data Analyser Configurations</b>	URL and the credentials of the event analyzer node. As this URL is used to deploy the toolbox, make sure that the BAM server is up and running in the given URL.
<b>Statistic Summary Datasource</b>	<p>Give the datasource definition that is used to store summarized statistical data. The tables are created automatically when the Hive script runs. You just need to create the schema. The same configurations will be done in the BAM server.</p> <ul style="list-style-type: none"> <li>• URL: The connection URL for the RDBMS datasource</li> <li>• JDBC Driver Class: The fully qualified Java class name of the JDBC driver</li> <li>• Username/Password: Credentials to be passed to the JDBC driver to establish a connection</li> </ul>

 **Tip:** To edit the datasource connection pool parameters, click the **Show More Options** link.

## Configure Analytics

Enable API usage publishing and Statistics aggregation

Enable

### Event Receiver Configurations

URL Group: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/>	<input type="text"/> admin	<input type="text"/> .....
<input type="button"/> Add URL Group		

Event Receiver Group	Username	Actions
(tcp://localhost:7614)	admin	<input type="button"/>

### Data Analyzer Configurations

URL: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/> https://localhost:9446	<input type="text"/> admin	<input type="text"/> .....
<input type="button"/>		

### Statistics Summary Datasource

URL: <sup>*</sup>	JDBC Driver Class: <sup>*</sup>	Username: <sup>*</sup>	Password: <sup>*</sup>
<input type="text"/> jdbc:mysql://localhost:3306/TestStatsDB	<input type="text"/> com.mysql.jdbc.Driver	<input type="text"/> root	<input type="text"/> .....
<input type="button"/> Show More Options			

Save

4. Click **Save** when you are done. It deploys the Analytics toolbox, which describes the information collected, how to analyze the data, and the location of the database where the analyzed data is stored.

 **Tip:** Are you working with an API Manager cluster or a distributed setup? If so,

- If your registry is shared, do the above configurations in one node (e.g., the API Publisher) and restart the other nodes.
- If your registry is not shared, do the same configuration in all API Gateway nodes, API Publisher node and API Store nodes by logging in to the admin-dashboard.
- Change the API Publisher node to get response-based statistics such as [destination-based usage tracking](#).

You change the stream names, versions, publisher class by editing the <APIM\_HOME>/repository/conf/api-manager.xml file as given in the example below:

 **Tip:** Please read the code comments for details. If you change the default values under streams, the <APIM\_HOME>/statistics/API\_Manager\_Analytics.tbox must also be changed accordingly.

```

<APIUsageTracking>
    <!-- Below property is used to skip trying to connect to event receiver
nodes when publishing events even if
        the stats enabled flag is set to true. -->
    <SkipEventReceiverConnection>false</SkipEventReceiverConnection>

    <!-- API Usage Data Publisher. -->

<PublisherClass>org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataBridgeDataP
ublisher</PublisherClass>
    <!-- If below property set to true,then the response message size will be
calculated and publish
        with each successful API invocation event. -->
    <PublishResponseMessageSize>false</PublishResponseMessageSize>
    <!-- Data publishing stream names and versions of API requests, responses
and faults. If the default values
        are changed, the toolbox also needs to be changed accordingly. -->
    <Streams>
        <Request>
            <Name>org.wso2.apimgt.statistics.request</Name>
            <Version>1.0.0</Version>
        </Request>
        <Response>
            <Name>org.wso2.apimgt.statistics.response</Name>
            <Version>1.0.0</Version>
        </Response>
        <Fault>
            <Name>org.wso2.apimgt.statistics.fault</Name>
            <Version>1.0.0</Version>
        </Fault>
        <Destination>
            <Name>org_wso2_apimgt_statistics_destination</Name>
            <Version>1.0.0</Version>
            <BAMProfileName>bam-profile</BAMProfileName>
        </Destination>
        <Throttle>
            <Name>org.wso2.apimgt.statistics.throttle</Name>
            <Version>1.0.0</Version>
        </Throttle>
        <Workflow>
            <Name>org.wso2.apimgt.statistics.workflow</Name>
            <Version>1.0.0</Version>
        </Workflow>
    </Streams>
</APIUsageTracking>

```

After configuring WSO2 BAM to collect and analyze statistics of APIs hosted and managed by the API Manager, you can view them through various statistical dashboards in the API Publisher and Store, depending on your permission [Viewing API Statistics](#). For information, see

#### **Change the statistics database**

To use a different database than the default H2 for statistical publishing, you change the properties of the datasource element, and additionally delete some metadata tables created by previous executions of the Hive script, if there are any.

To delete the metadata tables,

1. Log in to BAM management console and select **Add** in **Analytics** menu.
2. Go to the Script Editor in the window that opens.
3. Execute the following script.

```
drop TABLE APIRequestData;
drop TABLE APIRequestSummaryData;
drop TABLE APIVersionUsageSummaryData;
drop TABLE APIResourcePathUsageSummaryData;
drop TABLE APIResponseData;
drop TABLE APIResponseSummaryData;
drop TABLE APIFaultData;
drop TABLE APIFaultSummaryData;
drop TABLE APIDestinationData;
drop TABLE APIDestinationDataSummaryData;
```

4. If there are previous executions of the Hive scripts, manually execute them again by going to **Main > Analytics > List** in the management console of BAM. Alternatively, you can wait until the periodical execution time occurs.

#### **Troubleshoot common issues**

Given below is how to do troubleshoot some common issues users come across:

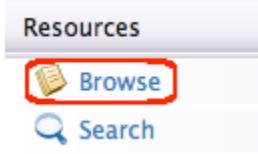
1. Do you get an out of memory issue?  
See the [performance tuning guide](#) for recommendations to tune the server for optimal performance.
2. Do you get an exception as **unable to connect to server** Cassandra?  
Check if you changed the Cassandra port according to the port offset applied to the default BAM port. See [Step 3](#) under configuring BAM section.
3. Do you get a **connection refused exception** on the BAM console?  
This happens when you execute Hive scripts prior to changing the default port. Add the following line at the beginning of the Hive scripts and rerun: `drop table <hive_cassandra_table_name>;` You can find the Hive scripts deployed with the toolbox file, which is inside `<BAM_HOME>/repository/deployment/server/bam-toolbox` folder. For information, see [Editing an Analytic Script](#) in WSO2 BAM documentation.

## **Integrating with Google Analytics**

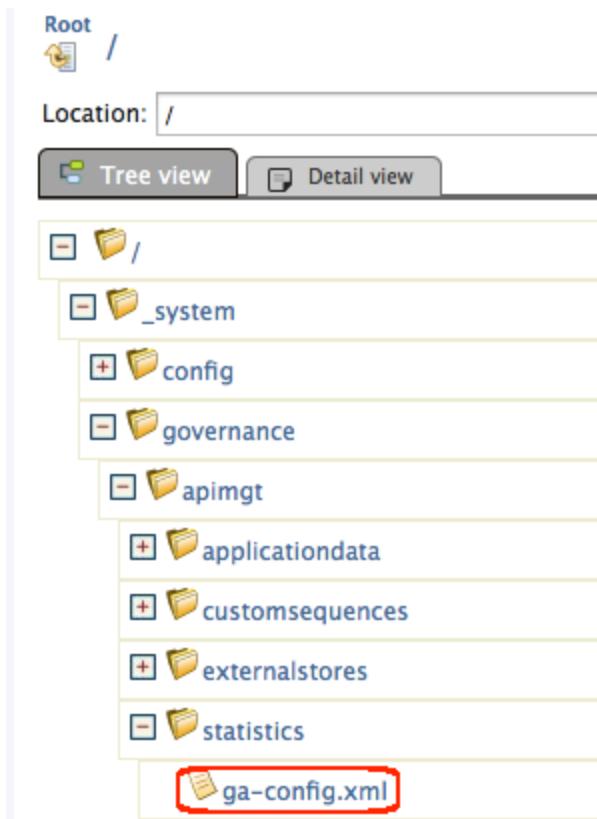
You can configure the API Manager to track runtime statistics of API invocations through Google Analytics (<http://www.google.com/analytics>). Google Analytics is a service that allows you to track visits to a website and generate detailed statistics on them.

This guide explains how to setup API Manager in order to feed runtime statistics to Google analytics for summarization and display.

1. Setup a Google Analytics account if not subscribed already and receive a Tracking ID, which is of the format "UA-XXXXXXX-X". A Tracking ID is issued at the time an account is created with Google Analytics.
2. Log in to the API Manager management console (<https://localhost:9443/carbon>) using admin/admin credentials and go to **Main -> Resources -> Browse** menu.



3. Navigate to `/_system/governance/apimgt/statistics/ga-config.xml` file.



4. Change the <Enabled> element to true, set your tracking ID in <TrackingID> element and **Save**.

**Content**

Display as text | **Edit as text** (highlighted with a red box) | Upload | Download

Plain Text Editor (radio button selected) | Rich Text Editor

```
<!--Google Analytics publisher configuration. Create Google Analytics account
     and obtain a Tracking ID. Refer http://support.google.com/analytics/bin/answer.py?hl=en&
answer=1009694 -->
<GoogleAnalyticsTracking>
    <!--Enable/Disable Google Analytics Tracking -->
    <Enabled>true</Enabled>

    <!-- Google Analytics Tracking ID -->
    <TrackingID>UA-XXXXXXXX-X</TrackingID>

</GoogleAnalyticsTracking>
```

Save Content | Cancel

5. If you want to enable tracking for tenants, log in to the management console with a tenant's credentials, click **Source View**, and then add the following parameter to the org.wso2.carbon.mediation.registry.WS02Registry registry definition near the top (repeat this step for each tenant):  
`<parameter name="cachableDuration">15000</parameter>`

The following screen shot illustrates this change:

WSO2 API Manager

Home > Manage > Service Bus > Source View > Service Bus Configuration

### Service Bus Configuration

Make the required modifications to the configuration and click 'Update' to apply the changes to the server. Use 'Reset' button to undo your changes.

