Version 2.1.2

Layer 7 OAuth Toolkit User Manual

LAYER 7 TECHNOLOGIES

1.800.681.9377
info@layer7tech.com
www.layer7tech.com

OASIS MEMBER

WS-I MEMBER

Contents

List of Figures

List of Tables

Chapter One: Getting Started

The Layer 7 OAuth Toolkit provides a full featured and standards compliant OAuth 1.0 and 2.0 solution.

This implementation conforms to the following specifications:

OAuth 1.0: http://tools.ietf.org/html/rfc5849

OAuth 2.0: http://tools.ietf.org/html/rfc6749

This implementation may provide incomplete support for the following draft specifications:

MAC: https://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-01#section-3.2

Base 64: https://tools.ietf.org/html/rfc4648#section-5

Note: Be careful when using the Layer 7 OAuth Toolkit against draft specifications. These specifications may change without notice, possibly causing the OAuth Toolkit to produce incorrect results.

Installing the OAuth Toolkit

The CA API Gateway - Policy Manager provides an installer that configures and installs the OAuth Toolkit.

➢ *To install the Layer 7 OAuth Toolkit:*

1. Start the Policy Manager.
2. Select the folder (or create a new folder) under which the OAuth Toolkit will be installed, in a folder named **OAuth**. If a folder is not selected, then the Toolkit will be installed off the root folder. For more information on working with folders, see "Services and Policies" in the *Layer 7 Policy Manager User Manual*.

   IMPORTANT: Do not install the OAuth Toolkit over any existing installation of the toolkit.

3. Select [**Tasks**] > **Additional Items** > **Install OAuth Toolkit in <*folder*>** from the Main Menu (on the browser client, from the Manage menu). The <*folder*> is the location of the **OAuth** folder from step 2. The OAuth Toolkit Installer dialog appears.

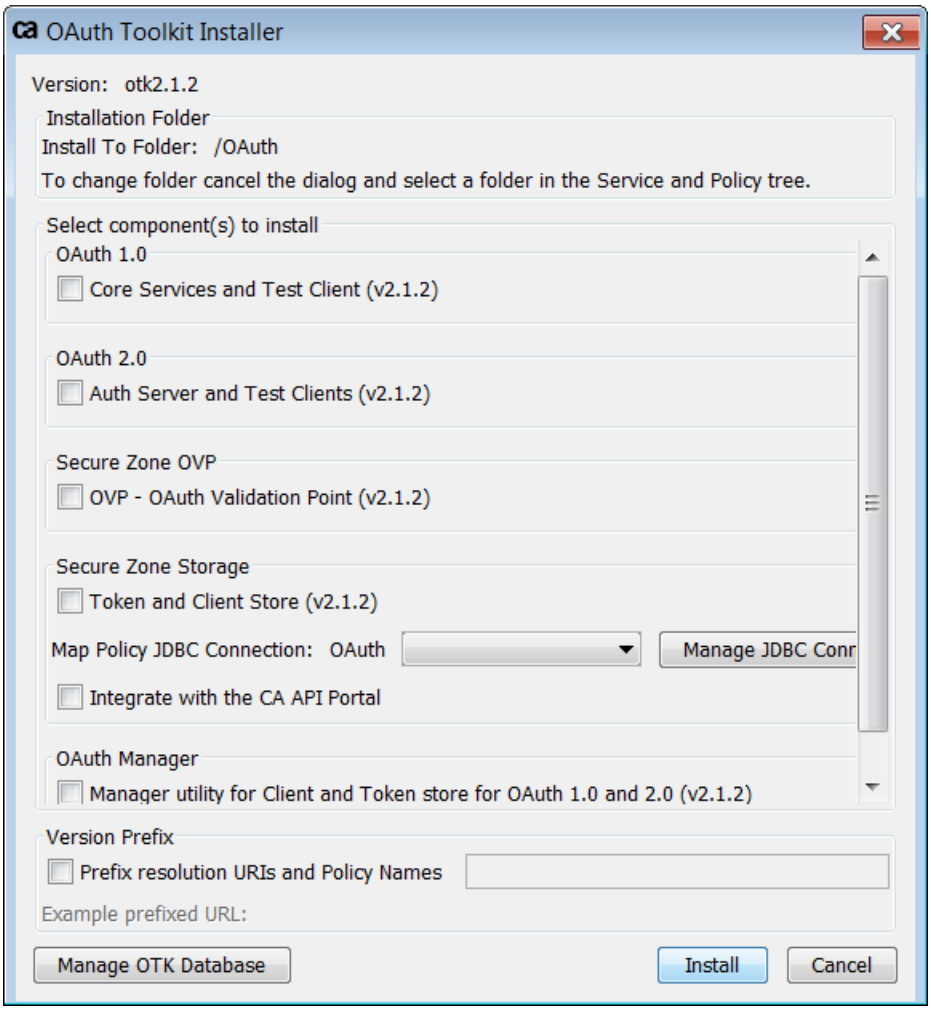

Figure 1: OAuth Toolkit Installer dialog

4. Configure the dialog as follows:

Table 1: OAuth Toolkit Installer settings

Setting	Description
Installation Folder	
Install to Folder	This displays the folder that was selected in step 2. This is where the OAuth Toolkit will be installed. If no folder was selected, then the default is to install into a folder named "OAuth" off of the root ("/"). **Note:** To use another folder, click [**Cancel**] to exit the installer dialog, select the other folder, and then restart the installer.
Select OAuth Toolkit components to install	
OAuth 1.0	Select the **Core Services and Test Client** check box to install the OAuth 1.0 core services and test clients.

Setting	Description
OAuth 2.0	Select the **Auth Server and Test Clients** check box to install the OAuth 2.0 authorization server and test clients. **Note:** Even if OAuth 2.0 is not selected for installation, you may still see a "OAuth 2.0" folder created. This folder stores policy fragments that may be required for validation purposes by the OAuth Toolkit.
SecureZone OVP	Select the **OVP - OAuth Validation Point** check box to install the SecureZone OAuth Validation Point (OVP) onto the Gateway. These endpoints provide validation services for the OAuth protocol for requests that must contain valid OAuth tokens. Clear the **OVP - OAuth Validation Point** check box if this Gateway is to be deployed in the DMZ. **Tip:** If you intend to use OAuth 2.0, you will need to have an OVP available for token validation. This OVP can either be installed on the same machine or be available via an external call. CA recommends deploying the OVP in your network's Trusted Zone on a separate Gateway, but you can also call out to your own OVP. For more information on the recommended deployment patterns, refer to "Understanding the OAuth Toolkit Architecture" on page 10.
SecureZone Storage	These endpoints provide an API for CRUD (Create, Read, Update, Delete) services for tokens and for OAuth client information. Session information is also stored in these folders. 1. Select the **Token and Client Store** check box to deploy the token store on the Gateway. This contains the client and token store endpoints. Clear the **Token and Client Store** check box if this Gateway is to be deployed in the DMZ. 2. Create a JDBC connection to the token and client store: a. Click [**Manage JDBC Connections**]. b. Click [**Add**]. c. Enter a **Connection Name** (for example, "OAuth"). d. Choose a **Driver Class**. If you are deploying the token store on a CA API Gateway, a common choice would be MySQL Community Edition ("com.mysql.jdbc.Driver"). e. Enter a **JDBC URL** (for example: "jdbc:mysql://localhost:3306/otk_db" for a token store located in a MySQL database called "otk_db" deployed on the local Gateway). f. Enter a **User Name** and **Password**. g. Click [**Test**] to test the connection to the token store. h. If the test succeeds, click [**OK**] and then click [**Close**]. For more information, see "Managing JDBC Connections" in the *Layer 7 Policy Manager User Manual*. **Note:** If no connection is chosen, then the default connection shown (**OAuth**) will be used. If this connection does not exist, it will be displayed as a missing item during the conflict resolution check. If you continue the installation, you must ensure that this connection is created later.

Setting	Description
	3. Select the **Integrate with the Layer 7 API Portal** check box if you are deploying this Gateway in conjunction with the Layer 7 API Portal. This will ensure that your deployment utilizes the OTK's OAuth capabilities, rather than those of the API Portal. Note the following once you enable this integration: • The "/oauth/clientstore/*" endpoint looks up client information via the Lookup API Key assertion, rather than from the OAuth Toolkit database. • The OAuth Manager can no longer be used to configure clients, as this is managed by the API Portal. **Note:** You will receive a warning if the Lookup API Key assertion is not available. If this happens, please contact CA Technical Support for assistance. **Tip:** If you intend to use OAuth 2.0, you will need to have a token store available. This token store can either be installed on the same machine or be available via an external call. CA recommends deploying the token store in your network's Trusted Zone on a separate Gateway, but you can also call out to your own token store. For more information on the recommended deployment patterns, refer to "Understanding the OAuth Toolkit Architecture" on page 10. To test your OAuth Toolkit installation, refer to Chapter Two, "Post Configuration".
OAuth Manager	Select the **Manager utility...** check box to install the Manager utility for the Client and Token store. This applies to both OAuth 1.0 and 2.0.
Version Prefix	
Prefix resolution URIs and Policy Names	Select this check box to add a prefix to the URIs and policy names. This is useful if you wish to version the installation or have a side-by-side installation. A prefix avoids naming conflicts in the routing URI and policy names. The prefix will be used to: • Prefix the resolution URI for each created published service (Note: All services use URI resolution in the OAuth Toolkit) • Update any services that route to an OAuth endpoint to route to the URI with the prefix instead • Prefix each created policy fragment name If a prefix is entered, an example URL is shown to illustrate how the prefix will be used.

Note: The components listed in the table above were designed to be installed on the same Gateway. If your environment requires that components be split across different Gateways, additional post configuration will be required. Contact CA Technical Support for assistance at support@layer7tech.com.

5. Click [**Manage OTK Database**] if you need to create an OTK database. Refer to "Installing the OTK Database" below for details.
6. Click [**Install**] to begin the installation of the OAuth Toolkit. An installation summary is displayed when installation is successfully completed.

If there are conflicts or missing items that prevent the OAuth Toolkit from being installed, these will be listed in a conflicts dialog. Correct the issues and try installing the toolkit again.