#### ESB Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <registry provider="org.wso2.carbon.mediation.registry.WSO2Registry">
        <parameter name="cachableDuration">15000</parameter>
    </registry>
    <sequence name="production_key_error">
        <property name="ERROR_CODE" value="900091"/>
        <property name="ERROR_MESSAGE" value="Production key offered to the API with no production endpoint"/>
        <property name="CUSTOM_HTTP_SC" value="403"/>
        <sequence key="fault"/>
    </sequence>
    <sequence name="cors_request_handler">
        <filter source="$ctx:CORSConfiguration.Enabled" regex="true">
            <then>
                <filter source="boolean($tp:Access-Control-Allow-Origin)" regex="false">
                    <then>
                        <property name="Access-Control-Allow-Origin" expression="$ctx:Access-Control-Allow-Origin" scope="transport" type="STRING"/>
                    </then>
                </filter>
                <filter source="boolean($tp:Access-Control-Allow-Methods)" regex="false">
                    <then>
                        <property name="Access-Control-Allow-Methods" value="GET,POST,PUT,DELETE,OPTIONS" scope="transport" type="STRING"/>
                    </then>
                </filter>
            </then>
        </filter>
    </sequence>

```

Position: Ln 1, Ch 1 Total: Ln 119, Ch 5232

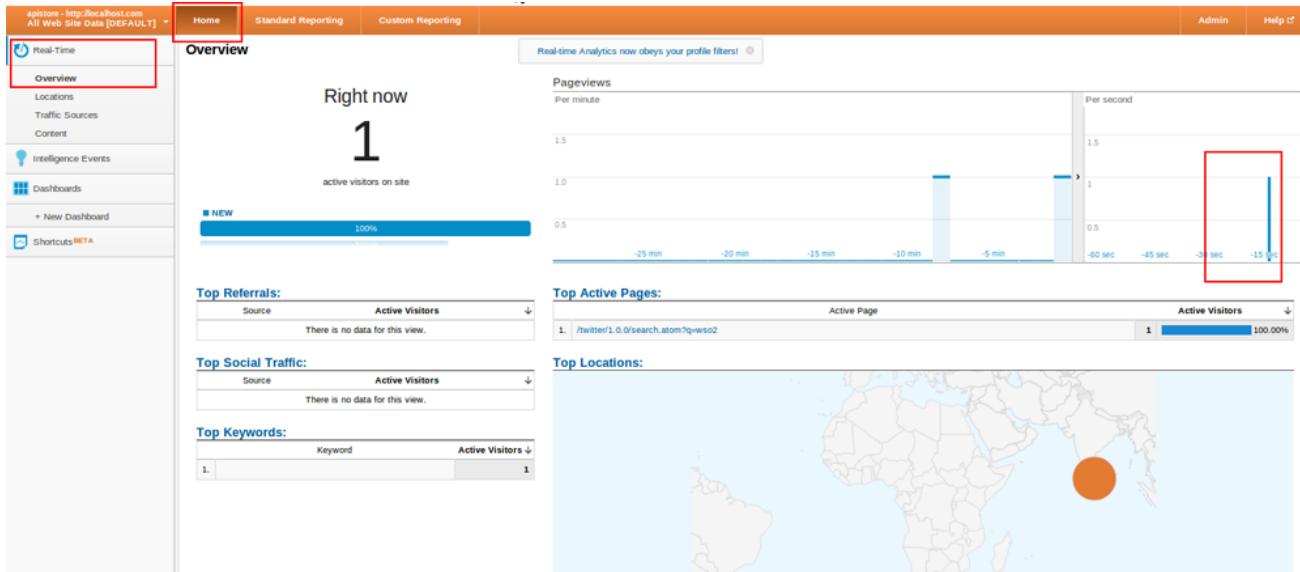
Toggle editor

**Update** **Reset**

6. API Manager is now integrated with Google Analytics. A user who has subscribed to a published API through the API Store should see an icon as Real-Time after logging into their Google Analytics account. Click on this icon and select **Overview**.
7. Invoke the above API using the embedded **WSO2 REST Client** (or any third-part rest client such as cURL).

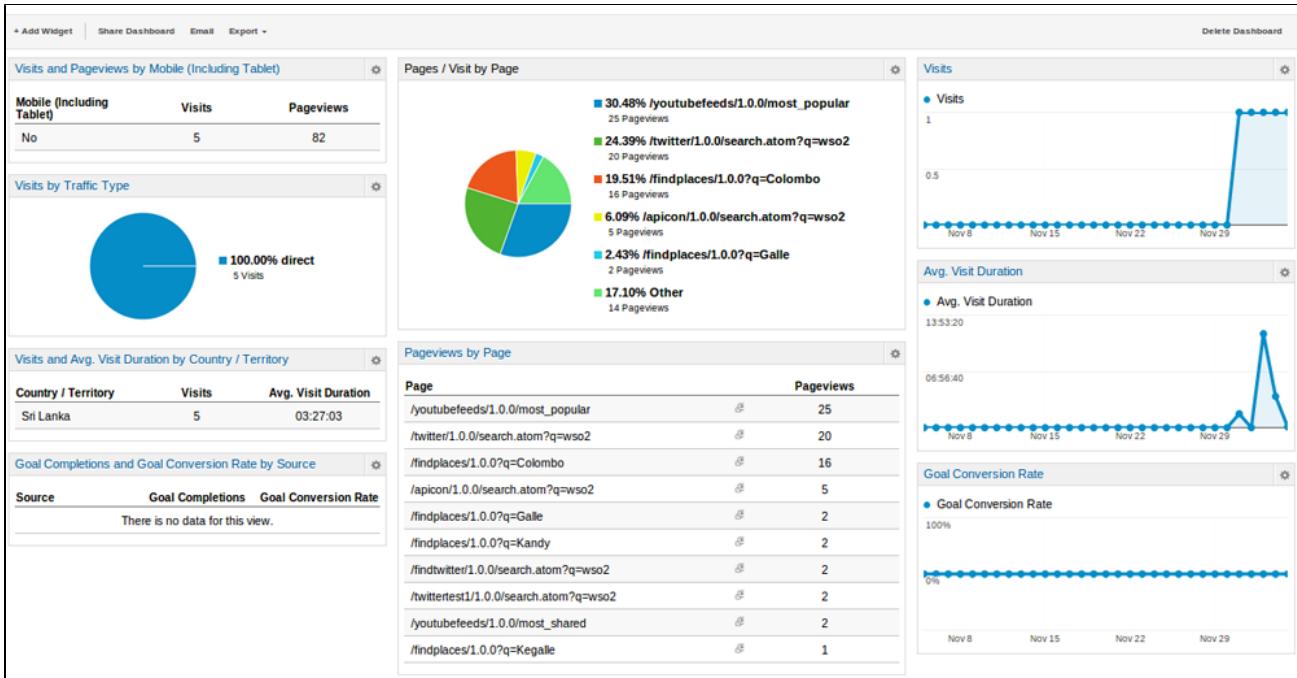
### Real-time statistics

8. This is one invocation of the API. Accordingly, Google Analytics graphs and statistics will be displayed at runtime. This example displays the **PageViews** per second graph and 1 user as active.



### Report statistics

Google analytics reporting statistics take more than 24 hours from the time of invocation to populate. Shown below is a sample Dashboard with populated statistics.



There are widgets with statistics related to Audience, Traffic, Page Content, Visit Duration etc. You can add any widget of your preference to dashboard.

## Viewing API Statistics

API statistics are provided in both API Publisher and API Store Web applications. Apart from the number of subscriptions per API, all other statistical dashboards require that an instance of WSO2 Business Activity Monitor (version 2.3.0 or above) is installed. For instructions to set up BAM, see [Publishing API Runtime Statistics](#). Once BAM is set up, follow the instructions below to view statistics through the API Publisher.

First, trigger some activities via the API gateway as explained in section [Browser-Based REST Clients](#) and wait a few seconds.

**⚠** The graphs you see on the API Manager statistical dashboards without setting up BAM are just samples and are not based on real runtime statistics of your server.

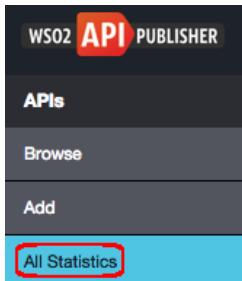
The sections below explain how to access the statistical dashboards:

- [API Publisher statistics](#)
- [API Store statistics](#)

### API Publisher statistics

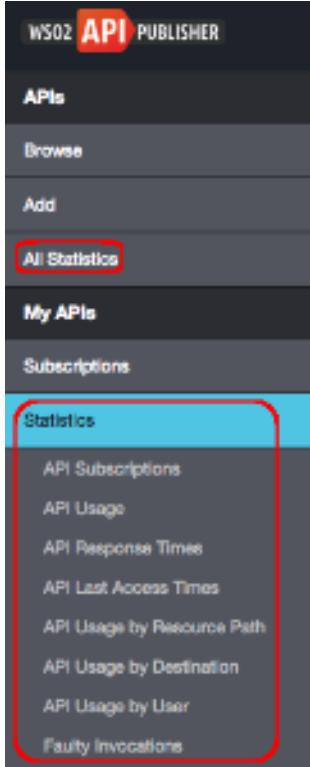
Log in to the API Publisher. If you have API creator and publisher privileges, the statistical menus that you see change as described below:

- If you have permission as publisher, the **All Statistics** menu will be visible in the left panel of the API Publisher.



- If you have permission to create APIs, in addition to the **All Statistics** menu, you also see the **Statistics** men

u in the left panel of the API Publisher. The latter shows stats specific to the APIs created by you.



- Anyone who can create and/or publish APIs can view API-level usage and subscription statistics by clicking on a selected API and referring to its **Versions** and **Users** tabs.

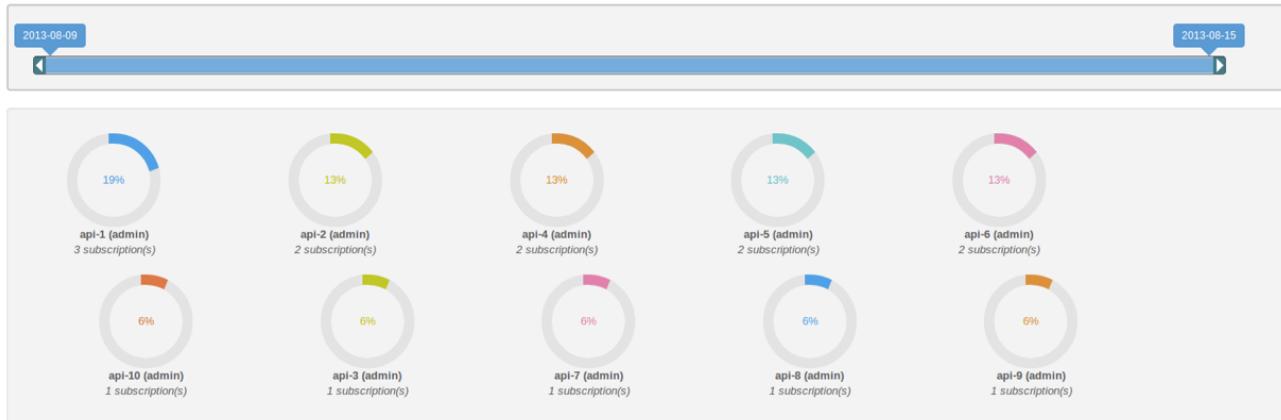
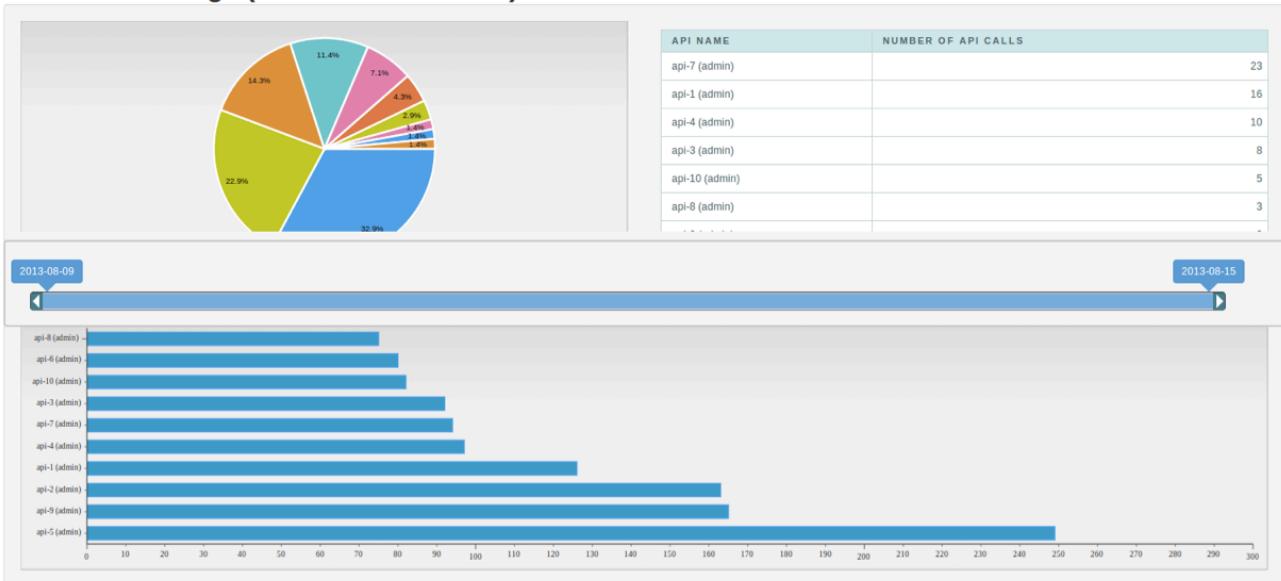
The screenshot shows the WSO2 API Publisher interface for a specific API named 'PhoneVerification-1.0.0'. The left sidebar has the same menu items as the first screenshot. The main area shows the API details for 'PhoneVerification - 1.0.0'. Below the title, there are tabs: Overview, Lifecycle, Versions (which is highlighted with a red box), Docs, and Users. The 'Versions' tab is active, showing a list of API versions.

Given below are the statistical dashboards that are available:

- **API Subscriptions:** Number of subscriptions per API (across all versions of an API)
- **API Usage:** Number of API calls being made per API (across all versions of an API)
- **API Response Times:**
- **API Last Access Times:** The subscribers who did the last 10 API invocations and the APIs/versions they invoked
- **API Usage by Resource Path:** Usage of an API and from which resource path (per API version)
- **API Usage by Destination:** To see destination-based usage tracking, you must first enable it. See [API Usage by Destination](#).
- **API Usage by User:** Number of times a user has accessed an API
- **Faulty Invocations:** The number of API invocations that failed to reach the endpoint per API per user. In a faulty API invocation, the message is mediated through the `fault` sequence. By default, the API Manager considers an API invocation to be faulty when the backend service is unavailable.

Several examples of usage and performance statistics are given below:

- API Subscriptions

**Overall API Subscriptions (Across All Versions)****API Usage****Overall API Usage (Across All Versions)****Last Access Times:****API Last Access Times (Across All Versions / Last 10 invocations)**

API	LAST ACCESSED VERSION	SUBSCRIBER	ACCESS TIME
api-5 (admin)	1.0.0	user2	8/15/2013 9:40:00 AM
api-6 (admin)	1.0.0	user2	8/15/2013 9:40:00 AM
api-10 (admin)	1.0.0	user2	8/15/2013 9:36:00 AM
api-7 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-8 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-9 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-1 (admin)	1.0.0	user2	8/15/2013 9:31:00 AM
api-4 (admin)	1.0.0	user2	8/15/2013 9:31:00 AM
api-3 (admin)	1.0.0	user1	8/15/2013 9:26:00 AM
api-2 (admin)	1.0.0	user1	8/15/2013 9:25:00 AM

**API Usage by Resource Path:**

## API Usage from Resource Path

api-10	1.0.0	/api10	POST		5
api-2	1.0.0	/api2	POST		2
api-3	1.0.0	/api3	POST		8
api-4	1.0.0	/api4	POST		10
api-5	1.0.0	/api5	POST		1
api-6	1.0.0	/api6	POST		1
api-7	1.0.0	/api7	POST		23
api-8	1.0.0	/api8	POST		3
api-9	1.0.0	/api9	POST		1

- API Usage by User:

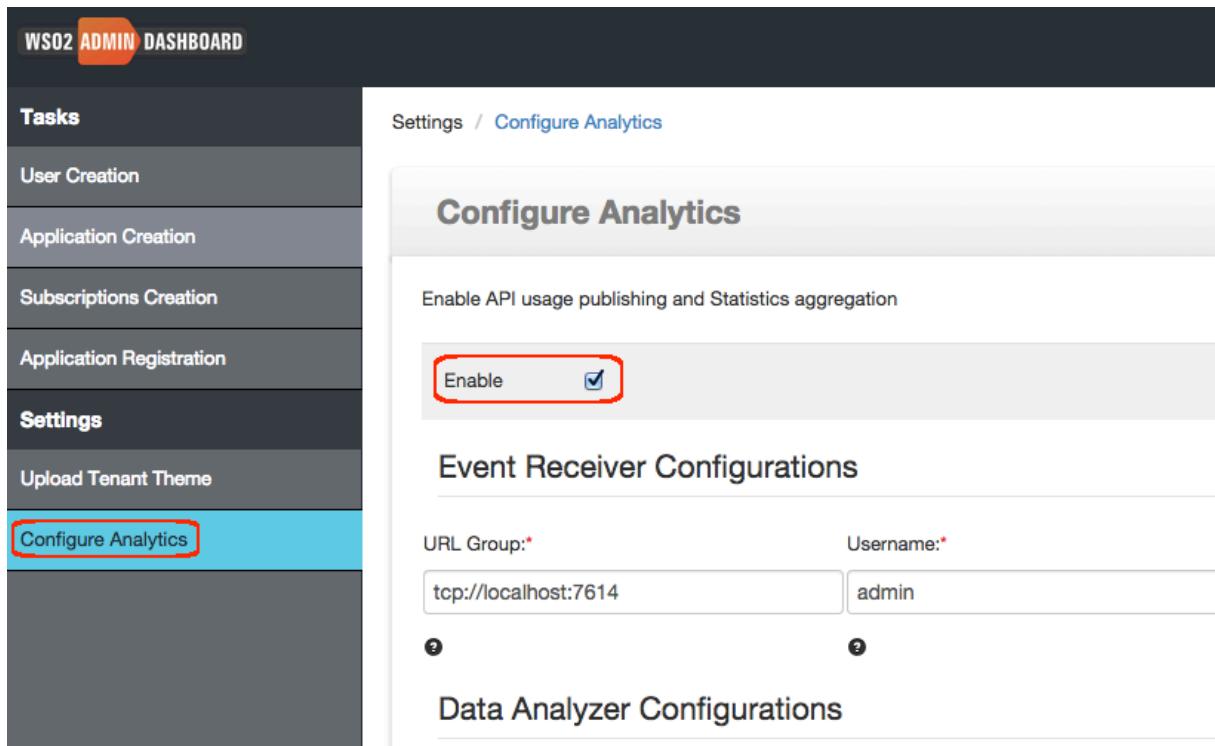
## API Usage By User

API	VERSION	USER	NUMBER OF ACCESS
api-7	1.0.0	user2	23
api-1	1.0.0	user1	13
api-3	1.0.0	user1	8
api-4	1.0.0	user2	7
api-10	1.0.0	user2	5
api-1	1.0.0	user2	3
api-4	1.0.0	user1	3
api-8	1.0.0	user2	3

- API Usage by Destination

An overview of the requests that leave the API Gateway to destination endpoints. It's particularly useful when the same API can reach different destinations such as load-balanced endpoints. This graph is not enabled by default. You must do it manually as follows:

1. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>).
2. Click the **Configure Analytics** menu, enable and configure API usage publishing and statistics.



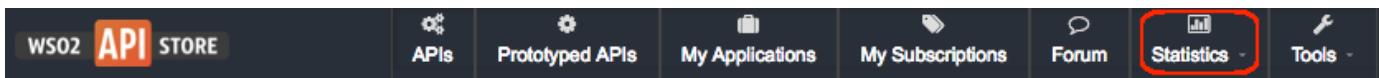
The screenshot shows the WSO2 Admin Dashboard. On the left, there's a sidebar with a 'Tasks' section containing 'User Creation', 'Application Creation', 'Subscriptions Creation', and 'Application Registration'. Below that is a 'Settings' section with 'Upload Tenant Theme' and 'Configure Analytics'. The 'Configure Analytics' option is highlighted with a red box. The main content area is titled 'Configure Analytics' and contains a sub-section 'Event Receiver Configurations'. It shows a checked checkbox labeled 'Enable'. Below it are fields for 'URL Group:' (tcp://localhost:7614) and 'Username:' (admin), each with a question mark icon. There's also a 'Data Analyzer Configurations' section.

3. When creating the API, enable the graph from the **Implement** tab of the API Publisher UI:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like APIs, Browse, Add, All Statistics, My APIs (which is selected), Subscriptions, Statistics, Tier Permissions, and Tier Permissions. The main area has a progress bar at the top with steps 1 Design, 2 Implement (which is highlighted with a red box), and 3 Manage. Below the progress bar, it says "PhoneVerification : /phoneverify/1.0.0". Under "Implementation Method", there are two radio buttons: "Backend Endpoint" (selected) and "Specify Inline". The "Endpoints" section contains fields for "Production Endpoint" (http://ws.cdyne.com/phoneverify) and "Sandbox Endpoint" (empty). There are "Advanced Options" and "Test" buttons for both. Below these, there are fields for "WSDL" and "WADL", each with a "Test URI" button. At the bottom of the endpoints section, there's a "Destination-Based Usage Tracking:" dropdown set to "Enabled", which is also highlighted with a red box.

### API Store statistics

Log in to the API Store. You can self subscribe to the store. Next, click the Statistics menu.

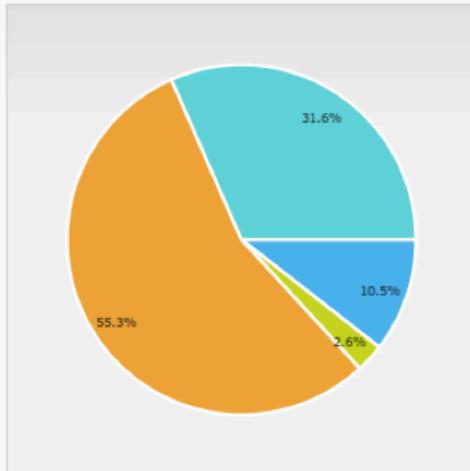


Given below are the statistical dashboards that are available:

- **API Usage per Application:**
- **Top Users per Application:** Users who make the most API invocations, per application
- **API Usage from Resource Path per Application:**
- **Faulty Invocations per Application:** Number of faulty API invocations, per application  
In a faulty API invocation, the message is mediated through the `fault` sequence. By default, the API Manager considers an API invocation to be faulty when the backend service is unavailable.

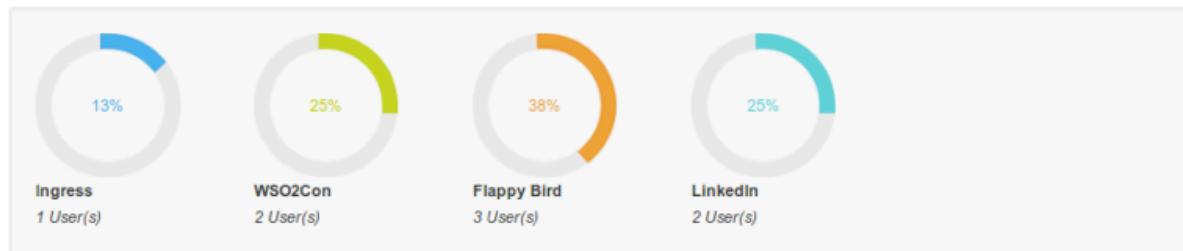
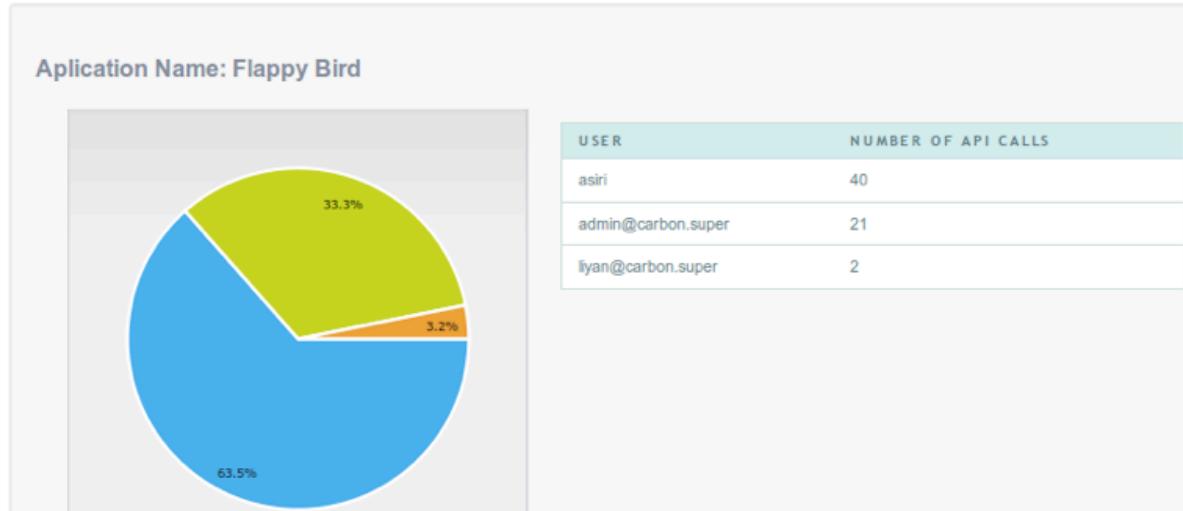
Several examples of usage and performance statistics are given below:

- API usage per application

**Application Name: WSO2Con**

API NAME	NO OF API CALLS
Foursquare	4
Google	1
facebook	21
tube	12

- Top Users:

**Registered Users For Applications****Top Users For Applications**

- API usage from resource path, per application

## API Usage from Resource Path

APPLICATION NAME	API NAME	API USAGE FROM RESOURCE PATH PER APPLICATION
Ingress	Foursquare	/top_rated (GET)
	Google	/most_popular (GET) /top_rated (GET)
	facebook	/top_rated (GET)
	tube	/top_rated (GET)
WSO2Con	Foursquare	/most_shared (GET)
	Google	/most_popular (GET)
	facebook	/most_popular (GET) /most_viewed (GET) /top_rated (GET)
	tube	/most_popular (GET)
Flappy Bird	Foursquare	/most_shared (GET)
	Google	/top_rated (GET)
	facebook	/most_viewed (GET)

- Faulty Invocations:

## Faulty Invocations per Application

APPLICATION NAME	API NAME	FAULTY API INVOCATION COUNT
Ingress	Foursquare	1
	Google	6
WSO2Con	Foursquare	4
	Google	1
Flappy Bird	Foursquare	8
	Google	6

## Admin Guide

The following topics explore various product deployment scenarios and other topics useful for system administrators.

- Deploying and Clustering the API Manager
- Tuning Performance
- Removing Unused Tokens from the Database
- Migrating the APIs to a Different Environment

### Deploying and Clustering the API Manager

In a typical production environment, you set up the different API Manager components (API Publisher, Gateway, Store and Key Manager) in separate servers so that you can scale them independently. You also install multiple instances of a component in a cluster to ensure proper load balancing. When one node becomes unavailable or is experiencing high traffic, another node handles the requests.

- For information on clustering, see [Clustering WSO2 API Manager](#).
- For information on deployment patterns, see [Deployment Patterns of WSO2 API Manager](#).

### Tuning Performance

This section describes some recommended performance tuning configurations to optimize the API Manager. It assumes that you have set up the API Manager on Unix/Linux, which is recommended for a production deployment. We also recommend a [distributed API Manager setup](#) for most production systems. Out of all components of an API Manager distributed setup, the API Gateway is the most critical, because it handles all inbound calls to APIs. Therefore, we recommend you to have at least a 2-node cluster of API Gateways in a distributed setup.

- OS-level settings
- JVM-level settings
- WSO2 Carbon platform-level settings
- APIM-level settings



#### Important:

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The values we discuss here are general recommendations. They might not be the optimal values for the specific hardware configurations in your environment. We recommend you to carry out load tests on your environment to tune the API Manager accordingly.

#### *OS-level settings*

1. To optimize network and OS performance, configure the following settings in `/etc/sysctl.conf` file of Linux. These settings specify a larger port range, a more effective TCP connection timeout value, and a number of other important parameters at the OS-level.



It is not recommended to use `net.ipv4.tcp_tw_recycle = 1` when working with network address translation (NAT), such as if you are deploying products in EC2 or any other environment configured with NAT.

```

net.ipv4.tcp_fin_timeout = 30
fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288
net.core.wmem_default = 524288
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65535

```

2. To alter the number of allowed open files for system users, configure the following settings in /etc/security/limits.conf file of Linux (be sure to include the leading \* character).