Installing the OTK Database

The Manage OTK Database dialog box is used to create a new OAuth Toolkit Database or to view/copy the OTK database schema.

➢ *To access the Manage OTK Database dialog box:*

1. Click [**Manage OTK Database**] on the OAuth Toolkit Installer. The following tabs are displayed in the Manage OTK Database dialog box:
   - **Create OTK Database:** Used to install the OTK database.
   - **OTK Schema:** Used to view or copy the OTK schema.
2. Complete the [Create OTK Database] tab (see Figure 2 and Table 2 below).
3. Click [**Close**] to close the Manage OTK Database dialog, if necessary.

Completing the [Create OTK Database] Tab

The [Create OTK Database] tab is used to create the MySQL OAuth Toolkit database and configure a JDBC Connection to the new database.

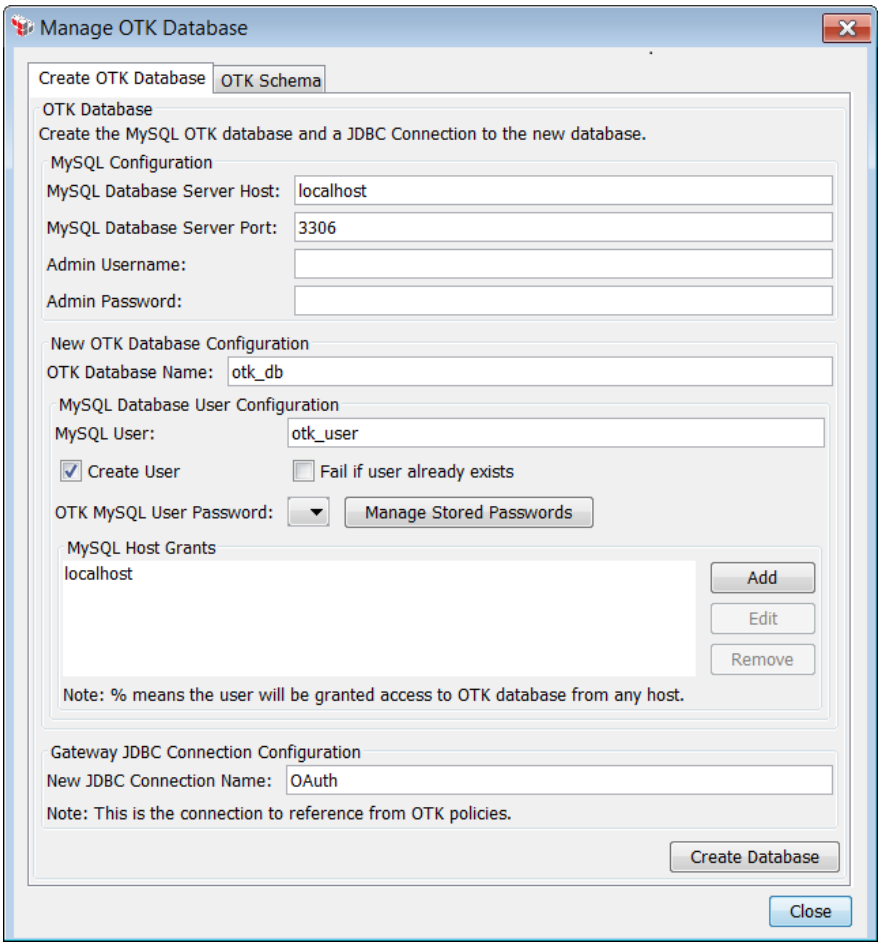

Figure 2: [Create OTK Database] tab

Complete the tab as follows:

Table 2: Configuring the [Create OTK Database] tab

Setting	Description
MySQL Configuration	
MySQL Database Server Host	Enter the hostname or IP address of the server where the MySQL database is located.
MySQL Database Server Port	Enter the port number for the MySQL database server.
Admin Username	Enter the user name of the admin user. **Note:** The user must have all the required privileges on the target server where the OTK database is installed. These privileges include: ALL and GRANT OPTION.

Setting	Description
Admin Password	Enter the password for the admin user.
New OTK Database Configuration	
OTK Database Name	Enter the name for the new OTK database, maximum 64 characters (for example, “otk_db”).
MySQL User	Enter the name of the MySQL user. You may specify an existing MySQL user or enter a new username. This is the name of the user you will be entering when you create a new JDBC connection. See “SecureZone Storage” in Table 1 on page 2 for more information on creating the JDBC connection.
Create User	Select this check box to treat the name of the MySQL User entered as a new user. A new user with this name will be created. Clear this check box to treat the name of the MySQL User entered as an existing user.
Fail if user already exits	When creating a new user, select this check box to prevent an existing user’s password from being overwritten. This is helpful if you have specified to create a new user, but accidentally entered an existing username. Clear this check box to create a new user even if a user with that name already exists. This will update the user’s password with the new one that you specify here.
OTK MySQL User Password	From the drop-down list, select the password for the MySQL user specified above. If the password you require is not listed, click [**Manage Stored Passwords**] to add it to the list of stored passwords. This is the password that the JDBC Connection uses. **Tip:** You cannot type the password directly here; it must be defined in the Gateway's secure password storage. To add a new stored password: 1. Click [**Add**]. 2. Enter a **Name** and then fill in the password fields. 3. Click [**OK**]. 4. Click [**Close**]. For more information, see “Manage Stored Passwords” in the *Layer 7 Policy Manager User Manual*.
MySQL Host Grants	Grant information is required for the MySQL user to access the new OTK database. By default, access from the **localhost** is configured. • Click [**Add**] to grant access to another hostname. Valid hostnames or IPV4 or IPV6 addresses are supported. Netmasks are also supported. • Click [**Edit**] to modify an existing hostname. • Click [**Remove**] to delete a hostname from the grant list.