```

* soft nofile 4096
* hard nofile 65535

```

Optimal values for these parameters depend on the environment.

3. To alter the maximum number of processes your user is allowed to run at a given time, configure the following settings in /etc/security/limits.conf file of Linux (be sure to include the leading \* character). Each carbon server instance you run would require upto 1024 threads (with default thread pool configuration). Therefore, you need to increase the nproc value by 1024 per each carbon server (both hard and soft).

```

* soft nproc 20000
* hard nproc 20000

```

### **JVM-level settings**

If one or more worker nodes in a clustered deployment require access to the management console, increase the entity expansion limit as follows in the <APIM\_HOME>/bin/wso2server.bat file (for Windows) or the <APIM\_HOME>/bin/wso2server.sh file (for Linux/Solaris). The default entity expansion limit is 64000.

```
-DentityExpansionLimit=10000
```

### **WSO2 Carbon platform-level settings**

In multitenant mode, the WSO2 Carbon runtime limits the thread execution time. That is, if a thread is stuck or taking a long time to process, Carbon detects such threads, interrupts and stops them. Note that Carbon prints the current stack trace before interrupting the thread. This mechanism is implemented as an Apache Tomcat valve. Therefore, it should be configured in the <PRODUCT\_HOME>/repository/conf/tomcat/catalina-server.xml file as shown below.

```

<Valve className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve"
threshold="600" />

```

- The className is the Java class used for the implementation. Set it to org.wso2.carbon.tomcat.ext.

- `valves.CarbonStuckThreadDetectionValve`.
- The threshold gives the minimum duration in seconds after which a thread is considered stuck. The default value is 600 seconds.

#### APIM-level settings

Improvement Area	Performance Recommendations										
API Gateway nodes	<p>Increase memory allocated by modifying <code>/bin/wso2server.sh</code> with the following setting:</p> <ul style="list-style-type: none"> <li><code>-Xms2048m -Xmx2048m -XX:MaxPermSize=1024m</code></li> </ul>										
NHTTP transport of API Gateway	<p>Recommended values for <code>&lt;AM_HOME&gt;/repository/conf/nhttp.properties</code> file are given below. <b>Note</b> that the commented out values in this file are the default values that will be applied if you do not change anything.</p> <p><b>Property descriptions:</b></p> <table border="1"> <tbody> <tr> <td><code>snd_t_core</code></td><td>Transport sender worker pool's initial thread count</td></tr> <tr> <td><code>snd_t_max</code></td><td>Transport sender worker pool's maximum thread count</td></tr> <tr> <td><code>snd_io_threads</code></td><td>Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.</td></tr> <tr> <td><code>snd_alive_sec</code></td><td>Sender-side keep-alive seconds</td></tr> <tr> <td><code>snd_qlen</code></td><td>Sender queue length, which is infinite by default</td></tr> </tbody> </table> <p><b>Recommended values:</b></p> <p># HTTP Sender thread pool parameters</p> <ul style="list-style-type: none"> <li><code>snd_t_core=200</code></li> <li><code>snd_t_max=250</code></li> <li><code>snd_alive_sec=5</code></li> <li><code>snd_qlen=-1</code></li> <li><code>snd_io_threads=16</code></li> </ul> <p># HTTP Listener thread pool parameters</p> <ul style="list-style-type: none"> <li><code>lst_t_core=200</code></li> <li><code>lst_t_max=250</code></li> <li><code>lst_alive_sec=5</code></li> <li><code>lst_qlen=-1</code></li> <li><code>lst_io_threads=16</code></li> </ul> <p>#timeout parameters</p> <ul style="list-style-type: none"> <li><code>http.socket.timeout.receiver</code>: Recommended socket timeout for listener is 120000</li> <li><code>http.socket.timeout.sender</code>: Recommended socket timeout for sender is 120000</li> </ul>	<code>snd_t_core</code>	Transport sender worker pool's initial thread count	<code>snd_t_max</code>	Transport sender worker pool's maximum thread count	<code>snd_io_threads</code>	Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.	<code>snd_alive_sec</code>	Sender-side keep-alive seconds	<code>snd_qlen</code>	Sender queue length, which is infinite by default
<code>snd_t_core</code>	Transport sender worker pool's initial thread count										
<code>snd_t_max</code>	Transport sender worker pool's maximum thread count										
<code>snd_io_threads</code>	Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.										
<code>snd_alive_sec</code>	Sender-side keep-alive seconds										
<code>snd_qlen</code>	Sender queue length, which is infinite by default										

PassThrough transport of API Gateway	<p>Recommended values for &lt;AM_HOME&gt;/repository/conf/passthru-http.properties file are given below. <b>Note</b> that the commented out values in this file are the default values that will be applied if you do not change anything.</p> <p><b>Property descriptions</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="padding: 5px;">worker_thread_keepalive_sec</td><td style="padding: 5px;">Defines the keep-alive time for extra threads in the worker pool</td></tr> <tr> <td style="padding: 5px;">worker_pool_queue_length</td><td style="padding: 5px;">Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool</td></tr> <tr> <td style="padding: 5px;">io_threads_per_reactor</td><td style="padding: 5px;">Defines the number of IO dispatcher threads used per reactor</td></tr> <tr> <td style="padding: 5px;">http.max.connection.per.host.port</td><td style="padding: 5px;">Defines the maximum number of connections per host port</td></tr> <tr> <td style="padding: 5px;">worker_pool_queue_length</td><td style="padding: 5px;">Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.</td></tr> </tbody> </table> <p><b>Recommended values</b></p> <ul style="list-style-type: none"> <li>• worker_thread_keepalive_sec: Default value is 60s. This should be less than the socket timeout.</li> <li>• worker_pool_queue_length: Set to -1 to use an unbounded queue. If a bound queue is used and the queue gets filled to its capacity, any further attempts to submit jobs will fail, causing some messages to be dropped by Synapse. The thread pool starts queuing jobs when all the existing threads are busy and the pool has reached the maximum number of threads. So, the recommended queue length is -1.</li> <li>• io_threads_per_reactor: Value is based on the number of processor cores in the system (Runtime.getRuntime().availableProcessors())</li> <li>• http.max.connection.per.host.port : Default value is 32767, which works for most systems but you can tune it based on your operating system (for example, Linux supports 65K connections)</li> <li>• worker_pool_size_core: 400</li> <li>• worker_pool_size_max: 500</li> <li>• io_buffer_size: 16384</li> <li>• http.socket.timeout: 60000</li> <li>• snd_t_core: 200</li> <li>• snd_t_max: 250</li> <li>• snd_io_threads: 16</li> <li>• lstd_t_core: 200</li> <li>• lstd_t_max: 250</li> <li>• lstd_io_threads: 16</li> </ul> <p>Make the number of threads equal to the number of processor cores.</p>	worker_thread_keepalive_sec	Defines the keep-alive time for extra threads in the worker pool	worker_pool_queue_length	Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool	io_threads_per_reactor	Defines the number of IO dispatcher threads used per reactor	http.max.connection.per.host.port	Defines the maximum number of connections per host port	worker_pool_queue_length	Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.
worker_thread_keepalive_sec	Defines the keep-alive time for extra threads in the worker pool										
worker_pool_queue_length	Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool										
io_threads_per_reactor	Defines the number of IO dispatcher threads used per reactor										
http.max.connection.per.host.port	Defines the maximum number of connections per host port										
worker_pool_queue_length	Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.										

Time-out configurations	<p>The API Gateway routes the requests from your client to an appropriate endpoint. The most common reason for your client getting a timeout is when the Gateway's timeout is larger than the client's timeout values. You can resolve this by either increasing the timeout on the client's side or by decreasing it on the API Gateway's side.</p> <p>Here are few parameters, in <b>addition to the timeout parameters discussed in the previous sections</b>.</p> <table border="1" data-bbox="326 369 1516 1134"> <tr> <td data-bbox="326 369 734 665">synapse.global_timeout_interval</td><td data-bbox="734 369 1516 665"> <p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the &lt;APIM_HOME&gt;/repository/conf/synapse.properties file. Recommended value is 120000.</p> </td></tr> <tr> <td data-bbox="326 665 734 1134">Endpoint-level timeout</td><td data-bbox="734 665 1516 1134"> <p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> &lt;timeout&gt;     &lt;duration&gt;10000&lt;/duration&gt;     &lt;responseAction&gt;fault&lt;/responseAction&gt; &lt;/timeout&gt;</pre> </td></tr> </table>	synapse.global_timeout_interval	<p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the &lt;APIM_HOME&gt;/repository/conf/synapse.properties file. Recommended value is 120000.</p>	Endpoint-level timeout	<p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> &lt;timeout&gt;     &lt;duration&gt;10000&lt;/duration&gt;     &lt;responseAction&gt;fault&lt;/responseAction&gt; &lt;/timeout&gt;</pre>
synapse.global_timeout_interval	<p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the &lt;APIM_HOME&gt;/repository/conf/synapse.properties file. Recommended value is 120000.</p>				
Endpoint-level timeout	<p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> &lt;timeout&gt;     &lt;duration&gt;10000&lt;/duration&gt;     &lt;responseAction&gt;fault&lt;/responseAction&gt; &lt;/timeout&gt;</pre>				

**Key Manager nodes**

Set the following in <APIM\_HOME>/repository/conf/axis2/axis2\_client.xml file:

```
<parameter name="defaultMaxConnPerHost">1000</parameter>
<parameter name="maxTotalConnections">30000</parameter>
```

Set the MySQL maximum connections:

```
mysql> show variables like "max_connections";
max_connections was 151
set to global max_connections = 250;
```

Set the open files limit to 200000 by editing the /etc/sysctl.conf file:

```
sudo sysctl -p
```

Set the following in <APIM\_HOME>/repository/conf/tomcat/catalina-server.xml file.

```
maxThreads="750"
minSpareThreads="150"
disableUploadTimeout="false"
enableLookups="false"
connectionUploadTimeout="120000"
maxKeepAliveRequests="600"
acceptCount="600"
```

Set the following connection pool elements in <APIM\_HOME>/repository/conf/datasource/master-datasources.xml file:

```
<maxActive>50</maxActive>
<maxWait>60000</maxWait>
<testOnBorrow>true</testOnBorrow>
<validationQuery>SELECT 1</validationQuery>
<validationInterval>30000</validationInterval>
```

**Note** that you set the <testOnBorrow> element to true and provide a validation query (e.g., i Oracle, SELECT 1 FROM DUAL), which is run to refresh any stale connections in the connectio pool. Set a suitable value for the <validationInterval> element, which defaults to 3000 milliseconds. It determines the time period after which the next iteration of the validation query wi be run on a particular connection. It avoids excess validations and ensures better performance.

## Removing Unused Tokens from the Database

As you use WSO2 API Manager, the number of revoked, inactive and expired tokens accumulates in the IDN\_OAUTH2\_ACCESS\_TOKEN table. These tokens are kept in the database for logging and audit purposes, but they can have a negative impact on the server's performance over time. Therefore, it is recommended to clean them periodically as given in the instructions below:

1. Take a backup of the running database.
2. Make the production database read-only so that users cannot create new tokens while the cleanup process is in progress.

3. Set up the database dump in a test environment and test it for any issues.

 **Tip:** We recommend you to test the database dump before the cleanup task as the cleanup can take some time.

4. Run the following scripts on the database dump. It takes a backup of the necessary tables, turns off SQL updates and cleans the database of unused tokens.

```

USE 'WSO2AM_DB';
DROP PROCEDURE IF EXISTS 'cleanup_tokens';

DELIMITER $$
CREATE PROCEDURE 'cleanup_tokens' ()
BEGIN

-- Backup IDN_OAUTH2_ACCESS_TOKEN table
DROP TABLE IF EXISTS 'IDN_OAUTH2_ACCESS_TOKEN_BAK';
CREATE TABLE 'IDN_OAUTH2_ACCESS_TOKEN_BAK' AS SELECT * FROM
'IDN_OAUTH2_ACCESS_TOKEN';

-- 'Turn off SQL_SAFE_UPDATES'
SET @OLD_SQL_SAFE_UPDATES = @@SQL_SAFE_UPDATES;
SET SQL_SAFE_UPDATES = 0;

-- 'Keep the most recent INACTIVE key for each CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'INACTIVE';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'INACTIVE') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'INACTIVE') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y);

SELECT 'AFTER:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN WHERE
TOKEN_STATE = 'INACTIVE';

-- 'Keep the most recent REVOKED key for each CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_REVOKED_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN WHERE
TOKEN_STATE = 'REVOKED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'REVOKED') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'REVOKED') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y);

```

```
SELECT 'AFTER:TOTAL_REVOKED_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN WHERE
TOKEN_STATE = 'REVOKED';

-- 'Keep the most recent EXPIRED key for each CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN WHERE
TOKEN_STATE = 'EXPIRED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'EXPIRED') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT ACCESS_TOKEN FROM (SELECT
ACCESS_TOKEN, CONSUMER_KEY, AUTHZ_USER, TOKEN_SCOPE FROM IDN_OAUTH2_ACCESS_TOKEN
WHERE TOKEN_STATE = 'EXPIRED') x GROUP BY CONSUMER_KEY, AUTHZ_USER,
TOKEN_SCOPE)y);

SELECT 'AFTER:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM IDN_OAUTH2_ACCESS_TOKEN WHERE
TOKEN_STATE = 'EXPIRED';

-- 'Restore the original SQL_SAFE_UPDATES value'
SET SQL_SAFE_UPDATES = @OLD_SQL_SAFE_UPDATES;
```

```
END$$
DELIMITER ;
```

5. Once the cleanup is over, start the API Manager pointing to the cleaned-up database dump and test thoroughly for any issues.  
You can also schedule a cleanup task that will be automatically run after a given period of time. Here's an example:

```
USE 'WSO2AM_DB';
DROP EVENT IF EXISTS 'cleanup_tokens_event';
CREATE EVENT 'cleanup_tokens_event'
    ON SCHEDULE
        EVERY 1 WEEK STARTS '2015-01-01 00:00:00'
    DO
        CALL 'WSO2AM_DB'.'cleanup_tokens'();

-- 'Turn on the event_scheduler'
SET GLOBAL event_scheduler = ON;
```

## Migrating the APIs to a Different Environment

When you migrate the API Manager from one server environment to another, you must move all created APIs, so that you do not have to create them again in the new environment. The sections below explain how to do this:

- Deploying the API import/export tool
- Exporting an API
- Importing an API
- Understanding the API import/export tool

### Deploying the API import/export tool

1. Download WSO2 API Manager 1.9.0 from <http://wso2.com/products/api-manager/>.
2. Download WSO2 API import/export tool from the table below. If you are a new user, it is recommended to download the latest version.

Version	File
v0.9.1	<a href="#">Download</a>
v0.9.0	<a href="#">Download</a>

3. Copy the downloaded api-import-export-<version>.war file to <APIM\_HOME>/repository/deployment/server/webapps.
4. Start the API Manager. If the server is already started, the file will be automatically deployed as hot deployment is enabled.

### Exporting an API

After successfully deploying the import/export tool, you can export an existing API as a .zip archive. Issue the following cURL command using the command line:

```
curl -H "Authorization:Basic <base64-encoded-credentials-separated-by-a-colon>"  
-X GET  
"https://<APIM_HOST:Port>/api-import-export-<version>/export-api?name=<API-name>&versi  
on=<API-version>  
&provider=<API-provider>" -k > <exportedApiName>.zip
```

Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -X GET  
"https://10.100.7.40:9443/api-import-export-<version>/export-api?name=WeatherAPI&versi  
on=1.0.0&provider=admin"  
-k > myExportedAPI.zip
```

## Importing an API

You can use the archive created in the previous section to import APIs to an API Manager instance.

1. Make sure the API Manager is started and the import/export tool is deployed.
2. Run the following cURL command:

```
curl -H "Authorization:Basic  
<base64-encoded-username-and-password-separated-by-a-colon>"  
-F file=@"/full/path/to/the/zip/file" -k -X POST  
"https://<APIManagerHost:Port>/api-import-export-<version>/import-api"
```

Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -F  
file=@"/Desktop/MyAPIFolder/myExportedAPI.zip" -k -X POST  
"https://10.100.7.40:9443/api-import-export-<version>/import-api"
```

 You must add a parameter named **preserveProvider** to the cURL command and set its value to false if the API is imported to a different domain than its exported one. This parameter sets the provider of the imported API to the user who is issuing the cURL command. Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -F  
file=@"/Desktop/MyAPIFolder/myExportedAPI.zip" -k -X POST  
"https://10.100.7.40:9443/api-import-export-<version>/import-api?preserveP  
rovider=false"
```

## Understanding the API import/export tool

The API import/export tool uses a RESTful API, protected by basic authentication. Only users with admin privileges are allowed to access it in the initial phase. To allow access to the import/export feature only for a particular tenant, log in to WSO2 API Manager's Management Console and add the downloaded archive file as a Web application to the server.

### ***The export functionality***

API export functionality retrieves the information required for the requested API from the registry and databases and generates a zip file, which the exporter can download. This exported ZIP file has the following structure:

```
<APIName>--<version>
|_ Meta Information
    |_ api.json
    |_ swagger.json
|_ Documents
    |_ docs.json
    |_ documents with type 'file'
|_ Image
    |_ icon.<extension>
|_ WSDL
    |_ <ApiName>--<version>.wsdl
|_ Sequences
    |_ In Sequence
        |_<Sequence Name>.xml
    |_ Out Sequence
        |_<Sequence Name>.xml
    |_ Fault Sequence
        |_<Sequence Name>.xml
```

The structure of the ZIP file is explained below:

Sub directory/File	Description
Meta Information	<ul style="list-style-type: none"> <li>api.json: contains all the basic information required for an API to be imported in another environment</li> <li>swagger.json: contains the API swagger definition</li> </ul>
Documents	<ul style="list-style-type: none"> <li>docs.json: contains the summary of all the documents available for the API</li> <li>Add the uploaded files for API documentation also</li> </ul>
Image	Thumbnail image of the API
WSDL	WSDL file of the API
Sequences	The sequences available for the API

Given below is the RESTful API for export functionality. It is secured using Basic Authentication.

Parameter	Description
URI	<a href="https://&lt;host name&gt;:9443/api-import-export-&lt;version&gt;/export-api">https://&lt;host name&gt;:9443/api-import-export-&lt;version&gt;/export-api</a>
Query parameters	name=<api_name>&version=<api_version>&provider=<provider_name>
HTTP method	GET

<p><b>Examples</b></p> <pre>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://10.100.7.39:9443/api-import-export-&lt;version&gt;/export-api? name=test&amp;version=1.0.0&amp;provider=admin"</pre> <p>It gives a data stream as the output. To download it as a zipped archive, use the following command:</p> <pre>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://10.100.7.39:9443/api-import-export-&lt;version&gt;/export-api?name= test &amp;version=1.0.0&amp;provider=admin" -k &gt; exportedApi.zip</pre> <p>To verify the output status of the API call:</p> <pre>curl -v -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://10.100.7.39:9443/api-import-export-&lt;version&gt;/export-api?name= test&amp;version=1.0.0&amp;provider=admin" -k &gt; exportedApi.zip</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### The import functionality

The import functionality uploads the exported ZIP file of the API to the target environment. It creates a new API with all the registry and database resources exported from the source environment. Note the following:

- The lifecycle status of an imported API will always be CREATED even when the original API in the source environment has a different state. This is to enable the importer to modify the API before publishing it.
- Tiers and sequences are provider-specific. If an exported tier is not already available in the imported environment, that tier will not be added to the new environment. However, if an exported sequence is not available in the imported environment, it will be added.
- The importer can decide whether to keep the original provider's name or replace it. Set the `preserveProvider` parameter to true to keep it. If you set it to false, the original provider will be replaced by the user who is sending the cURL command.
- Cross-tenant imports are not allowed by preserving the original provider. For example, if an API is exported from tenant A and imported to tenant B, the value of the `preserveProvider` parameter must always be false.