Setting	Description
	Tip: You can enter "%" as a hostname to indicate that users can access the OTK database from any host.
Gateway JDBC Connection Configuration	
New JDBC Connection Name	Enter a name for the JDBC Connection that will be created, maximum 128 characters. This is the name of the JDBC connection you will enter when creating a new JDBC connection. See "SecureZone Storage" in Table 1 on page 2 for more information on creating the JDBC connection. After the database has been created, a new JDBC Connection with the specified name will be created, which can then be referenced from policies. This name will be visible in the "Map Policy JDBC Connection" drop-down list in Figure 1 and will also be visible in the *Manage JDBC Connections* task in the Policy Manager. The connection will be configured to use a secure password reference for the MySQL user's password.

Click [**Create Database**] to install the database schema using the supplied parameters.

When the database is created, you can view it in the [OTK Schema] tab or you can click [**Close**] to close the Manage OTK Database dialog box.

Using the [OTK Schema] Tab

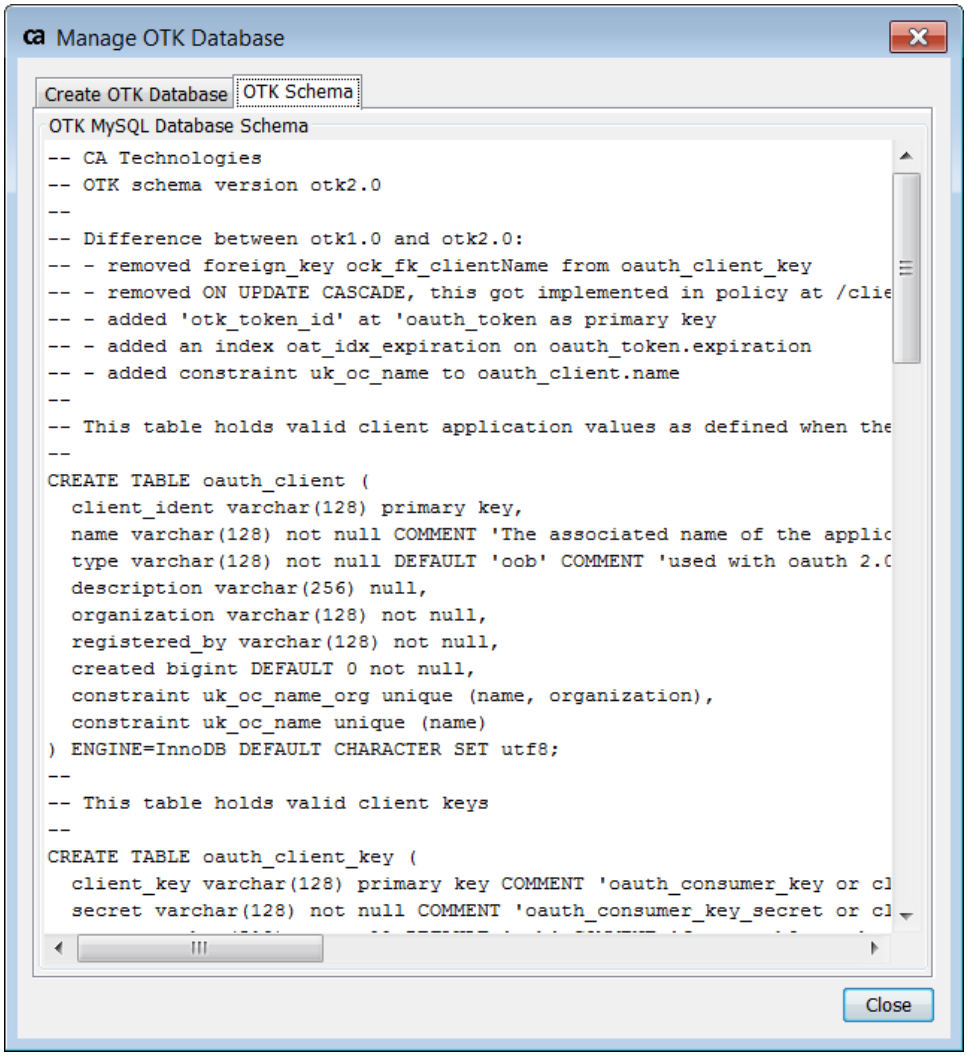

Figure 3: [OTK Schema] tab

The [OTK Schema] tab simply displays the database schema for the OAuth Toolkit database. You can view the schema or copy the contents via the Clipboard.

Conflict Resolution

The OAuth Toolkit Installer checks for potential conflicts in the following areas before the installation begins:

- Service routing conflicts
- Policy conflicts
- Certificate conflicts
- Encapsulated Assertion conflicts

- Missing JDBC connections
- Missing assertions

If conflicts are detected, you may either continue or cancel the installation. Continuing an installation with detected conflicts will result in the following:

- Any service with a conflicting URI will not be installed.
- Any policy fragment with a conflicting name will not be installed.
- Any reference to a conflicting fragment will instead reference the existing fragment.

Alternatively, you can cancel the installation when conflicts are detected and either uninstall the conflicting services/policies or choose a different install prefix to avoid the conflict.

IMPORTANT: If you cancel the installation while items are actively being installed, you may need to manually delete items that were installed—canceling does not perform a rollback.

Uninstalling the OAuth Toolkit

To uninstall the OAuth Toolkit, delete the install folder and then manually delete the database.

Understanding the OAuth Toolkit Architecture

The OAuth Toolkit comes separated in logically different components. This section explains how the components can be introduced within the network environment.

Conceptually these components are available:

- OAuth Validation Point (OVP)
    - This is an endpoint that validates incoming requests for OAuth 1.0 and OAuth 2.0. It is accessed via a REST API.
- API Proxy
    - The CA API Gateway holding the OAuth installation enforcing the OAuth token requirement.
- Clientstore
    - All oauth_consumer_keys (OAuth 1.0) and client_ids (OAuth 2.0) are stored here. The clientstore is accessable via a REST API.
- Tokenstore
    - All tokens are stored here. The clientstore is accessable via a REST API.

- Sessionstore
  - This endpoint provides caching and session services to the OTK components. This allows OTK components to avoid going to the database in calls to clientstore and tokenstore APIs.
- Resource Server
  - This server provides endpoints to access resources. These endpoints require a valid OAuth token.

See the following graphic which displays the components within their preferred network zones.

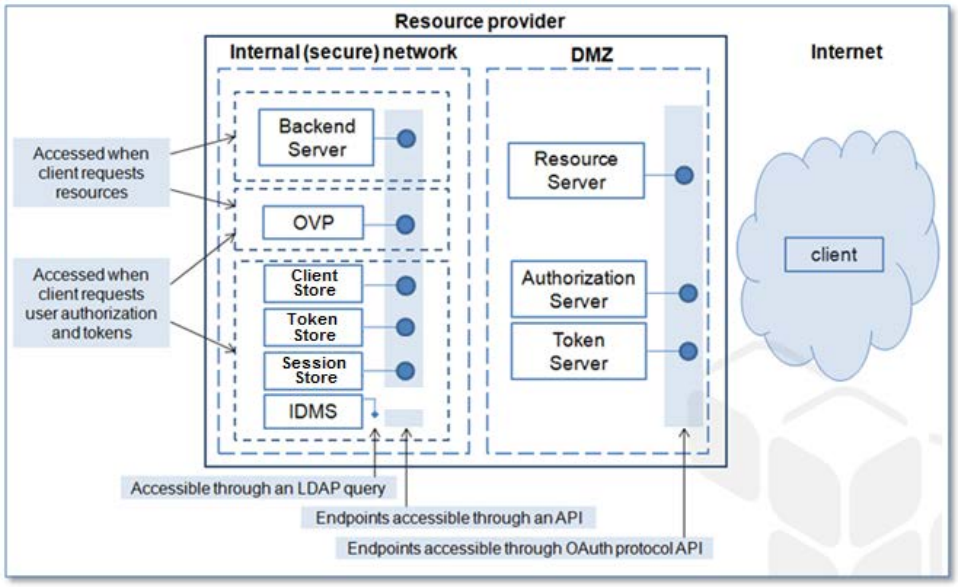

Figure 4: Components within their preferred network zones

Getting Help

The Layer 7 OAuth Toolkit is a working OAuth implementation designed for maximum flexibility. This allows you to configure it for your particular performance requirements and deployment environment.

For professional help with tailoring the Layer 7 OAuth implementation to meet your needs, please contact CA Technical Support at support@layer7tech.com.

Chapter Two: Post Configuration

After the OAuth Toolkit is installed, it is recommended that you perform the post-configuration steps to ensure that everything is set up correctly.

Post Configuration Steps

Step 1: Import Public Certificate

The Gateway needs to "trust" its own SSL certificate before you can use the test clients. To do this, import the Gateway's public certificate into the certificate store of the Gateway.

➢ *To import the public certificate:*

1. In the Policy Manager, choose [**Tasks**] **> Manage Certificates**. The Manage Certificate dialog appears.
2. Click [**Add**]. The Add Certificate Wizard appears.
3. In Step 1 of the wizard, choose **Retrieve via SSL Connection (HTTPS or LDAPS URL)** and then enter **https://localhost:8443** in the adjacent field.
4. Click [**Next**] to display wizard in Step 2.
5. Click [**Next**] to display wizard in Step 3.
6. Select the check boxes for:

   Outbound SSL Connections
   Signing Certificates for Outbound SSL Connections
   Signing Client Certificates
   Signing SAML Tokens
7. Click [**Finish**] to complete the wizard.
8. Click [**Close**] to close the Manage Certificates dialog.