Given below is the RESTful API for import functionality.

Parameter	Description
URI	<a href="https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api">https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api</a>
Query parameters	<code>preserveProvider=&lt;true false&gt;</code>
HTTP method	POST
Example	<p>Imports the API with the original provider preserved: curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -F file=@"full/path/to/the/zip/file" -k -X POST "<a href="https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api">https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api</a>"</p> <p>Imports the API with the provider set to the current user: curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -F file=@"full/path/to/the/zip/file" -k -X POST "<a href="https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api?preserveProvider=false">https://10.100.7.40:9443/api-import-export-&lt;version&gt;/import-api?preserveProvider=false</a>"</p>

## Published APIs

The following topics discuss the APIs exposed from the API Publisher and API Store Web applications using which you can create and manage APIs. You can consume APIs directly through their UIs or an external REST client like cURL or the [WSO2 REST client](#). The Token APIs exposed in API Manager are also described here.

- [Publisher APIs](#)
- [Store APIs](#)
- [Token API](#)
- [WSO2 Admin Services](#)

### Publisher APIs

Publisher APIs provide the following REST resources.

[ [Login](#) ] [ [Logout](#) ] [ [Add API](#) ] [ [Update API](#) ] [ [Get All APIs](#) ] [ [Get an API](#) ] [ [Remove an API](#) ] [ [Copy an API](#) ] [ [Check Older Version](#) ] [ [Change API Status](#) ] [ [Add/Update an API Document](#) ] [ [Remove an API Document](#) ] [ [Get all Throttling Tiers](#) ] [ [Check if API Exists](#) ] [ [Validate Roles](#) ]



**Note:** When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

The responses is a JSON message.

#### Login

Description	Log in to API Publisher web application.
URI	<a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a>
URI Parameters	action=login&username=xxx&password=xxx
HTTP Methods	POST
Example	curl -X POST -c cookies <a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a> -d 'action=login&username=admin&password=admin'

#### Logout

Description	Log out from API Publisher web application.
URI	<a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a>
URI Parameters	?action=logout
HTTP Methods	GET
Example	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a> ?action=logout

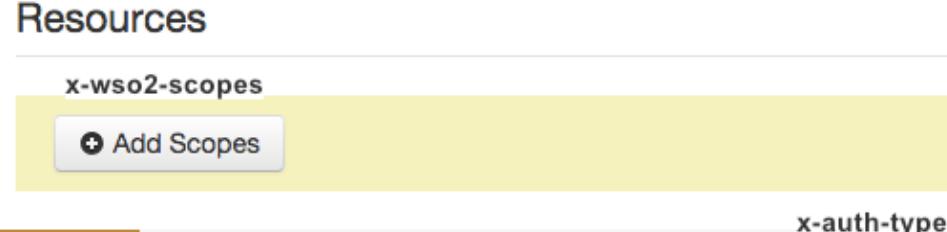
#### Add API

Description	Add a new API.
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>

## URI Parameters

Given below are the parameters that you can pass with an Add-API call. Mandatory ones are marked

Parameter name	Syntax
Action*	action=addAPI
Name*	name=xxx
Context*	context=/xxx
Version*	version=x.x.x
API visibility	visibility=<public private restricted> The default is public. If you select restricted, mention to which roles as follows: You can read more about <b>API visibility</b> from <a href="#">here</a> .
Thumbnail image	<ul style="list-style-type: none"> <li>To add a thumbnail image as a file object, create the object and pass it with the</li> <li>To add a thumbnail image as a URL of the image, pass the URL with the thumb</li> </ul>
Description	description=xxx
Tags	tags=x,y,z
Resources*	resourceCount=0&resourceMethod-0=GET&resourceMethodAuthType-0=Applicatio <ul style="list-style-type: none"> <li><b>resourceMethod</b> can take any one of the following values: GET, POST, DELE</li> <li><b>resourceMethodAuthType</b> can take any one of the following values: Applicat</li> <li><b>resourceMethodThrottlingTier</b> can take any one of the following default v</li> </ul> tem/governance/apimgt/applicationdata/tiers.xml registry location

Resources as Swagger	<p>Instead of adding resources directly as above, you can add resources, including scc</p> <pre>swagger={"paths": {"/CheckPhoneNumber": {"post": {"x-auth-type": "Unlimited", "responses": {"200": {}}, "get": {"x-auth-type": "Unlimited", "responses": {"200": {}}, "parameters": [{"name": "PhoneNumber", "paramType": "query"}, {"name": "LicenseKey", "paramType": "query", "required": true}], "put": {"responses": {"200": {}}, "get": {"responses": [{"description": "", "name": "read_number", "roles": "admin", "x-wso2-scopes": [{"key": "read_number"}]}]}}}}}}</pre>
	<p>In the above code, note that you have one resource path defined with the URL pattern /CheckPhoneNumber. You can have multiple similar resource paths to a single API and no need to define them separately.</p> <p>For more information of the Swagger objects used in this example, see the <a href="#">Swagger API Reference</a>.</p> <ul style="list-style-type: none"> <li>• <b>x-wso2-scopes:</b> The list of scope elements that you want to define. Each element has the following properties: <ul style="list-style-type: none"> <li>• <b>description:</b> Scope description</li> <li>• <b>roles:</b> Allowed roles</li> <li>• <b>name:</b> Scope Name</li> <li>• <b>key:</b> Scope Key</li> </ul> </li> <li>• <b>x-auth-type:</b> Authentication type of the method.</li> <li>• <b>x-throttling-tier:</b> Throttling tier of the method.</li> <li>• <b>x-scope:</b> OAuth scope of the method. This must be one of the list of element you defined in the x-wso2-scopes object.</li> </ul> <p>The following image shows the WSO2-specific parameters we describe here. Also see the <a href="#">API Resources UI</a>.</p>  <p>The screenshot shows the 'Resources' section of the WSO2 API Manager interface. A yellow header bar contains the text 'x-wso2-scopes' and a button labeled '+ Add Scopes'. Below this, a table row for a 'PUT' method at the URL '/CheckPhoneNumber' includes a 'Summary' link. To the right of the URL, the 'x-auth-type' column is set to 'Application &amp; User'.</p>
Endpoints*	<p>This example adds an HTTP production endpoint: <code>endpoint_config={"production_endpoints": [{"url": "http://server1.int.com", "config": null}], "actionSelect": "fault", "actionDuration": 60000}</code>, "endpoint_type": "Production".</p> <p>To give advanced endpoint configurations, add the JSON implementation inside "config" field.</p> <p>You add sandbox endpoints in the same way. The only difference is that instead of "actionSelect": "fault", "actionDuration": 60000, you add "actionSelect": "return", "actionDuration": 60000.</p> <p>If you want to add other types of endpoints, follow the examples below. <b>Note</b> that the examples below are for WSO2 API Manager 1.9.0 and later versions.</p> <ul style="list-style-type: none"> <li>• <b>For address endpoints:</b> <code>endpoint_config={"production_endpoints": [{"url": "http://server1.int.com", "config": null}], "actionSelect": "fault", "actionDuration": 60000}</code></li> <li>• <b>For failover endpoints:</b> <code>endpoint_config={"production_endpoints": [{"url": "http://server1.int.com", "config": null}, {"url": "http://failover2.endpoint"}], "actionSelect": "fault", "actionDuration": 60000}</code></li> <li>• <b>For load balanced endpoints:</b> <code>endpoint_config={"production_endpoints": [{"url": "http://server1.int.com", "config": null}], "algoCombo": "org.apache.synapse.endpoints.algorithms.RoundRobin", "sessionManagement": "simpleClientSession", "actionSelect": "fault", "actionDuration": 60000}</code></li> </ul>
Endpoint security scheme	<p><code>endpointType=&lt;secured nonsecured&gt;</code></p> <p>The default is non-secured but if you select 'secured', you must pass the credentials.</p>

Make default version	To mark this version of the API as the <b>default version</b> from a group of versions, give the <b>Default Version</b> option means that you make this version the default in a group. For example, if you mark <a href="http://host:port/youtube/2.0">http://host:port/youtube/2.0</a> as the default version when the API has two URLs. If you mark any version of an API as the default, you get two API URLs in its <b>Overview</b> both URLs. If you mark an unpublished API as the default, the previous default, published API will be unpublished.
Tier Availability*	tiersCollection=<Gold,Silver,Bronze,Unlimited>
Transports	http_checked=http&https_checked=https Both are selected by default. If you want to set only the <b>HTTP transport</b> , leave the <b>https</b> checked.
Sequences	If you want to engage a custom sequence to the API, give <b>inSequence=&lt;sequence name&gt;</b> in the registry.
Response caching	responseCache=<enabled disabled> It is disabled by default but if you enable it, pass the response cache timeout as follows: <code>responseCache=enabled&amp;cacheTime=10000</code> See <a href="#">Configuring Caching</a> for more information.
Subscriptions	By default, subscription is allowed to the current tenant only. Add the argument <code>subscriptions=all_tenants</code> to <b>enable subscriptions</b> to this tenant <tenant name>. For example, <code>&amp;subscriptions=all_tenants</code> . See <a href="#">API visibility and subscription</a> for more information.
Business information	Add a section like this: <code>bizOwner=&lt;name&gt;&amp;bizOwnerMail=&lt;e-mail address&gt;</code>
HTTP Methods	POST
Example	<code>curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a> -d "action=add&amp;name=John Doe&amp;phone number&amp;tags=phone,mobile,multimedia&amp;endpointType=nonsecured&amp;tiersCollection=Gold,Bronze,Unlimited&amp;resourceMethodThrottlingTier-0=Unlimited&amp;uriTemplate-0=/*&amp;default_version_check=1&amp;endpoint_config={"production_endpoints": {"url": "http://ws.cdyne.com/phonverify/phonverify.asmx", "method": "GET", "headers": [{"name": "Content-Type", "value": "application/json"}], "body": {"method": "POST", "path": "/phonverify", "parameters": [{"name": "phone", "value": "1234567890"}]}}, "headers": [{"name": "Content-Type", "value": "application/json"}]}]</code>

## Update API

Description	Update an existing API
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
URI Parameters	<b>Parameters are same as in Add API</b> except that <b>action=updateAPI</b> and you can only update the following parameters: name, tags, endpointType, endpoint_config (can change the endpoint URL etc.,) http_checked, https_checked, add new resources. See example below.
HTTP Methods	POST

Example	<b>Update API :</b> curl -X POST -b cookies <a "http:="" a="" api="" feeds="" gdata.youtube.com="" href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.json&amp;provider=admin&amp;version=1.0.0&amp;visibility=public&amp;description=Youtube Feeds&amp;endpointType=nonsecured&amp;http_checked=http&amp;https_checked=https&amp;tags=youtube,gdata,migideas.com.au/www/573/files/pf-thumbnail-youtube_logo.jpg&amp;context=/youtube&amp;tiersCollection=Gold GET&amp;resourceMethodAuthType-0=Application&amp;resourceMethodThrottlingTier-0=Unlimited&amp;uri-d'endpoint_config={" production_endpoints":="" standardfeeds"}}<="" {"url":=""></a>
---------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### Get All APIs

Description	Lists all the created APIs.
URI	<a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a>
URI Parameters	?action=getAllAPIs
HTTP Methods	GET
Example	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag?action=getAll APIs">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag?action=getAll APIs</a>

#### Get an API

Description	Get details of a specific API.
URI	<a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a>
URI Parameters	action=getAPI&name=xxx&version=xxx&provider=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies <a href='http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag -d "action=getAPI&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin'>http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag -d "action=getAPI&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin"</a>

#### Remove an API

Description	Remove an API.
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag</a>
URI Parameters	action=removeAPI&name=xxx&version=xxx&provider=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies <a href='http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag -d "action=removeAPI&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin'>http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag -d "action=removeAPI&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin"</a>

#### Copy an API

Description	Copy an API to a newer version.
URI	<a href="http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag">http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag</a>
URI Parameters	action=createNewAPI&provider=xxx&apiName=xxx&version=xxx&newVersion=xxx

HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/publisher/site/blocks?action=createNewAPI&provider=admin&apiName=PhoneVerification&version=1.0.0&newVersion=2.

**Check Older Version**

Description	Does older version of API exist.
URI	<a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a>
URI Parameters	action=isAPIOlderVersionExist&provider=xxx&name=xxx&version=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a> -d "action=isAPIOlderVersionExist&provider=admin&name=PhoneVerification&version=1.0.0"

**Change API Status**

Description	Change the API's status.
URI	<a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a>
URI Parameters	action=updateStatus&name=xxx&version=1.0.0&provider=apiCreateName&status=PUBLISHED&put
HTTP Methods	POST
Example	curl -X POST -b cookies ' <a href="http://localhost:9763/publisher/s">http://localhost:9763/publisher/s</a> 'action=updateStatus&name=PhoneVerification&version=1.0.0&provider=admin&status=PUBLISHED

**Add/Update an API Document**

Description	Add a new API document.
URI	<a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a>
URI Parameters	<b>A d d</b> action=addDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&docType=xxx  Note that docVisibility is applicable only if you have enabled it. See <a href="#">API documentation visibility</a> .  <b>U p d a t e</b> action=addDocumentation&mode=Update&provider=xxx&apiName=xxx&version=xxx&docName=xxx
HTTP Methods	POST
Example	<b>Add Document:</b> curl -X POST -b cookies "action=addDocumentation&provider=admin&apiName=PhoneVerification&version=1.0.0&docName=testDoc&docType=how to&sourceType=inline&docUrl=&sum"  <b>Update Document:</b> curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/documentation/verificatio">http://localhost:9763/publisher/site/blocks/documentation/verificatio</a> n&version=1.0.0&docName=testDoc&docType=how to&sourceType=inline&docUrl=&sum

**Remove an API Document**

Description	Remove an API document.
URI	<a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a>
URI Parameters	action=removeDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&docType=
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag?action=removeDocumentation&amp;provider=admin&amp;apiName=PhoneVerification&amp;version=1.0.0&amp;docName=PhoneVerification&amp;docType=application%2fjson&amp;To='">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag?action=removeDocumentation&amp;provider=admin&amp;apiName=PhoneVerification&amp;version=1.0.0&amp;docName=PhoneVerification&amp;docType=application%2fjson&amp;To='</a>

### Get all Throttling Tiers

Description	Get the throttling tiers that can be applied to APIs
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?</a>
URI Parameters	action=getTiers
HTTP Methods	GET
Example	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=getTiers">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=getTiers</a>

### Check if API Exists

Description	Check if an API by a given name exists in the API Publisher
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
URI Parameters	action=isAPINameExist&apiName=<name of the API>
HTTP Methods	GET
Example	curl -b cookies " <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINameExist&amp;apiName=PhoneVerification">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINameExist&amp;apiName=PhoneVerification</a> "

### Validate Roles

Description	Check if the user logged in user is any one in a given list of users
URI	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
URI Parameters	action=validateRoles&roles=<list of roles>
HTTP Methods	GET
Example	curl -b cookies " <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&amp;roles=admin">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&amp;roles=admin</a> "

## Store APIs

Store APIs provide the following REST resources.

[ Login ] [ Logout ] [ User Signup ] [ Search APIs ] [ Get all Paginated Published APIs ] [ Get Published APIs by Application ] [ Add an Application ] [ Update an Application ] [ Get Applications ] [ Remove an Application ] [ Generate an Application Key ] [ Add a Subscription ] [ List Subscriptions ] [ List Subscriptions by Application ] [ Remove a Subscription ] [ Delete an OAuth Application ] [ Clean Partially Created Keys ] [ Get all Documentation ] [ Get the Contents of a File Document ] [ Add an API Comment ] [ Get all Endpoint URLs ]

 **Note:** When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

The response is a JSON message.

## Login

Description	Log in to API Store.
URI	<a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag</a>
URI Parameters	action=login&username=xxx&password=xxx
HTTP Methods	POST
Example	curl -X POST -c cookies <a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag</a> -d 'action=login&username=admin&password=admin'

## Logout

Description	Log out from API Store.
URI	<a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout</a>
URI Parameters	?action=logout
HTTP Methods	GET
Example	curl -b cookies <a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout</a>

## User Signup

Description	Add a new API Consumer.
URI	<a href="http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag">http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag</a>
URI Parameters	action=addUser&username=xxx&password=xxx&allFieldsValues=firstname lastname email
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag">http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag</a> -d 'action=addUser&username=user1&password=test123&allFieldsValues=firstname lastname email'

## Search APIs

Description	Search for APIs using a given query.
-------------	--------------------------------------

URI	<a href="http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag">http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag</a>
URI Parameters	<p>action=searchAPIs&amp;query=&lt;query&gt;&amp;start=&lt;number&gt;&amp;end=&lt;number&gt;</p> <p>The <b>start</b> and <b>end</b> parameters determine the range of APIs you want to retrieve. For example, if start=1 and end=3, the first 3 APIs that appear in the search results will be returned. <b>Note</b> that both 0 and 1 represent the first API in the search results, so start=0 and start=1 both means the same.</p>
HTTP Methods	POST
Example	curl -X POST -b cookies " <a href="http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag">http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag</a> " -d "action=searchAPIs&query=test&start=0&end=3"

#### Get all Paginated Published APIs

Description	Get a list of all published APIs in paginated form so that browsing is easier.
URI	<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag</a>
URI Parameters	<p>action=<i>getAllPaginatedPublishedAPIs</i>, tenant, start, end, returnAPITags (optional)</p> <p>The <b>start</b> and <b>end</b> parameters determine the range of APIs you want to retrieve. For example, if start=1 and end=10, the first 10 APIs that appear in the API Store will be returned. <b>Note</b> that both 0 and 1 represent the first API in the store, so start=0 and start=1 both specify that you want to start with the first API.</p> <p>The <b>returnAPITags</b> parameter is optional. If <code>returnAPITags=true</code>, the system makes a call to the registry and returns the tags of each API in the response.</p>
HTTP Methods	GET
Example	<p>To get the first 100 APIs in the API Store:</p> <pre>curl -b cookies "<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllPaginatedPublishedAPIs&amp;tenant=carbon.super&amp;start=1&amp;end=100">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllPaginatedPublishedAPIs&amp;tenant=carbon.super&amp;start=1&amp;end=100</a>"</pre>

 Please note that the `getAllPublishedAPIs` API is now deprecated. You can get the same functionality from `getAllPaginatedPublishedAPIs`.