Step 2: Verify the OAuth Database Schema

Verify the database schema by using the [OTK Schema] tab in the Manage OTK Database dialog box. For more information, refer to "Using the [OTK Schema] Tab" on page 9.

Step 3: Configure Authentication

By default, the policies installed by the OAuth Toolkit are not usable until you configure authentication for the endpoints. This process involves the following steps:

1. Implement a Federated Identity Provider for the SAML Grant Type
2. Implement a Federated Identity Provider for Validation and Storage Endpoints
3. Add Validation of SAML Token Signer to SAML Token Grant_Type Policies
4. Add Checking of Client Certificate to Validation and Storage Endpoints

Step 1: Implement a Federated Identity Provider for the SAML Grant Type

By default, support for the SAML Token grant type is disabled in the policies delivered in the OAuth Toolkit. Do the following if you intend to enable support for this grant type:

1. Run the Manage Trusted Certificates task to ensure that any needed certificates have been imported. This includes the Gateway's own default SSL certificate and the SSL certificate of any Gateway that is connecting as a client.

   For more information, see *Managing Trusted Certificates* in the *Layer 7 Policy Manager User Manual*.

2. Create a Federated Identity Provider with the following attributes:

Wizard Step	Setting	Value
1	Provider Name Credential Source Type Allowed	**OAuth SAML Identity Provider** **SAML Token**
2	Trusted Certificates	Add the trusted certificates of the SAML issuing parties: • Gateway's own default SSL certificate • SSL certificate of any Gateway that is connecting as a client
3	Certificate Validation Options	**Validate Certificate Path**

   For more information, see *Federated Identity Provider Wizard* in the *Layer 7 Policy Manager User Manual.*

Step 2: Implement a Federated Identity Provider for Validation and Storage Endpoints

1. If not already done in Step 1, run the Manage Trusted Certificates task to ensure that any needed certificates have been imported. This includes the Gateway's own default SSL certificate and the SSL certificate of any Gateway that is connecting as a client.

   For more information, see *Managing Trusted Certificates* in the *Layer 7 Policy Manager User Manual*.

2. Create a Federated Identity Provider with the following attributes:

Wizard Step	Setting	Value
1	Provider Name Credential Source Type Allowed	**OAuth Client Identity Provider** **X.509 Certificate**
2	Trusted Certificates	Do not add any trusted certificates. Confirm when warned that all users will need their certificates to be imported individually.
3	Certificate Validation Options	**Validate Certificate Path**

   For more information, see *Federated Identity Provider Wizard* in the *Layer 7 Policy Manager User Manual.*

3. For each client that will be connecting to the validation and storage endpoints (possibly including this Gateway itself), create a Federated User within this new FIP with (as their imported certificate) the client's certificate for its outbound TLS connection:

   a. Right-click on the new "OAuth Client Identity Provider" and select **Create User**. The Create Federated User dialog appears.

   b. Complete the dialog as follows:

      - **X509 Subject DN:** Enter the complete DN of the client certificate that will be imported for this user. For example, if the client Gateway has a certificate with a DN of "CN=gateway.example.com" then you must enter "CN=gateway.example.com" here.
      - **Login/Email/User Name:** Complete as necessary.
      - **Define Additional Properties:** Select check box.

   c. In the Properties dialog for the user, select the Certificate tab.

   d. Click **Import** and then import the SSL certificate of the client Gateway as this user's certificate. For example:

- If the Gateway hosting the validation and storage endpoints is connecting to itself over localhost, select the **Import from Private Key's Certificate Chain** option and choose the default SSL key (which by default has the alias "ssl").
- If an external client Gateway will be connecting to the validation and storage endpoints, select the **Retrieve via SSL Connection (HTTPS or LDAPS URL)** option and type a URL that leads to a listen port on the client Gateway that uses its default SSL key as its server certificate. For example, "https://clientgateway.example.com:8443".

4. Save the new Federated User.

For more information, refer to these topics in the *Layer 7 Policy Manager User Manual*:

Creating a Federated User
Federated User Properties

Step 3: Add Validation of SAML Token Signer to SAML Token Grant_Type Policies

By default, support for the SAML Token grant type is disabled in the policies delivered in the OAuth Toolkit. Do the following if you need to enable support for this grant type:

1. In the Policy Manager, connect to the Gateway that implements the OAuth Toolkit endpoint */auth/oauth/v2/token*.
2. Replace the Stop Processing assertion in the SAML grant type branch with an Authenticate Against Identity Provider assertion.
   - Choose **OAuth SAML Identity Provider** as the identity provider and then close the assertion properties.
3. Right-click the Authenticate Against Identity Provider assertion in the policy window and choose **Select Target Message**.
4. Set the message target to "Other Context Variable" named **bearerToken**.

The assertion label in the policy should now read: *${bearerToken}: Authenticate against OAuth SAML Identity Provider*.

Step 4: Add Checking of Client Certificate to Validation and Storage Endpoints

In this step, you will update the OAuth storage and validation endpoints to use a Federated Identity Provider to check the client certificate.