#### Get Published APIs by Application

Description	Get a list of published APIs filtered by the subscribed Application. Login API needs be called prior to this API.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	action= <i>getSubscriptionByApplication</i> &app=App1
HTTP Methods	GET
Example	curl -b cookies ' <a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?app=App1&amp;defaultApplication">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?app=App1&amp;defaultApplication</a> '

#### Add an Application

Description	Add a new application.
-------------	------------------------

URI	<a href="http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag">http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag</a>
URI Parameters	action=addApplication&application=xxx&tier=xxx&description=xxx&callbackUrl
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag">http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag</a> -d 'action=addApplication&application=NewApp1&tier=Unlimited&description=&callbackUrl='

#### Update an Application

Description	Update an existing application.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-update/ajax/application-update.jag">http://localhost:9763/store/site/blocks/application/application-update/ajax/application-update.jag</a>
URI Parameters	action=updateApplication&applicationOld=xxx&applicationNew=xxx&callbackUrlNew=xxx&descriptior
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/application/application-update.jag">http://localhost:9763/store/site/blocks/application/application-update.jag</a> -d 'action=updateApplication&applicationOld=NewApp1&applicationNew=NewApp2&tier=Unlimited&des

#### Get Applications

Description	Get list of applications.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag">http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</a>
URI Parameters	?action=getApplications
HTTP Methods	GET
Example	curl -b cookies <a href="http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag">http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</a> ?action=getApplications

#### Remove an Application

Description	Remove an existing application.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag">http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag</a>
URI Parameters	action=removeApplication&application=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag">http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag</a> -d "action=removeApplication&application=NewApp2"

#### Generate an Application Key

Description	Generate the key and secret values for a new application.
-------------	-----------------------------------------------------------

URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	<p>action=generateApplicationKey&amp;application=&lt;app_name&gt;&amp;keytype=&lt;PRODUCTION SANDBOX&gt;&amp;callbackUrl=&lt;URL&gt;&amp;authorizedDomains=&lt;The domains from which requests are allowed to the APIs&gt;&amp;validityTime=&lt;time duration in seconds&gt;&amp;tokenScope</p> <p><b>tokenScope</b> is given in the request when your API has Auth scopes defined. See <a href="#">OAuth scopes</a>.</p>
HTTP Methods	POST
Examples	<ol style="list-style-type: none"> <li>curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> -d 'action=generateApplicationKey&amp;application=NewApp1&amp;keytype=PRODUCTION&amp;callbackUrl=&amp;authorizedDomains=ALL&amp;validityTime=360000'</li> <li>curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> -d 'action=generateApplicationKey&amp;application=NewApp1&amp;keytype=SANDBOX&amp;callbackUrl=&amp;authorizedDomains=ALL&amp;validityTime=360000&amp;<b>tokenScope=scope1</b>'</li> </ol>

#### Add a Subscription

Description	Add a new API subscription.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	<ul style="list-style-type: none"> <li><b>By application name:</b> action=addAPISubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;tier=xxx&amp;<b>applicationName=xxx</b></li> <li><b>By application ID:</b> action=addSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;tier=xxx&amp;<b>applicationId=xxx</b></li> </ul>
HTTP Methods	POST
Example	<ul style="list-style-type: none"> <li><b>By application name:</b> curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> -d 'action=addAPISubscription&amp;name=TestAPI&amp;version=1.0.0&amp;provider=admin&amp;tier=Gold&amp;<b>applicationName=DefaultApplication</b>'</li> <li><b>By application ID:</b> curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> -d 'action=addSubscription&amp;name=TestAPI&amp;version=1.0.0&amp;provider=admin&amp;tier=Gold&amp;<b>applicationId=1</b>'</li> </ul>

#### List Subscriptions

Description	List all applications with active subscriptions, along with the access key information of each application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	<p>action=getAllSubscriptions, selectedApp (optional)</p> <p>You can give an application's name in the <b>SelectedApp</b> parameter. The API then returns the given application's subscribed APIs and access key information. If you do not specify this parameter, only the first application in the retrieved application list will contain subscribed API details, in addition to the access key information.</p>
HTTP Methods	GET

Examples	<ol style="list-style-type: none"> <li>1. curl -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions</a></li> <li>2. curl -b cookies '<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions&amp;selectedApp&gt;NewApp1">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions&amp;selectedApp&gt;NewApp1</a>'</li> </ol>
----------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### List Subscriptions by Application

Description	List all API subscriptions of a given application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	action=getSubscriptionByApplication&app=<application_name>
HTTP Methods	GET
Example	curl -b cookies ' <a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getSubscriptionByApplication&amp;app=DefaultApplication">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getSubscriptionByApplication&amp;app=DefaultApplication</a> '

#### Remove a Subscription

Description	Remove an API subscription.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag">http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag</a>
URI Parameters	<ul style="list-style-type: none"> <li>• <b>By application name:</b> action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;<b>applicationName</b>=xxx</li> <li>• <b>By application Id:</b> action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;<b>applicationId</b>=xxx</li> </ul>
HTTP Methods	POST
Example	<ul style="list-style-type: none"> <li>• <b>By application Name:</b> curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag">http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag</a> -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin&amp;<b>applicationName</b>=DefaultApplication'</li> <li>• <b>By application Id:</b> curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag">http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag</a> -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin&amp;<b>applicationId</b>=1'</li> </ul>

#### Delete an OAuth Application

Description	Deletes an OAuth application in a third-party Authorization Server. If you delete it through the API Store UI, only the mapping that is maintained in the API Manager side will be deleted.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action= <b>deleteAuthApplication</b> &consumerKey=<application_key>
HTTP Methods	POST
Example	curl -k -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> -d 'action=deleteAuthApplication&consumerKey=4IHddsxCtpFa2zJE1EbBpJy_NIQa'

### Clean Partially Created Keys

Description	Cleans any partially created keys from the API Manager database, before adding a new subscription. Partially created keys can remain in the API Manager databases when an OAuth application of a <a href="#">third party authorization server</a> gets deleted using the API Store UI. It only deletes the mapping that is maintained in the API Manager side.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action=cleanUpApplicationRegistration&applicationName=xxx&keyType=PRODUCTION/SANDBC
HTTP Methods	POST
Example	curl -X POST -b cookies <a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a> ?action=cleanUpApplicationRegistration&applicationName=DefaultApplication&keyType=PRODUCTIC

### Get all Documentation

Description	Get all documents create for a given API
URI	<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag</a>
URI Parameters	action=getAllDocumentationOfApi&name=<API Name>&version=x.x.x&provider=<Name of the API provider>
HTTP Methods	GET
Example	curl -b cookies "http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllDocumentation&version=1.0.0&provider=admin"

### Get the Contents of a File Document

Description	Get the contents of a file that is attached to API documentation of type 'File'
URI	<a href="http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag">http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag</a>
URI Parameters	action=getFileDocumentByFilePath&filePath=<Get the file path using getAllDocumentationOfApi>
HTTP Methods	GET
Example	curl "http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag?action=getFileDocur"

### Add an API Comment

Description	Add a comment to an API.
URI	<a href="http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag">http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag</a>
URI Parameters	action=addComment&name=xxx&version=xxx&provider=xxx&comment=xxx
HTTP Methods	POST

Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag -d 'action=addComment&amp;name=WeatherAPI&amp;version=1.0.0&amp;provider=admin&amp;comment=testcomment'</pre>
---------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Get all Endpoint URLs

Description	Get all the endpoint URLs of the API Gateway environments configured for an API.
URI	<a href="http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag">http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag</a>
URI Parameters	action=getAPIEndpointURLs&name=xxx&version=x.x.x&provider=xxx
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag -d 'action=getAPIEndpointURLs&amp;name=WeatherAPI&amp;version=1.0.0&amp;provider=admin'</pre>

## Token API

Users need access tokens to invoke APIs subscribed under an application. Access tokens are passed in the HTTP header when invoking APIs. The API Manager provides a Token API that you can use to generate and renew user and application access tokens. The response of the Token API is a JSON message. You extract the token from the JSON and pass it with an HTTP Authorization header to access the API.

The following topic explain how to generate/renew access tokens and authorize them. WSO2 API Manager supports the four most common authorization grant types and you can also define additional types.

- Exchanging SAML2 Bearer Tokens with OAuth2 (SAML Extension Grant Type)
- Generating Access Tokens with Authorization Code (Authorization Code Grant Type)
- Generating Access Tokens with NT LAN Manager (NTLM Grant Type)
- Generating Access Tokens with User Credentials (Password Grant Type)

Also see the following:

- Renewing access tokens
- Revoking access tokens
- Configuring the token expiration time

### Renewing access tokens

After an access token is generated, sometimes you might have to renew the old token due to expiration or security concerns. You can renew an access token using a refresh token, by issuing a REST call to the Token API with the following parameters.

- The Token API URL is <https://localhost:8243/token>, assuming that both the client and the Gateway are run on the same server.
- payload: "grant\_type=refresh\_token&refresh\_token=<retoken>&scope=PRODUCTION". Replace the <retoken> value with the refresh token generated in the [previous section](#).
- headers: Authorization :Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded. Replace <base64 encoded string> as appropriate.

For example, the following cURL command can be used to access the Token API.

```
curl -k -d "grant_type=refresh_token&refresh_token=<retoken>&scope=PRODUCTION" -H
"Authorization: Basic
SVpzSWk2SERiQjV1OFZLZFpBblVpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

The above REST message grants you a renewed access token along with a refresh token, which you can use the next time you renew the access token. A refresh token can be used only once. At the moment, a refresh token never expires, but we will provide a way to configure an expiration time in a future release.

#### Revoking access tokens

After issuing an access token, a user or an admin can revoke it in case of theft or a security violation. You can do this by calling Revoke API using a utility like cURL. The Revoke API's endpoint URL is <http://localhost:8280/revoke>.

Parameters required to invoke this API are as follows:

- The token to be revoked
- Consumer key and consumer secret key. Must be encoded using Base64 algorithm

For example, curl -k -d "token=<ACCESS\_TOKEN\_TO\_BE\_REVOKED>" -H "Authorization: Basic Base64Encoded(Consumer key:consumer secret)" <http://localhost:8280/revoke>.

 **Tip:** When the API Gateway cache is enabled (it is enabled by default), even after revoking a token, it might still be available in the cache to consumers until the cache expires in approximately 15 minutes.

#### Configuring the token expiration time

User access tokens have a fixed expiration time, which is set to 60 minutes by default. Before deploying the API Manager to users, extend the default expiration time by editing the `<AccessTokenDefaultValidityPeriod>` element in `<PRODUCT_HOME>/repository/conf/identity.xml`.

Also take the **time stamp skew** into account when configuring the expiration time. The time stamp skew is used to manage small time gaps in the system clocks of different servers. For example, let's say you have two Key Managers and you generate a token from the first one and authenticate with the other. If the second server's clock runs 300 seconds ahead, you can configure a 300s time stamp skew in the first server. When the first Key Manager generates a token (e.g., with the default life span, which is 3600 seconds), the time stamp skew is deducted from the token's life span. The new life span is 3300 seconds and the first server calls the second server after 3200 seconds.

You configure the time stamp skew using the `<TimestampSkew>` element in `<PRODUCT_HOME>/repository/conf/identity.xml`.

 **Tip:** Ideally, the time stamp skew should not be larger than the token's life span. We recommend you to set it to zero if the nodes in your cluster are synchronized. Also, note that when the API Gateway cache is enabled (it is enabled by default), even after a token expires, it will still be available in the cache for consumers until the cache expires in approximately 15 minutes.

When a user access token expires, the user can try regenerating the token as explained in the [Renew user tokens section](#).

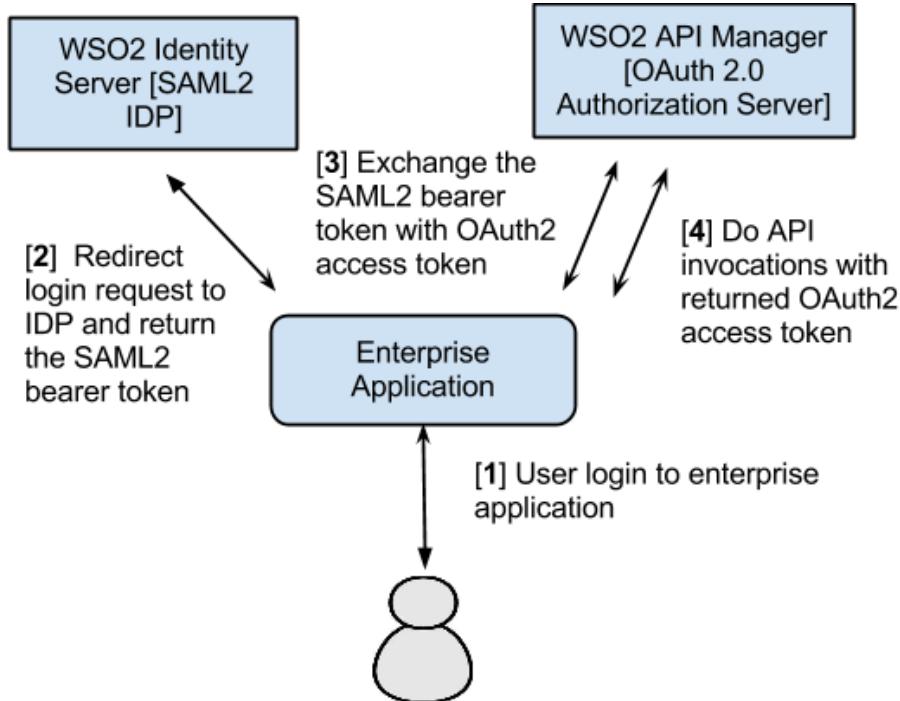
## Exchanging SAML2 Bearer Tokens with OAuth2 (SAML Extension Grant Type)

SAML 2.0 is an XML-based protocol. It uses security tokens containing assertions to pass information about an end-user between a SAML authority and a SAML consumer. A SAML authority is an identity provider (IDP) and a SAML consumer is a service provider (SP).

Enterprise applications that have SAML2 based SSO infrastructures sometimes need to consume OAuth-protected

resources through APIs. However, these apps prefer to use the existing trust relationship with the IDP, even if the OAuth authorization server is entirely different from the IDP. The API Manager leverages this trust relationship by exchanging the SAML2.0 token to an OAuth token with the authorization server. It acts as the OAuth authorization server.

The diagram below depicts the above with **WSO2 Identity Server** (version 4.5.0 onwards) as the IDP.



The steps of the above diagram are explained below:

**Step [1]:** User initiates a login call to an enterprise application

**Step [2]:**

- As the application is a SAML SP, it redirects the user to the SAML2.0 IDP to log in.
- The user provides credentials at the IDP and is redirected back to the SP with a SAML2.0 token signed by the IDP.
- The SP verifies the token and logs the user to the application.
- The SAML 2.0 token is stored in the user's session by the SP.

**Step [3]:**

- The enterprise application (SP) wants to access an OAuth2 protected API resource through WSO2 API Manager.
- The application makes a request to the API Manager to exchange the SAML2 bearer token for an OAuth2.0 access token.
- The API Manager validates the assertion and returns the access token.

**Step [4]:** User does API invocations through the API Manager by setting it as an Authorization header with the returned OAuth2 access token.

Let's configure the token exchange.

#### Configuring the token exchange



Before you begin, make sure you have the following:

- A valid user account in the API Store.
- A valid consumer key and consumer secret. Initially, these keys must be generated through the

management console by clicking the **Generate** link on **My Subscriptions** page.

- A running API Gateway instance.
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the <APIKeyManager><ServerURL> element of the <AM\_HOME>/repository/conf/api-manager.xml file.
- A valid SAML2 assertion. You can do this using the Identity Server as the Identity Server can act as a SAML2 SSO IDP. See [Configuring SAML2 Single-Sign-On Across Different WSO2 Products](#) in the Identity Server documentation for more information.

We use **WSO2 Identity Server 5.0.0** as the IDP to get a SAML token and the API Manager as the OAuth server.

1. Log in to the API Manager's management console (<https://localhost:9443/carbon>) using admin/admin credentials and select **Add** under **Identity Providers** menu in the **Main** menu.



2. Provide the following values to configure the IDP:

- Under **Basic Information**
  - **Identity Provider Name:** Enter a unique name for IDP
  - **Identity Provider Public Certificate:** The certificate used to sign the SAML assertion. Export the public certificate of WSO2 IS and import it here.

Alternatively, you can create a self-signed certificate and then export it as a .cer file using the following commands:

```
keytool -genkey -alias wookie -keyalg RSA -keystore wookieKeystore.jks
-keysize 4096
keytool -v -export -file keystore1.cer -keystore keystore1.jks -alias
keystore1
```

- **Alias:** Give the name of the alias if the Identity Provider identifies this token endpoint by an alias. E.g., <https://localhost:9443/oauth2/token>
- Under **Federated Authenticators -> SAML2 Web SSO Configuration**
  - **Enable SAML2 Web SSO:** true
  - **Identity Provider Entity Id:** The SAML2 issuer name specified when generating the assertion token, which contains the unique identifier of the IDP. **You give this name when configuring the SP.**
  - **Service Provider Entity Id:** Issuer name given when configuring the SP
  - **SSO URL:** Enter the IDP's SAML2 Web SSO URL value. E.g., <https://localhost:9444/samlss/> if you have offset the default port, which is 9443.

Add Identity Provider

Basic Information

Identity Provider Name: <sup>*</sup>	<input type="text" value="IS"/> <small>(i) Enter a unique name for this identity provider</small>												
Display Name:	<input type="text"/> <small>(i) Specify the identity provider's display name</small>												
Description:	<input type="text"/> <small>(i) A meaningful description about the identity provider</small>												
Federation Hub Identity Provider:	<input type="checkbox"/> <small>(i) Check if this points to a federation hub identity provider</small>												
Home Realm Identifier:	<input type="text"/> <small>(i) Enter the home realm identifier for this identity provider</small>												
Identity Provider Public Certificate:	<input type="button" value="Browse..."/> <small>No file selected.</small> <small>(i) Upload identity provider's public certificate in PEM format</small> <input type="button" value="Delete Identity Provider Public Certificate"/>												
<table border="1"> <thead> <tr> <th>Issuer DN</th> <th>Subject DN</th> <th>Not After</th> <th>Not Before</th> <th>Serial Number</th> <th>Version</th> </tr> </thead> <tbody> <tr> <td>CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US</td> <td>CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US</td> <td>13/02/2035</td> <td>19/02/2010</td> <td>1266562946</td> <td>3</td> </tr> </tbody> </table>		Issuer DN	Subject DN	Not After	Not Before	Serial Number	Version	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	13/02/2035	19/02/2010	1266562946	3
Issuer DN	Subject DN	Not After	Not Before	Serial Number	Version								
CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	13/02/2035	19/02/2010	1266562946	3								
Alias:	<input type="text" value="https://localhost:9443/oauth2/token"/> <small>(i) If the resident identity provider is known by an alias at the federated identity provider specify it</small>												

Claim Configuration  
 Role Configuration  
 Federated Authenticators  
 OpenID Configuration  
 SAML2 Web SSO Configuration ✓  
 OAuth2/OpenID Connect Configuration  
 WS-Federation (Passive) Configuration  
 Facebook Configuration  
 Just-in-Time Provisioning  
 Outbound Provisioning Connectors

Next, let's register a service provider.

3. Log in to the management console of the Identity Server and select **Add** under **Service Providers** menu in the main menu.



4. Choose to edit the service provider that you just registered and select **SAML2 Web SSO Configuration**.

## Service Providers

Basic Information

Service Provider Name:<sup>\*</sup>  ⑦ A unique name for the service provider

Description:  ⑦ A meaningful description about the service provider

(▼) Claim Configuration

(▼) Role/Permission Configuration

(▲) Inbound Authentication Configuration

(▼) SAML2 Web SSO Configuration

(▼) OAuth/OpenID Connect Configuration

(▼) WS-Trust Security Token Service Configuration

(▼) WS-Federation (Passive) Configuration

(▼) OpenID Configuration

5. Provide the following values to configure the SP:

- **Issuer:** Give any name
- **Assertion Consumer URL:** The URL to which the IDP sends the SAML response. E.g., [https://localhost:9443/store/jagg/jaggery\\_acs.jag](https://localhost:9443/store/jagg/jaggery_acs.jag).
- **Enable Response Signing:** true
- **Enable Assertion Signing:** true
- **Enable Audience Restriction:** true
- **Audience:** URL of the token API. E.g., <https://localhost:9443/oauth2/token>.

## Register New Service Provider

**Edit Service Provider(TestSP)**

Issuer *	<input type="text" value="TestSP"/>
Assertion Consumer URL *	<input type="text" value="https://localhost:9443/store/jagg/jaggi"/>
NameID format	<input type="text" value="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"/>
<input type="checkbox"/> Use fully qualified username in the NameID <input checked="" type="checkbox"/> Enable Response Signing <input checked="" type="checkbox"/> Enable Assertion Signing  <input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests <input type="checkbox"/> Enable Assertion Encryption Certificate Alias <input type="text" value="wso2carbon.cert"/>	
<input type="checkbox"/> Enable Single Logout <i>Custom Logout URL</i> <input type="text"/> <input checked="" type="checkbox"/> Enable Attribute Profile <input type="checkbox"/> <i>Include Attributes in the Response Always</i>	
<input checked="" type="checkbox"/> Enable Audience Restriction Audience <input type="text" value="https://localhost:9443/oauth2/token"/> <input type="button" value="Add Audience"/>	
<input type="checkbox"/> Enable Recipient Validation <i>Recipient</i> <input type="text"/> <input type="button" value="Add Recipient"/>	
<input type="checkbox"/> Enable IdP Initiated SSO	

Let's see how to get a signed SAML2 token (encoded assertion value) when authenticating against a SAML2 IDP. With the authentication request, you pass attributes such as the SAML2 issuer name, token endpoint and the restricted audience. In this guide, we use a command-line client program developed by WSO2 to create the 64-bit, URL-encoded SAML assertion.

6. Download the client program from [here](#) and unzip the SAML2AssertionCreator.zip file.
7. Execute the following command inside that SAML2AssertionCreator directory. It generates a SAML token.