1. In the Policy Manager, connect to the Gateway that implements the validation and storage endpoints.
2. Open each endpoint listed below and replace the Stop Processing assertion at the top of the policy with an Authenticate Against Identity Provider assertion.

- Choose the **OAuth Client Identity Provider** as the identity provider.

Update these endpoints within the folder "SecureZone – OVP":

/oauth/validation/v1/authorize
/oauth/validation/validate/v1/signature
/oauth/validation/v2/authorize
/oauth/validation/validate/v2/granttype
/oauth/validation/validate/v2/refreshtoken
/oauth/validation/validate/v2/token
/oauth/validation/validate/v2/tokenrequest

Update these endpoints within the folder "SecureZone – Storage":

*/oauth/clientstore/**
*/oauth/tokenstore/**
*/oauth/session/**

Step 4: Verify the OAuth Manager

After the components are installed, verify that the OAuth Manager is working correctly. This manager is used for both OAuth 1.0 and 2.0.

Prerequisite: To perform the OAuth Manager and OAuth Test Client verification, you will need a browser that has access to the endpoints configured on the CA API Gateway.

To verify that the OAuth Manager is working correctly, run through the steps under "Manage Clients" and "Manage Tokens" under "Working with the OAuth Manager" starting on page 21.

Step 5: Verify the OAuth Test Client

You should also run the appropriate test client after installation to ensure that it is working correctly.

➢ *To verify the OAuth Test Client:*

If you have integrated the OAuth Toolkit with the Layer 7 API Portal, be sure to follow the steps under "Integration with API Portal" below first.

- To verify the test client for OAuth 1.0, see "OAuth 1.0 Test Client" on page 26".
- To verify the test client for OAuth 2.0, see "OAuth 2.0 Test Client" on page 29.

Integration with API Portal

If you chose to integrate the OAuth Toolkit with the Layer 7 API Portal (see Figure 1 on page 2), you must perform the following steps before you can test your deployment:

1. Start the Policy Manager and connect to your Gateway.

2. Expand the “API Portal Integration” folder in the list of services and folders (lower-left corner of the Policy Manager).
3. Expand the “SecureZone - Storage” folder.
4. Expand the “OAuthClientStore” folder.
5. Double-click **oauth/clients [/oauth/clientstore/*]**. The policy associated with this endpoint is displayed in the policy window.
6. Click [**Show Assertion Numbers**] in the Policy Toolbar (upper-right corner of the Policy Manager). This displays line numbers next to each line in the policy.
7. Double-click line 6 and then change the value in the **Expression** text box from “true” to “false”.
8. Click [**OK**] to close the properties.
9. Click [**Save and Activate**] in the Policy Toolbar. You can now verify the test clients.

IMPORTANT: After verifying the test clients, repeat steps 1-9, but this time for step 7, reset the Expression value from “false” to “true”.

Chapter Three:
Using the OAuth Toolkit

The OAuth toolkit provides an implementation of both OAuth 1.0 and 2.0. Additionally test clients are provided for both and a simple client and token management application.

Note: The OAuth Toolkit currently does not perform any SCOPE validation. Be sure to manually verify that clients who request a SCOPE value are actually registered to receive it.

Working with the OAuth Manager

The OAuth Manager is a basic tool that displays information about registered client applications and users that have granted applications access to their resources.

Note: The OAuth Manager was designed as a simple tool to help get customers started with OAuth. It was not designed to manage their enterprise OAuth solution.

The OAuth Manager provides these two functions:

- Manage Clients and Client Keys
- Manage Tokens

The OAuth Manager displays registered client applications and applications that were granted by users.

Manage Clients

Note: Client management is not available via the OAuth Manager if the Layer 7 OAuth Toolkit has been integrated with the Layer 7 API Portal (see Figure 1 on page 2).

➢ *To manage clients using the OAuth Manager:*

1. Open a browser and navigate to this URL:

   https://<*Gateway_host*>**:8443/oauth/manager**

   The OAuth Manager welcome screen is displayed.

2. Click [**Manage Clients**] to list, delete, and register client applications.

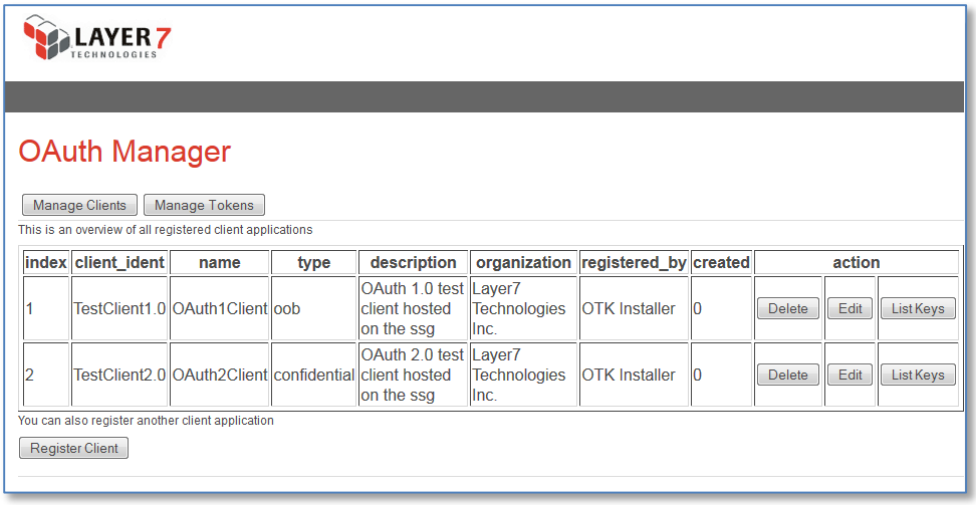

Figure 5: OAuth Manager - Manage Clients

The following table describes each element on the OAuth Manager screen for client applications.