```
java -jar SAML2AssertionCreator.jar <Identity_Provider_Entity_Id> <user_name>
<recipient> <requested_audience> <Identity_Provider_JKS_file>
<Identity_Provider_JKS_password> <Identity_Provider_certificate_alias>
<private_key_password>
```

Here's an example command with the issuer name as TestSP:

```
java -jar SAML2AssertionCreator.jar TestSP admin
https://localhost:9443/oauth2/token
https://localhost:9443/oauth2/token/home/dinusha/nothing/WSO2/API-Manager/saml-oa
uth/wso2is-5.0.0/rhbepository/resources/security/wso2carbon.jks wso2carbon
wso2carbon wso2carbon
```

You now have a SAML2 assertion.

8. Execute the following command to get the OAuth Access token. You can generate a consumer key and consumer secret pair by clicking the **Generate** link on **My Subscriptions** page of the API Publisher.

```
curl -k -d
"grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=<Assertion_pr
ovided_by_client>&scope=PRODUCTION" -H "Authorization: Basic <Base64 encoded
consumer key:consumer secret>, Content-Type: application/x-www-form-urlencoded"
https://<IP of the APIM server>:9443/oauth2/token
```

#### ***Invoking the Token API to generate tokens***

Follow the steps below to invoke the token API to generate access tokens from SAML2 assertions.

1. Combine the consumer key and consumer secret keys as `consumer-key:consumer-secret`. Encode the combined string using base64 (<http://base64encode.org>). Here's an example consumer key and secret combination: `wU62DjlyDBnq87G1Bwp1fqvmAbAa:ksdSdoefDDP7wpaElfqvmjDue`.

Let's create a SAML2 assertion using the same command-line client that you used in the previous section.

2. Download the command-line tool from [here](#) and extract the ZIP file.
3. Go to the extracted folder using the command line and execute the following command. We assume that both the client and the API Gateway run on the same server. Therefore, the Token API URL is <https://localhost:8243/token>.

```
java -jar SAML2AssertionCreator.jar <Identity_Provider_Entity_Id> admin
https://localhost:9443/oauth2/token https://localhost:9443/oauth2/token
<Identity_Provider_JKS_file> <Identity_Provider_JKS_password>
<Identity_Provider_certificate_alias>
```

The arguments are as follows:

- The saml:Issuer (a unique identifier of the identity provider) value
- The saml:Subject -> saml:NameId value
- The value of saml:Subject -> saml:SubjectConfirmation -> saml:SubjectConfirmationData.Recipient
- The fourth argument can take multiple values separated by commas. They are added to the saml:AudienceRestriction element of the token. Each value is added as a saml:Audience element within saml:AudienceRestriction.
- Pointer to the Java Key Store (JKS) file to be used for credentials
- The JKS password
- The alias of the public certificate
- The password of the private key that is used for signing

This command returns a SAML2 assertion XML string and a base64-URL encoded assertion XML string.

4. Access the Token API using a REST client such as curl. For example, the following Curl command generates an access token and a refresh token. You can use the refresh token at the time a token is renewed.

```
curl -k -d
"grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=<base64-URL_encoded_assertion>&scope=PRODUCTION" -H "Authorization: Basic <base64_encoded_consumer-key:consumer-secret>, Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

## Generating Access Tokens with Authorization Code (Authorization Code Grant Type)

Instead of requesting authorization directly from the resource owner (resource owner's credentials), in this grant type, the client directs the resource owner to an authorization server. The authorization server works as an intermediary between the client and resource owner to issues an authorization code, authenticate the resource owner and obtain authorization. As this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a Web browser) and receiving incoming requests (via redirection) from the authorization server.

The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint (you can use the `/authorize` endpoint for the authorization code grant type of OAuth 2.0). It includes the client identifier, `response_type`, requested scope, and a redirection URI to which the authorization server sends the user-agent back after granting access. The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner granted or denied the client's access request. Assuming the resource owner grants access, the authorization server then redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code.

The client then requests an access token from the authorization server's `/token` endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. It then includes the redirection URI used to obtain the authorization code for verification. The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI matches the URI used to redirect the client from the `/authorize` endpoint in the previous response. If valid, the authorization server responds back with an access token and, optionally, a refresh token.

### Invoking the Token API to generate tokens

Assuming that both the client and the API Gateway are run on the same server, the Authorization API URL is <https://localhost:8243/authorize>.

- query component: `response_type=code&client_id=<consumer_key>&scope=PRODUCTION&redirect_uri=<application_callback_url>`
- headers: `Content-Type: application/x-www-form-urlencoded`

For example, the client directs the user-agent to make the following HTTP request using TLS.

```
GET
/authorize?response_type=code&client_id=wU62DjlyDBnq87G1Bwp1fqvmAbAa&scope=PRODUCTION&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcbs
HTTP/1.1
Host: server.example.com
Content-Type:
application/x-www-form-urlencoded
```

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location:
https://client.example.com/cb?code=Spxl0BeZQQYbYS6WxSbIA
```

Now the client makes the following HTTP request using TLS to the /token endpoint.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic
SVpzSWk2SERiQjV1OFZLZFpBblVpX2ZaM2Y4YTpHbTBisjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzh
Content-Type:
application/x-www-form-urlencoded
grant_type=authorization_code&code=Spxl0BeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2F
client%2Eexample%2Ecom%2Fcbs
```

The /token endpoint responds in the same way like in password grant type.

Note that if you are using a separate server for authentication (e.g., a distributed API Manager setup or an instance of WSO2 Identity Server as the authentication server), be sure to give the full URL of the authentication server in <APIM\_HOME>/repository/conf/security/application-authenticators.xml file. The default configuration has a relative path, which works in a standalone API Manager setup:

```
<Authenticators>
    <Authenticator name="BasicAuthenticator" disabled="false" factor="1">
        <Status value="10" loginPage="/authenticationendpoint/login.do" />
    </Authenticator>
</Authenticators>
```

## Generating Access Tokens with NT LAN Manager (NTLM Grant Type)

**NTLM** is the successor of the authentication protocol in Microsoft LAN Manager (LANMAN), an older Microsoft product, and attempts to provide backwards compatibility with LANMAN. You can obtain an access token to your API in an API Manager instance running on **Windows** by providing a valid NTLM token as an authorization grant. The steps are given below:

### Invoking the Token API to generate tokens

1. Get a valid consumer key and consumer secret pair. Initially, you generate these keys through the API Store by clicking the **Generate** button on the **My Subscriptions** page.
2. Combine the consumer key and consumer secret keys in the format `consumer-key:consumer-secret` and encode the combined string using base64 (<http://base64encode.org>). In order to generate an access token with NTLM, you must have an NTLM token.
3. Generate an NTLM token by running the sample provided in the <APIM\_HOME>/samples/NTLMGrantClient directory. See the `Readme.txt` in the same folder for instructions.
4. Invoke the token API in the following manner to get an access token. The value of the `windows_token` in the following command is the NTLM token that you generated in the previous step.

```
curl -k -d "grant_type=iwa:ntlm&windows_token=<give the NTLM token you got in step 3>" -H "Authorization: Basic <give the string you got in step2>, Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

## Generating Access Tokens with User Credentials (Password Grant Type)

You can obtain an access token by providing the resource owner's username and password as an authorization grant. It requires the base64 encoded string of the `consumer-key:consumer-secret` combination. You need to meet the following prerequisites before using the Token API to generate a token.

### Prerequisites

- A valid user account in the API Store. You can self sign up if it is [enabled by an admin](#).
- A valid consumer key and consumer secret pair. Initially, these keys must be generated through the API Store by clicking the **Generate** link on **My Subscriptions** page.
- A running API Gateway instance (typically an API Manager instance should be running). For instructions on API Gateway, see [Components](#).
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the `<APIKeyManager><ServerURL>` element of the `<AM_HOME>/repository/conf/api-manager.xml` file.
- If you have multiple Carbon servers running on the same computer, [change the port with an offset](#) to avoid port conflicts.

### Invoking the Token API to generate tokens

1. Combine the consumer key and consumer secret keys in the format `consumer-key:consumer-secret` and encode the combined string using base64. Encoding to base64 can be done using the URL:<http://base64encode.org>.

Here's an example consumer key and secret combination: `wU62DjlyDBnq87G1BwplfqvmAbAa:ksdSdoefDDP7wpaElfqvmjDue`.

2. Access the Token API by using a REST client such as cURL, with the following parameters.

- Assuming that both the client and the API Gateway are run on the same server, the token API url is `https://localhost:8243/token`
- `payload` - `"grant_type=password&username=<username>&password=<password>&scope=<scope>"`. Replace the `<username>` and `<password>` values as appropriate.

 **Tip:** `<scope>` is optional.

If you define a **scope** for an API's resource, the API can only be accessed through a token that is issued for the scope of the said resource. For example, if you define a scope named 'update' and issue one token for the scopes 'read' and 'update', the token is allowed to access the resource. However, if you issue the token for the scope named 'read', the request to the API will be blocked.

- `headers` - `Authorization: Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded`. Replace the `<base64 encoded string>` as appropriate.

For example, use the following cURL command to access the Token API. It generates two tokens as an access token and a refresh token. You can use the refresh token at the time a token is renewed .

```
curl -k -d "grant_type=password&username=<username>&password=<password>" -H
"Authorization: Basic
SVpzSWk2SERiQjV1OFZLZfpBblVpX2ZaM2Y4YTpHbTBisjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

Instead of using the Token API, you can generate access tokens from the API Store's UI.

## WSO2 Admin Services

WSO2 products are managed internally using SOAP Web services known as **admin services**. WSO2 products come with a management console UI, which communicates with these admin services to facilitate administration capabilities through the UI.

A service in WSO2 products is defined by the following components:

- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

There can be instances where you want to call back-end Web services directly. For example, in test automation, to minimize the overhead of having to change automation scripts whenever a UI change happens, developers prefer to call the underlying services in scripts. The topics below explain how to discover and invoke these services from your applications.

### Discovering the admin services

By default, the WSDLs of admin services are hidden from consumers. Given below is how to discover them.

1. Set the <HideAdminServiceWSDLs> element to false in the <PRODUCT\_HOME>/repository/conf/carbon.xml file.
2. Restart the server.
3. Start the WSO2 product with the -DosgiConsole option, such as sh <PRODUCT\_HOME>/bin/wso2server.sh -DosgiConsole in Linux.
4. When the server is started, hit the enter/return key several times to get the OSGI shell in the console.
5. In the OSGI shell, type: osgi> listAdminServices
6. The list of admin services of your product are listed. For example:

```
osgi> listAdminServices
Admin services deployed on this server:
1. ProvisioningAdminService, ProvisioningAdminService, https://192.168.219.1:8243/services/ProvisioningAdminService
2. SynapseApplicationAdmin, SynapseApplicationAdmin, https://192.168.219.1:8243/services/SynapseApplicationAdmin
3. CarbonAppUploader, CarbonAppUploader, https://192.168.219.1:8243/services/CarbonAppUploader
4. OperationAdmin, OperationAdmin, https://192.168.219.1:8243/services/OperationAdmin
5. SequenceAdminService, SequenceAdminService, https://192.168.219.1:8243/services/SequenceAdminService
6. MediationLibraryAdminService, MediationLibraryAdminService, https://192.168.219.1:8243/services/MediationLibraryAdminService
7. StatisticsAdmin, StatisticsAdmin, https://192.168.219.1:8243/services/StatisticsAdmin
8. LoggedUserInfoAdmin, LoggedUserInfoAdmin, https://192.168.219.1:8243/services/LoggedUserInfoAdmin
9. MediationStatisticsAdmin, MediationStatisticsAdmin, https://192.168.219.1:8243/services/MediationStatisticsAdmin
10. TopicManagerAdminService, TopicManagerAdminService, https://192.168.219.1:8243/services/TopicManagerAdminService
11. MessageProcessorAdminService, MessageProcessorAdminService, https://192.168.219.1:8243/services/MessageProcessorAdminService
12. ApplicationAdmin, ApplicationAdmin, https://192.168.219.1:8243/services/ApplicationAdmin
13. NDataSourceAdmin, NDataSourceAdmin, https://192.168.219.1:8243/services/NDataSourceAdmin
14. ServiceGroupAdmin, ServiceGroupAdmin, https://192.168.219.1:8243/services/ServiceGroupAdmin
15. ClassMediatorAdmin, ClassMediatorAdmin, https://192.168.219.1:8243/services/ClassMediatorAdmin
```

7. To see the service contract of an admin service, select the admin service's URL and then paste it in your browser with ?wsdl at the end. For example: https://localhost:9443/services/UserAdmin?wsdl

 In products like WSO2 ESB and WSO2 API Manager, the port is 8243 (assuming 0 port offset). However, you should be accessing the Admin Services via the management console port, which is 9443 when there is no port offset.

8. Note that the admin service's URL appears as follows in the list you discovered in step 6:

```
AuthenticationAdmin, AuthenticationAdmin, https://<host  
IP>:8243/services/AuthenticationAdmin
```

### Invoking an admin service

Admin services are secured using common types of security protocols such as HTTP basic authentication, WS-Security username token, and session based authentication to prevent anonymous invocations. For example, the UserAdmin Web service is secured with the HTTP basic authentication. To invoke a service, you do the following:

1. Authenticate yourself and get the session cookie.
2. Generate the client stubs to access the back-end Web services.

 To generate the stubs, you can write your own client program using the Axis2 client API or use an existing tool like [SoapUI](#) (4.5.1 or later) or wsdl2java.

The wsdl2java tool, which comes with WSO2 products by default hides all the complexity and presents you with a proxy to the back-end service. The stub generation happens during the project build process within the Maven POM files. It uses the Maven ant run plug-in to execute the wsdl2java tool.

You can also use the Java client program given [here](#) to invoke admin services. All dependency JAR files that you need to run this client are found in the /lib directory.

### Authenticate the user

The example code below authenticates the user and gets the session cookie:

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.wso2.carbon.authenticator.stub.AuthenticationAdminStub;
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.apache.axis2.context.ServiceContext;
import java.rmi.RemoteException;

public class LoginAdminServiceClient {
    private final String serviceName = "AuthenticationAdmin";
    private AuthenticationAdminStub authenticationAdminStub;
    private String endPoint;

    public LoginAdminServiceClient(String backEndUrl) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        authenticationAdminStub = new AuthenticationAdminStub(endPoint);
    }

    public String authenticate(String userName, String password) throws
RemoteException,
        LoginAuthenticationExceptionException {

        String sessionCookie = null;

        if (authenticationAdminStub.login(userName, password, "localhost")) {
            System.out.println("Login Successful");

            ServiceContext serviceContext = authenticationAdminStub.
                _getServiceClient().getLastOperationContext().getServiceContext();
            sessionCookie = (String)
serviceContext.getProperty(HTTPConstants.COOKIE_STRING);
            System.out.println(sessionCookie);
        }

        return sessionCookie;
    }

    public void logOut() throws RemoteException,
LogoutAuthenticationExceptionException {
        authenticationAdminStub.logout();
    }
}

```

 To resolve dependency issues, if any, add the following dependency JARs location to the class path: <PRODUCT\_HOME>/repository/components/plugins.

 The the AuthenticationAdminStub class requires `org.apache.axis2.context.ConfigurationContext` as a parameter. You can give a null value there.

### Generate the client stubs

After authenticating the user, give the retrieved admin cookie with the service endpoint URL as shown in the sample below. The service management service name is ServiceAdmin. You can find its URL (e.g., <https://localhost:9443/services/ServiceAdmin>) in the `service.xml` file in the `META-INF` folder in the respective bundle that you find in <PRODUCT\_HOME>/repository/components/plugins.

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.wso2.carbon.service.mgt.stub.ServiceAdminStub;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;
import java.rmi.RemoteException;

public class ServiceAdminClient {
    private final String serviceName = "ServiceAdmin";
    private ServiceAdminStub serviceAdminStub;
    private String endPoint;

    public ServiceAdminClient(String backEndUrl, String sessionCookie) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        serviceAdminStub = new ServiceAdminStub(endPoint);
        //Authenticate Your stub from sessionCookie
        ServiceClient serviceClient;
        Options option;

        serviceClient = serviceAdminStub._getServiceClient();
        option = serviceClient.getOptions();
        option.setManageSession(true);
        option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_STRING,
sessionCookie);
    }

    public void deleteService(String[] serviceGroup) throws RemoteException {
        serviceAdminStub.deleteServiceGroups(serviceGroup);
    }

    public ServiceMetaDataWrapper listServices() throws RemoteException {
        return serviceAdminStub.listServices("ALL", "*", 0);
    }
}

```

The following sample code lists the back-end Web services:

```
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaData;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;

import java.rmi.RemoteException;

public class ListServices {
    public static void main(String[] args)
        throws RemoteException, LoginAuthenticationExceptionException,
               LogoutAuthenticationExceptionException {
        System.setProperty("javax.net.ssl.trustStore",
                           "$ESB_HOME/repository/resources/security/wso2carbon.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "wso2carbon");
        System.setProperty("javax.net.ssl.trustStoreType", "JKS");
        String backEndUrl = "https://localhost:9443";

        LoginAdminServiceClient login = new LoginAdminServiceClient(backEndUrl);
        String session = login.authenticate("admin", "admin");
        ServiceAdminClient serviceAdminClient = new ServiceAdminClient(backEndUrl,
                session);
        ServiceMetaDataWrapper serviceList = serviceAdminClient.listServices();
        System.out.println("Service Names:");
        for (ServiceMetaData serviceData : serviceList.getServices()) {
            System.out.println(serviceData.getName());
        }

        login.logOut();
    }
}
```

## Reference Guide

The following topics provide reference information for working with WSO2 API Manager:

- Product Profiles
- Default Product Ports
- Changing the Default Ports with Offset
- Error Handling
- WSO2 Patch Application Process

### Product Profiles

When a WSO2 product starts, it starts all components, features and related artifacts bundled with it. Multi-profile support allows you to run the product on a selected profile so that only the features specific to that profile along with common features start up with the server.

Given below are the different profiles available in WSO2 API Manager.

Profile	Command Option with Profile Name	Description
Gateway manager	-Dprofile=gateway-manager	<p>Starts only the components related to the API Gateway.</p> <p>You use this when the API Gateway acts as a manager node in a cluster. This profile starts frontend/UI features such as login as well as backend services that allow the product instance to communicate with other nodes in the cluster.</p>
Gateway worker	-Dprofile=gateway-worker	<p>Starts only the components related to the API Gateway.</p> <p>You use this when the API Gateway acts as a worker node in a cluster. This profile only starts the backend features for data processing and communicating with the manager node.</p>
Key Manager	-Dprofile=api-key-manager	Starts only the features relevant to the Key Manager component of the API Manager.
API Publisher	-Dprofile=api-publisher	Starts only the front end/backend features relevant to the API Publisher.
API Store	-Dprofile=api-store	Starts only the front end/backend features relevant to the API Store.

Note that the WSO2 products platform currently doesn't block/allow Web applications depending on profiles. Starting a product on a preferred profile only blocks/allows the relevant OSGI bundles. As a result, even if you start the server on a profile such as the `api-store` for example, you will still be able to access the API Publisher Web application.

Execute the following commands to start a product on any profile:

OS	Command
Windows	<code>&lt;PRODUCT_HOME&gt;/bin/wso2server.bat -Dprofile=&lt;preferred-profile&gt; --run</code>
Linux/Solaris	<code>sh &lt;PRODUCT_HOME&gt;/bin/wso2server.sh -Dprofile=&lt;preferred-profile&gt;</code>

#### How multi-profiling works

Starting a product on a preferred profile starts only a subset of features bundled in the product. In order to identify what feature bundles apply to which profile, each product maintains a set of `bundles.info` files in `<PRODUCT_HOME>/repository/components/<profile-name>/configuration/org.eclipse.equinox.simpleconfigurator` directories. The `bundles.info` files contain references to the actual bundles. Note that `<profile-name>`

e> in the directory path refers to the name of the profile. For example, when there's a product profile named webapp, references to all the feature bundles required for webapp profile to function are in a `bundles.info` file saved in `<PRODUCT_HOME>/repository/components/webapp/configuration/org.eclipse.equinox.simpleconfigurator` directory.

Note that when you start the server without using a preferred profile, the server refers to `<PRODUCT_HOME>/repository/components/default/configuration/org.eclipse.equinox.simpleconfigurator/bundles.info` file by default. This file contains references to all bundles in `<PRODUCT_HOME>/repository/components/plugins` directory, which is where all components/bundles of a product are saved.

## Default Product Ports

This page describes the default ports that are used for each WSO2 product when the `port offset` is 0.

 **Note** that it is recommended to disable the HTTP transport in an API Manager production setup. Using theearer token over HTTP is a violation of the OAuth specification and can lead to security vulnerabilities.

- Common ports
- Product-specific ports

### Common ports

The following ports are common to all WSO2 products that provide the given feature. Some features are bundled in the WSO2 Carbon platform itself and therefore are available in all WSO2 products by default.

#### **Management console ports**

WSO2 products that provide a management console use the following servlet transport ports:

- 9443 - HTTPS servlet transport (the default URL of the management console is <https://localhost:9443/carbon>)
- 9763 - HTTP servlet transport

#### **LDAP server ports**

Provided by default in the WSO2 Carbon platform.

- 10389 - Used in WSO2 products that provide an embedded LDAP server

#### **KDC ports**

- 8000 - Used to expose the Kerberos key distribution center server

#### **JMX monitoring ports**

WSO2 Carbon platform uses TCP ports to monitor a running Carbon instance using a JMX client such as JConsole. By default, JMX is enabled in all products. You can disable it using `<PRODUCT_HOME>/repository/conf/etc/jmx.xml` file.

- 11111 - RMIServer port. Used to monitor Carbon remotely
- 9999 - RMIServer port. Used along with the RMIServer port when Carbon is monitored from a JMX client that is behind a firewall

#### **Clustering ports**

To cluster any running Carbon instance, either one of the following ports must be opened.

- 45564 - Opened if the membership scheme is multicast
- 4000 - Opened if the membership scheme is wka

#### **Random ports**

Certain ports are randomly opened during server startup. This is due to specific properties and configurations that become effective when the product is started. Note that the IDs of these random ports will change every time the server is started.

- A random TCP port will open at server startup because of the `-Dcom.sun.management.jmxremote` property set in the server startup script. This property is used for the JMX monitoring facility in JVM.
- A random UDP port is opened at server startup due to the log4j appender (`SyslogAppender`), which is configured in the `<PRODUCT_HOME>/repository/conf/log4j.properties` file.

#### Product-specific ports

Some products open additional ports.

[API Manager](#) | [BAM](#) | [BPS](#) | [Complex Event Processor](#) | [Elastic Load Balancer](#) | [ESB](#) | [Identity Server](#) | [Message Broker](#) | [Storage Server](#) | [Enterprise Mobility Manager](#)

#### API Manager

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM/CEP: stat pub

 If you change the default API Manager ports with a port offset, most of its ports will be changed automatically according to the offset except a few exceptions described in the [APIM Manager documentation](#).

#### BAM

- 9160 - Cassandra port using which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM
- 7611 - Thrift TCP port to receive events from clients to BAM
- 21000 - Hive Thrift server starts on this port

#### BPS

- 2199 - RMI registry port (datasources provider port)

#### Complex Event Processor

- 9160 - Cassandra port on which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to CEP
- 7611 - Thrift TCP port to receive events from clients to CEP
- 11224 - Thrift TCP port for HA management of CEP

#### Elastic Load Balancer

- 8280, 8243 - NIO/PT transport ports

#### ESB

Non-blocking HTTP/S transport ports: Used to accept message mediation requests. If you want to send a request to an API or a proxy service for example, you must use these ports. `ESB_HOME/repository/conf/axis2/axis2.xml` file.

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

#### Identity Server

- 8000 - KDCServerPort. Port which KDC (Kerberos Key Distribution Center) server runs

- 10500 - ThriftEntitlementReceivePort

## Message Broker

Message Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5672 - Port for listening for messages on TCP when the AMQP transport is used.
- 8672 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1883 - Port for listening for messages on TCP when the MQTT transport is used.
- 8833 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7611 - The port for Apache Thrift Server.

## Storage Server

Cassandra:

- 7000 - For Inter node communication within cluster nodes
- 7001 - For inter node communication within cluster nodes via SSL
- 9160 - For Thrift client connections
- 7199 - For JMX

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port
- 50020 - Data node IPC server port

## Enterprise Mobility Manager

The following ports need to be opened for Android and iOS devices, so that it can connect GCM (Google Cloud Message) and APNS (Apple Push Notification Service) and enroll to WSO2 EMM.

Android :

The ports to open are 5228, 5229 and 5230. GCM typically only uses 5228, but it sometimes uses 5229 and 5230. GCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

iOS:

- 5223 - TCP port used by devices to communicate to APNs servers
- 2195 - TCP port used to send notifications to APNs
- 2196 - TCP port used by the APNs feedback service
- 443 - TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223

The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

API Manager:



The following WSO2 API Manager ports are only applicable to WSO2 EMM 1.1.0 onwards.

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports

## Changing the Default Ports with Offset

When you run multiple WSO2 products/clusters or multiple instances of the same product on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts. An offset defines the number by which all ports in the runtime (e.g., HTTP/S ports) will be increased. For example, if the default HTTP port is 9763 and the offset is 1, the effective HTTP port will change to 9764. For each additional WSO2 product instance, you set the port offset to a unique value. The offset of the default ports is considered to be 0.

There are two ways to set an offset to a port:

- Pass the port offset to the server during startup. The following command starts the server with the default port incremented by 3: `./wso2server.sh -DportOffset=3`
- Set the Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml`. E.g., `<Offset>3</Offset>`

Usually, when you offset the server's port, it automatically changes all ports it uses. However, there are few exceptions in the API Manager where you have to manually adjust some ports.

### ***Changing the Thrift client and server ports***

The port offset specified earlier in carbon.xml does not affect the ports of the Thrift client and server because Thrift is run as a separate server within WSO2 servers. Therefore, you must change the Thrift ports separately using `<ThriftClientPort>` and `<ThriftServerPort>` elements in the `<APIM_HOME>/repository/conf/api-manager.xml` file. For example, the following configuration sets an offset of 2 to the default Thrift port, which is 10397:

```

<!--
    Configurations related to enable thrift support for key-management related
    communication.

    If you want to switch back to Web Service Client, change the value of
    "KeyValidatorClientType" to "WSClient".
    In a distributed environment;
        -If you are at the Gateway node, you need to point "ThriftClientPort" value to
        the "ThriftServerPort" value given at KeyManager node.
        -If you need to start two API Manager instances in the same machine, you need
        to give different ports to "ThriftServerPort" value in two nodes.
        -ThriftServerHost - Allows to configure a hostname for the thrift server. It
        uses the carbon hostname by default.
    -->

    <KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
    <ThriftClientPort>10399</ThriftClientPort>
    <ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
    <ThriftServerPort>10399</ThriftServerPort>
    <!--ThriftServerHost>localhost</ThriftServerHost-->
    <EnableThriftServer>true</EnableThriftServer>

```

When you run multiple instances of the API Manager in distributed mode, the Gateway and Key Manager (used for validation and authentication) can run on two different JVMs. Communication between API Gateway and Key Manager happens in either of the following ways:

- Through a Web service call
- Through a Thrift call

The default communication mode is using Thrift. Assume that the Gateway port is offset by 2, Key Manager port by 5 and the default Thrift port is 10397. If the Thrift ports are changed by the offsets of Gateway and Key Manager, the

Thrift client port (Gateway) will now be 10399 while the Thrift server port (Key Manager) will change to 10402. This causes communication between the Gateway and Key Manager to fail because the Thrift client and server ports are different.

To fix this, you must change the Thrift client and server ports of Gateway and Key Manager to the same value. In this case, the difference between the two offsets is 3, so you can either increase the default Thrift client port by 3 or else reduce the Thrift server port by 3.

#### **Changing the offset of the Workflow Callback Service**

The API Manager has a Service which listens for workflow callbacks. This service configuration can be found at <AM\_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml. Open this file and change the port value of the <address uri accordingly.

For example,

```
<address
uri="https://localhost:9445/store/site/blocks/workflow/workflow-listener/ajax/workflow
-listener.jag" format="rest"/>
```

For a list of all default ports opened in WSO2 API Manager, see [Default Product Ports](#).

## **Error Handling**

When errors/exceptions occur in the system, the API Manager throws XML-based error responses to the client by default. To change the format of these error responses, you change the relevant XML file in the <AM\_HOME>/repository/deployment/server/synapse-configs/default/sequences directory. The directory includes multiple XML files, named after the type of errors that occur. You must select the correct file.

For example, to change the message type of authorization errors, open the <AM\_HOME>/repository/deployment/server/synapse-configs/default/sequences/\_auth\_failure\_handler.xml file and change application/xml to something like application/json.

```
<sequence name="_auth_failure_handler_" xmlns="http://ws.apache.org/ns/synapse">
<property name="error_message_type" value="application/json"/>
<sequence key="_cors_request_handler_"/>
</sequence>
```

Similarly, to change the error messages of throttling errors (e.g., quota exceeding), change the \_throttle\_out\_handler.xml file; resource mismatch errors, the \_resource\_mismatch\_handler.xml file, etc.

Given below are some error codes and their meanings.

#### **API handlers error codes**

Error code	Error Message	Description
900900	Unclassified Authentication Failure	An unspecified error has occurred
900901	Invalid Credentials	Invalid Authentication information provided
900902	Missing Credentials	No authentication information provided

900905	Incorrect Access Token Type is provided	The access token type used is not supported when invoking the API. The supported access token types are application and user accesses tokens. See <a href="#">Access Tokens</a> .
900906	No matching resource found in the API for the given request	A resource with the name in the request can not be found in the API.
900907	The requested API is temporarily blocked	The status of the API has been changed to an inaccessible/unavailable state.
900908	Resource forbidden	The user invoking the API has not been granted access to the required resource.
900909	The subscription to the API is inactive	Happens when the API user is blocked.
900910	The access token does not allow you to access the requested resource	Can not access the required resource with the provided access token. Check the valid resources that can be accessed with this token.
900800	Message throttled out	The maximum number of requests that can be made to the API within a designated time period is reached and the API is throttled for the user.
700700	API blocked	This API has been blocked temporarily. Please try again later or contact the system administrators.

#### Sequences error codes

Error code	Description
900901	Production/sandbox key offered to the API with no production/sandbox endpoint
403	No matching resource found in the API for the given request

In addition to the above error codes, we have engaged Synapse-level error codes to the default fault sequence and custom fault sequences (e.g.,\_token\_fault\_.xml) of the API Manager. For information, see [Error Handling](#) in WSO2 ESB documentation.

## WSO2 Patch Application Process

You apply patches to WSO2 products either as individual patches or through a service pack. A service pack is recommended when the number of patches increase. The following sections explain the WSO2 patch application process:

- Applying service packs to the Kernel
- Applying individual patches to the Kernel
- Verifying the patch application
- Overview of the patch application process



*Before you begin*

- You can download all WSO2 Carbon Kernel patches from [here](#).
- Before you apply a patch, check its `README.txt` file for any configuration changes required.

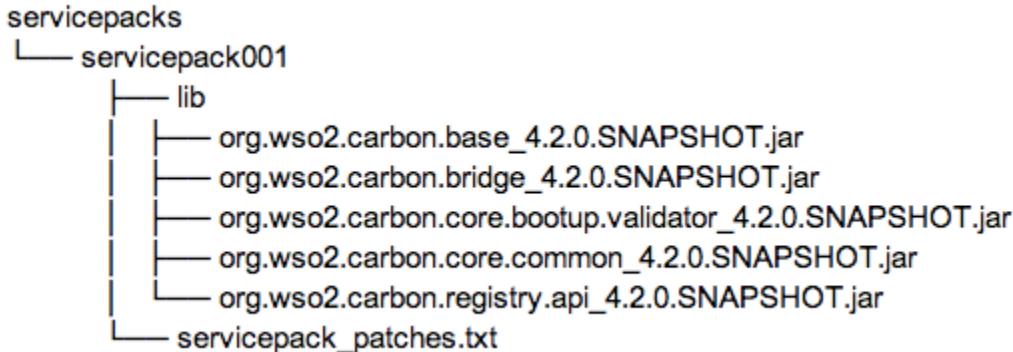
## Applying service packs to the product

Carbon 4.2.0 Kernel supports service packs. A service pack is a collection of patches in a single pack. It contains two elements:

- The `lib` directory: contains all the JARs relevant to the service pack.
- The `servicepack_patches.txt` text file: contains the list of JARs in the service pack.

Follow the steps below to apply service packs to your product.

1. Copy the service pack file to the `<PRODUCT_HOME>/repository/components/servicepacks/` directory. For example, the image below shows how a new service pack named `servicepack001` is added to this directory.



2. Start your product. The following steps will be executed:

- a. Before applying any patches, the process first creates a backup folder named `patch0000` inside the `<PRODUCT_HOME>/repository/components/patches/` directory, which will contain the original content of the `<PRODUCT_HOME>/repository/components/plugins/` directory. This step enables you to revert back to the previous state if something goes wrong during operations.
- b. The latest service pack in the `<PRODUCT_HOME>/repository/components/servicepacks/` directory will be applied. That is, the patches in the service pack will be applied to the `<PRODUCT_HOME>/repository/components/plugins/` directory.
- c. In addition to the service pack, if there are **individual patches** added to the `<PRODUCT_HOME>/repository/components/patches/` directory, those will also be incrementally applied to the `plugins` directory.

**(i)** The metadata file available in the service pack will maintain a list of the applied patches by service pack. Therefore, the metadata file information will be compared against the `<PRODUCT_HOME>/repository/components/patches/` directory, and only the patches that were not applied by the service pack will be incrementally applied to the `plugins` directory.

## Applying individual patches to the product

You can apply each patch individually to your system as explained below. Alternatively, you can apply patches through **service packs** as explained above.

1. Copy the patches to the `<PRODUCT_HOME>/repository/components/patches/` directory.
2. Start the Carbon server. The patches will then be incrementally applied to the `plugins` directory.

**⚠** Before applying any patches, the process first creates a backup folder named `patch0000` inside the `<PRODUCT_HOME>/repository/components/patches/` directory, which will contain the original

content of the <PRODUCT\_HOME>/repository/components/plugins/ directory. This step enables you to revert back to the previous state if something goes wrong during operations.

- i** Prior to Carbon 4.2.0, users were expected to apply patches by starting the server with `wso2server.sh -DapplyPatches`. Now, you do not have to issue a special command to trigger the patch application process. It starts automatically if there are changes in either the <PRODUCT\_HOME>/repository/components/servicepacks/ directory or the <PRODUCT\_HOME>/repository/components/patches/ directory. It verifies all the latest JARs in the servicepacks and patches directories against the JARs in the plugins directory by comparing MD5s of JARs.

### Verifying the patch application

After the patch application process is completed, the patch verification process ensures that the latest service pack and other existing patches are correctly applied to the <PRODUCT\_HOME>/repository/components/plugins/ folder.

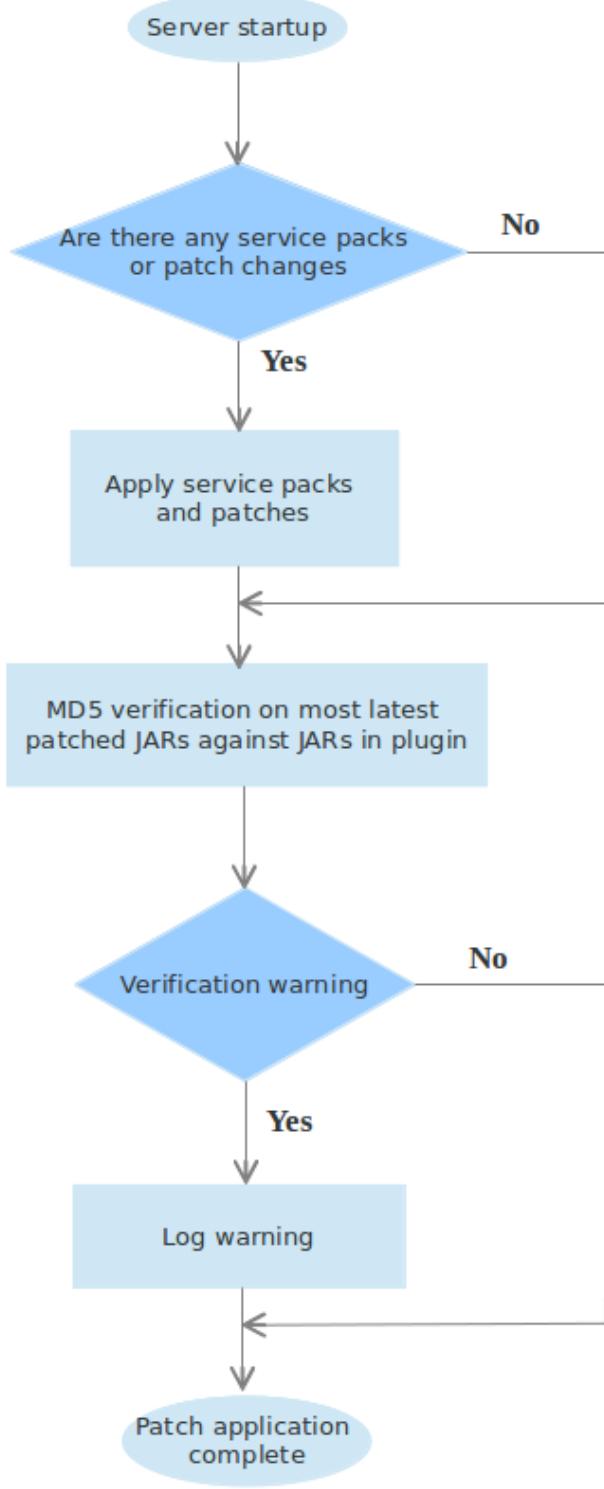
- All patch related logs are recorded in the <PRODUCT\_HOME>/repository/logs/patches.log file.
- The <PRODUCT\_HOME>/repository/components/patches/.metadata/prePatchedJARs.txt meta file contains the list of patched JARs and the md5 values.
- A list of all the applied service packs and patches are in the <PRODUCT\_HOME>/repository/components/default/configuration/prePatched.txt file.

**!** Do not change the data in the <PRODUCT\_HOME>/repository/components/default/configuration/prePatched.txt file. The patch application process gets the pre-patched list from this file and compares the list with the patches available in the servicepack and patches directories. If you change the data in this file, you will get a startup error when applying patches.

### Overview of the patch application process

The diagram below shows how the patch application process is implemented when you start the server.

## FAQ



- About WSO2 API Manager
- What is WSO2 API Manager?
- What is the open source license of the API Manager?
- How do I download and get started quickly?
- Is there commercial support available for WSO2 API Manager?
- What are the default ports opened in the API Manager?
- What are the technologies used underneath WSO2 API Manager?
- Can I get involved in APIM development activities?
- What is the default communication protocol of the API Manager?
- Installation and startup
- What are the minimum requirements to run WSO2 API Manager?
- What Java versions are supported by the API Manager?
- How do I deploy a third-party library into the API Manager?
- Do you provide automated installation scripts based on Puppet or similar solutions?
- Is it possible to connect the API Manager directly to an LDAP or Active Directory where the corporate identities are stored?
- Can I extend the management console UI to add custom UIs?
- I don't want some of the features that come with WSO2 API Manager. Can I remove them?
- How can I change the memory allocation for the API Manager?
- I don't want all the components of the API Manager up when I start the server. How do I start up only selected ones?
- Deployment and clustering
- Where can I look up details of different deployment patterns and clustering configurations of the API Manager?
- What is the recommended way to manage multiple artifacts in a product cluster?
- Is it recommended to run multiple WSO2 products on a single server?
- Can I install features of other WSO2 products to the API Manager?

- How can I set up a reverse proxy server to pass server requests?
- Functionality
  - I cannot see all the APIs that I published on the API Store. Why is this?
  - When editing an API's resource's parameters, how can I add multiple options to the parameter Response Content Type?
  - Why are the changes I did to the resource parameter Response Content Type of a published API not reflected in the API Store after saving?
  - I have set up the API Manager with WSO2 BAM to collect and analyze runtime statistics. But, the 'API

'Usage by Destination' graph shows no data. Why is this?

- How can I add more features to the API Manager server and extend its functionality?
- How do I change the pass-through transport configurations?
- If I want to extend the default API Manager server by installing new features, how can I do it?
- How can I preserve the CDATA element tag in API responses?
- Authentication and security
  - How can I manage authentication centrally in a clustered environment?
  - How can I manage the API permissions/visibility?
  - How can I add security policies (UT, XACML etc.) for the services?
  - How can I enable self signup to the API Store?
  - How can I disable self signup capability to the API Store? I want to engage my own approval mechanism.
  - Is there a way to lock a user's account after a certain number of failed login attempts to the API Store?
  - How do I change the default admin password and what files should I edit after changing it?
  - How can I recover the admin password used to log in to the management console?
  - How can I remove the authentication headers from the message going out of the API Gateway to the backend?
  - Can I give special characters in the passwords that appear in the configuration files?
  - How to protect my product server from security attacks caused by weak ciphers such as the Logjam attack (Man-in-the-Middle attack)?
- Troubleshooting
  - Why do I get the following warning:  
org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt while trying to authenticate APIKeyValidationService?
  - I hit the DentityExpansionLimit and it gives an error as  
{org.wso2.carbon.apimgt.hostobjects.APIStoreHostObject} - Error while getting Recently Added APIs Information. What is the cause of this?
  - I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?
  - When I add new users or roles, I get an error message as 'Entered user name is not conforming to policy'. What should I do?
  - When I call a REST API, I find that a lot of temporary files are created in my server and they are not cleared. This takes up a lot of space. What should I do?
- General questions
  - Can I implement an API facade with the API Manager?
  - How can I write automated test scripts for the API Manager?

## About WSO2 API Manager

### **What is WSO2 API Manager?**

WSO2 API Manager is a complete solution for creating, publishing and managing all aspects of an API and its life cycle. See [About API Manager](#).

### **What is the open source license of the API Manager?**

Apache Software License Version 2.0

### **How do I download and get started quickly?**

Go to <http://wso2.com/products/api-manager> to download the binary or source distributions. See [Tutorials](#).

### **Is there commercial support available for WSO2 API Manager?**

It is completely supported from evaluation to production. See [WSO2 Support](#).

### **What are the default ports opened in the API Manager?**

See [Default Ports of WSO2 Products](#).

***What are the technologies used underneath WSO2 API Manager?***

The API Manager is built on top of [WSO2 Carbon](#), an OSGi based components framework for SOA. See [component S](#).

***Can I get involved in APIM development activities?***

Not only are you allowed, but also encouraged. You can start by subscribing to [dev@wso2.org](mailto:dev@wso2.org) and [architecture@wso2.org](mailto:architecture@wso2.org) mailing lists. Feel free to provide ideas, feedback and help make our code better. For more information on contacts, mailing lists and forums, see [Getting Support](#).

***What is the default communication protocol of the API Manager?***

The default communication protocol is [Thrift](#).

---

**Installation and startup*****What are the minimum requirements to run WSO2 API Manager?***

Minimum requirement is Oracle Java SE Development Kit (JDK). See [Installation Prerequisites](#).

***What Java versions are supported by the API Manager?***

Oracle JDK 1.6.23 and above and JDK 1.7.\*.

***How do I deploy a third-party library into the API Manager?***

Copy any third-party JARs to <APIM\_HOME>/repository/components/lib directory and restart the server.

***Do you provide automated installation scripts based on Puppet or similar solutions?***

Yes. For information, [contact us](#).

***Is it possible to connect the API Manager directly to an LDAP or Active Directory where the corporate identities are stored?***

Yes. You can configure the API Manager with multiple user stores. See [Configuring User Stores](#).

***Can I extend the management console UI to add custom UIs?***

Yes, you can extend the management console (default URL is <https://localhost:9443/carbon>) easily by writing a custom UI component and simply deploying the OSGi bundle.

***I don't want some of the features that come with WSO2 API Manager. Can I remove them?***

Yes, you can do this using the **Features** menu under the **Configure** menu of the management console (default URL is <https://localhost:9443/carbon>).

***How can I change the memory allocation for the API Manager?***

The memory allocation settings are in <APIM\_HOME>/bin/wso2server.sh file.

***I don't want all the components of the API Manager up when I start the server. How do I start up only selected ones?***

Even though the API Manager bundles all components together, you can select which component/s you want to start by using the -Dprofile command at product startup. See [Product Profiles](#) for more information.

---

**Deployment and clustering*****Where can I look up details of different deployment patterns and clustering configurations of the API Manager?***

See [WSO2 clustering and deployment guide](#).

***What is the recommended way to manage multiple artifacts in a product cluster?***

For artifact governance and lifecycle management, we recommend you to use a shared [WSO2 Governance Registry](#) instance.

***Is it recommended to run multiple WSO2 products on a single server?***

This is not recommended in a production environment involving multiple transactions. If you want to start several WSO2 products on a single server, you must change their default ports to avoid port conflicts. See [Changing the Default Ports with Offset](#).

***Can I install features of other WSO2 products to the API Manager?***

Yes, you can do this using the management console. The API Manager already has features of WSO2 Identity Server, WSO2 Governance Registry, WSO2 ESB etc. embedded in it. However, if you require more features of a certain product, it is recommended to use a separate instance of it rather than install its features to the API Manager.

---

***How can I set up a reverse proxy server to pass server requests?***

See [Adding a Reverse Proxy Server](#).

---

## Functionality

***I cannot see all the APIs that I published on the API Store. Why is this?***

If you have multiple versions of an API published, only the latest version is shown in the API Store. To display multiple versions, set the `<DisplayMultipleVersions>` element to `true` in `<APIIM_HOME>/repository/conf/api-manager.xml` file.

***When editing an API's resource's parameters, how can I add multiple options to the parameter Response Content Type ?***

You cannot do this using the UI. Instead, edit the Swagger definition of the API as `content_type: ["text/xml", "text/plain"]` for example.

***Why are the changes I did to the resource parameter Response Content Type of a published API not reflected in the API Store after saving?***

If you edited the **Response Content Type** using the UI, please open the API's Swagger definition, do your changes and save. Then the changes should be reflected back in the API Store. This will be fixed in a future release.

***I have set up the API Manager with WSO2 BAM to collect and analyze runtime statistics. But, the 'API Usage by Destination' graph shows no data. Why is this?***

To populate this graph, you must enable destination-based usage tracking manually. See [Viewing API Statistics](#) on how to do that.

***How can I add more features to the API Manager server and extend its functionality?***

You can install any WSO2 component to the API Manager. See the [Installing Features](#) section in the WSO2 Carbon docs for more information.

***How do I change the pass-through transport configurations?***

If you have enabled the pass-through transport, you can change its default configurations by adding the following under the `<transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener">` element in the `<`

PRODUCT\_HOME>/repository/conf/axis2/axis2.xml file. Be sure to **stop the server** before editing the file.

1. If you are using JDK 1.6, add the parameter given below:

```
<transportReceiver name="passthru-https"
class="org.wso2.carbon.transport.passthru.PassThroughHttpSSLListener">
<parameter name="HttpsProtocols">TLSv1</parameter>
.....
</transportReceiver>
```

2. If you are using JDK 1.7, add the parameter given below:

```
<transportReceiver name="passthru-https"
class="org.wso2.carbon.transport.passthru.PassThroughHttpSSLListener">
<parameter name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
.....
</transportReceiver>
```

**If I want to extend the default API Manager server by installing new features, how can I do it?**

See [Feature Management](#) in the WSO2 Carbon documentation.

**How can I preserve the CDATA element tag in API responses?**

Set the javax.xml.stream.isCoalescing property to false in the <APIM\_HOME>/XMLInputFactory.properties file. Here's an example:

```
<XacuteResponse xmlns="http://aaa/xI">
<Rowset>
<Row>
<outxml><! [ CDATA[<inSequence>
<send>
<endpoint>
<address uri="http://localhost:8080/my-webapp/echo" />
</endpoint>
</send>
</inSequence> ]]></outxml>
</Row>
</Rowset>
</XacuteResponse>
```

## Authentication and security

**How can I manage authentication centrally in a clustered environment?**

You can enable centralized authentication using a WSO2 Identity Server based security and identity gateway solution, which [enables SSO](#) (Single Sign On) across all the servers.

**How can I manage the API permissions/visibility?**

To set visibility of the API only to selected user roles in the server, see [API Visibility](#).

**How can I add security policies (UT, XACML etc.) for the services?**

This should be done in the backend services in the Application Server or WSO2 ESB.

**How can I enable self signup to the API Store?**

See [how to enable self signup](#).

**How can I disable self signup capability to the API Store? I want to engage my own approval mechanism.**

To disable the self signup capability, open the APIM management console and click the **Resources -> Browse** menu. The registry opens. Navigate to `/_system/governance/apimgt/applicationdata/sign-up-config.xml` and set `<SelfSignUp><Enabled>` element to false. To engage your own signup process, see [Adding a User Signup Workflow](#).

**Is there a way to lock a user's account after a certain number of failed login attempts to the API Store?**

If your identity provider is WSO2 Identity Server, this facility comes out of the box. If not, install the identity-mgt feature to the API Manager and configure it. For information, see [Account Lock/Unlock](#) page in the Identity Server documentation.

**How do I change the default admin password and what files should I edit after changing it?**

To change the default admin password, log in to the management console with admin/admin credentials and use the "Change my password" option. After changing the password, do the following:

Change the following elements in `<APIM_HOME>/repository/conf/api-manager.xml` file:

```

<AuthManager>
    <Username>admin</Username>
    <Password>newpassword</Password>
</AuthManager>

<APIGateway>
    <Username>admin</Username>
    <Password>newpassword</Password>
</APIGateway>

<APIKeyManager>
    <Username>admin</Username>
    <Password>newpassword</Password>
</APIKeyManager>

```

Go to the **Resources -> Browse** menu in the management console to open the registry and update the credentials in `/_system/governance/apimgt/applicationdata/sign-up-config.xml` registry location.

**How can I recover the admin password used to log in to the management console?**

Use `<APIM_HOME>/bin/chpasswd.sh` script.

**How can I remove the authentication headers from the message going out of the API Gateway to the backend?**

Uncomment the `<RemoveOAuthHeadersFromOutMessage>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and set its value to true.

**Can I give special characters in the passwords that appear in the configuration files?**

If the config file is in XML, take care when giving special characters in the user names and passwords. According to XML specification (<http://www.w3.org/TR/xml/>), some special characters can disrupt the configuration. For example,

the ampersand character (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA ([http://www.w3schools.com/xml/xml\\_cdata.asp](http://www.w3schools.com/xml/xml_cdata.asp)) as shown below or remove the character:

```
<Password>
<! [ CDATA[ xnvYh?@VHAkc?qZ%Jv855&A4a , %M8B@h ] ]>
</Password>
```

#### **How to protect my product server from security attacks caused by weak ciphers such as the Logjam attack (Man-in-the-Middle attack)?**

You can disable weak ciphers as described in [Disable weak ciphers](#) in the WSO2 Carbon documentation.

#### Troubleshooting

##### **Why do I get the following warning: org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt while trying to authenticate APIKeyValidationService?**

Did you change the default admin password? If so, you need to change the credentials stored in the <APIKeyValidator> element of the <APIM\_HOME>/repository/conf/api-manager.xml file of the API Gateway node/s.

Have you set the priority of the SAML2SSOAuthenticator handler higher than that of the BasicAuthenticator handler in the authenticators.xml file? If so, the SAML2SSOAuthenticator handler tries to manage the basic authentication requests as well. Set a lower priority to the SAML2SSOAuthenticator than the BasicAuthenticator handler as follows:

```
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
    <Priority>0</Priority>
    <Config>
        <Parameter name="LoginPage">/carbon/admin/login.jsp</Parameter>
        <Parameter name="ServiceProviderID">carbonServer</Parameter>
        <Parameter
            name="IdentityProviderSSOServiceURL">https://localhost:9444/samlsso</Parameter>
            <Parameter
                name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
                <Parameter name="ISAuthnReqSigned">false</Parameter>
                <!--<Parameter
                    name="AssertionConsumerServiceURL">https://localhost:9443/acs</Parameter>-->
            </Config>
    </Authenticator>
```

##### **I hit the DentityExpansionLimit and it gives an error as {org.wso2.carbon.apimgt.hostobjects.APIStoreHostObject} - Error while getting Recently Added APIs Information. What is the cause of this?**

This error occurs in JDK 1.7.0\_45 and is fixed in JDK 1.7.0\_51 onwards. See [here](#) for details of the bug.

In JDK 1.7.0\_45, all XML readers share the same XMLSecurityManager and XMLLimitAnalyzer. When the total count of all readers hits the entity expansion limit, which is 64000 by default, the XMLLimitanalyzer's total counter is accumulated and the XMLInputFactory cannot create more readers. If you still want to use update 45 of the JDK, try restarting the server with a higher value assigned to the DentityExpansionLimit.

##### **I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?**

Set the `<parameter name="HostnameVerifier">` element to `AllowAll` in `<APIM_HOME>/repository/conf/axis2/axis2.xml` file's HTTPS transport sender configuration. For example, `<parameter name="HostnameVerifier">AllowAll</parameter>`.

This parameter verifies the hostname of the certificate of a server when the API Manager acts as a client and does outbound service calls.

***When I add new users or roles, I get an error message as 'Entered user name is not conforming to policy'. What should I do?***

This is because your user name or password length or any other parameter is not conforming to the RegEx configurations of the user store. See [Managing Users and Roles](#).

***When I call a REST API, I find that a lot of temporary files are created in my server and they are not cleared. This takes up a lot of space. What should I do?***

There might be multiple configuration context objects created per each API invocation. Please check whether your client is creating a configuration context object per each API invocation. Also, configure a HouseKeeping task in the `<APIM_HOME>/repository/conf/carbon.xml` file to clear the temporary folders. For example.

```
<HouseKeeping>
    <AutoStart>true</AutoStart>

    <!-- The interval in *minutes*, between house-keeping runs -->
    <Interval>10</Interval>

    <!-- The maximum time in *minutes*, temp files are allowed to live in the
        system. Files/directories which were modified more than
        "MaxTempFileLifetime" minutes ago will be removed by the house-keeping task
    -->
    <MaxTempFileLifetime>30</MaxTempFileLifetime>
</HouseKeeping>
```

## General questions

***Can I implement an API facade with the API Manager?***

You can use the API Manager and WSO2 ESB to implement an [API facade architecture pattern](#). WSO2 recommends this architecture if you are performing heavy mediation in your setup. For implementation details of an API facade, see [implementing an API facade with WSO2 API management platform](#).

As the API Manager does not have the ESB's GUI to perform mediation functions, you need to use the XML-based source view for configuration. Alternatively, you can create the necessary mediation sequences using the GUI of the ESB, and copy them from the ESB to the API Manager.

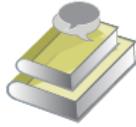
Also see [the following use cases](#) in WSO2 ESB documentation for more information on REST to SOAP conversion.

***How can I write automated test scripts for the API Manager?***

Use WSO2 Test Automation Framework (TAF) as explained in [Writing a Test Case for API Manager](#).

## Getting Support

In addition to this documentation, there are several ways to get help as you work on WSO2 products.

	<b>Explore learning resources:</b> For tutorials, articles, whitepapers, webinars, and other learning resources, look in the <b>Resources</b> menu on the <a href="#">WSO2 website</a> . In products that have a visual user interface, click the Help link in the top right-hand corner to get help with your current task.
	<b>Try our support options:</b> WSO2 offers a variety of development and production support programs, ranging from web-based support during normal business hours to premium 24x7 phone support. For support information, see <a href="http://wso2.com/support/">http://wso2.com/support/</a> .
	<b>Ask questions in the user forums</b> at <a href="http://stackoverflow.com">http://stackoverflow.com</a> . Ensure that you tag your question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you can't find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at <a href="http://wso2.org/mail">http://wso2.org/mail</a> .
	<b>Report issues</b> , submit enhancement requests, track and comment on issues using our public bug-tracking system, and contribute samples, patches, and tips & tricks (see the <a href="#">WSO2 Contributor License Agreement</a> ).

## Site Map

Use this site map to quickly find the topic you're looking for by searching for a title on this page using your browser's search feature. You can also use the search box in the upper right corner of this window to search for a word or phrase in all the pages in this documentation.