Table 3: Descriptions for OAuth Manager - Manage Clients section

Field	Description
client_ident	A unique identifier of this client. It is unique within the used client store
name	The name of the client application This name will be displayed in the grant/deny dialog during the authorization process.
type	The type is either “public” or “confidential”. Used with OAuth 2.0.
description	An optional description of this client
organization	The organization that has registered this client
registered_by	The user that has registered this client
created	The time this client was registered. The provided text clients have “0” as the value
action	• **Delete:** Click this to delete the client. Use this action with care, as deleting the client application will also delete all tokens issued for this client. • **Edit:** Click this to edit the values on the Manage Client screen. Edit with caution—for example, if you change the consumer_key or the consumer_key_secret, existing client applications will no longer be able to access a protected resource endpoint. • **List Keys:** Will display a list of existing client keys for this client
register client	Displays the client registration screen (). Complete the fields and then click [**Register**]. The application is now available within the environment.

The registration page is used for OAuth 1.0 and OAuth 2.0 clients. Some fields may only apply to one of the types. Values for handling "SCOPE" and "ENVIRONMENT" are collected but rules for validating these values must be implemented. There is no general rule defined in the OAuth specifications for this.

3. Click [**List Keys**] to view the client_keys.

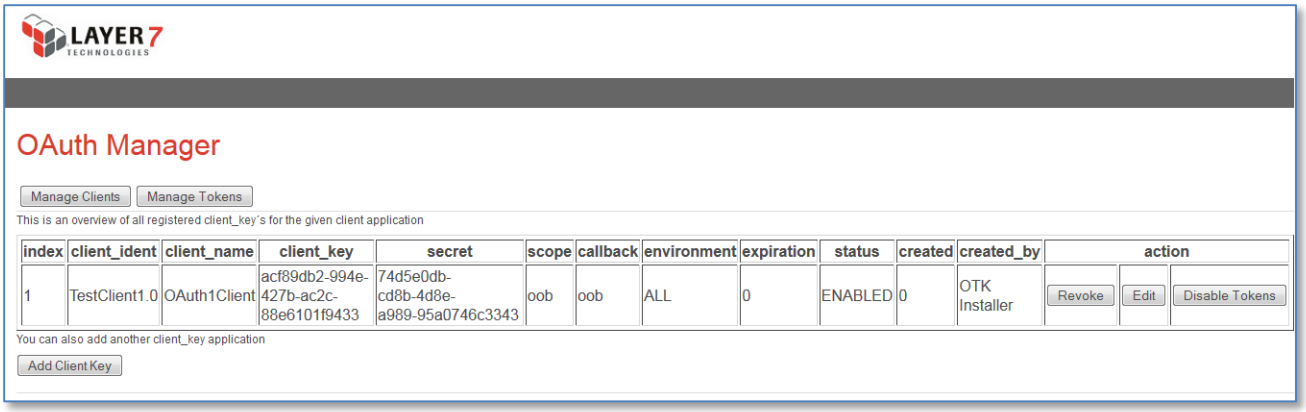

Figure 6: OAuth Manager - Manage Clients - List Keys

This page is used for OAuth 1.0 and OAuth 2.0 clients. Some fields may only apply to one of the types.

Table 4: Field descriptions for OAuth Manager - Manage Clients – List Keys section

Field	Description
client_ident	The unique identifier of the "owning" client
client_name	The name of the "owning" client
client_key	In OAuth 1.0 it is the "oauth_consumer_key"; in OAuth 2.0 it is the "client_id"
secret	The associated client key secret
scope	The scope of this key
callback	In OAuth 2.0 this is "redirect_uri". In Oauth 1.0 it can either hold "oob" or a valid URI. In OAuth 2.0 this can be one or a list of URIs
environment	The environment used with this key
expiration	The date until this key is valid. "0" indicates "forever"
status	Either "ENABLED" or "DISABLED". Disabled tokens will cause resource requests to be denied
created	The time this token was created
created_by	The user who created this key

4. Select an action to perform:
   - **Revoke:** Click this to revoke (delete) the client key. All tokens for that client key will also be revoked.
   - **Edit:** Click this to edit the key—for example, to change the STATUS, or to disable the key.

     Note: Disabling a client key will *not* disable the tokens for that client key. It only prevents the client key from being used for any future tokens.
   - **Disable Tokens:** Click this to disable all tokens for the client key. Note: To disable and re-enable individual tokens, see Figure 7.

Manage Tokens

➢ *To manage tokens using the OAuth Manager:*

1. Open a browser and navigate to this URL:

   https://<*Gateway_host*>**:8443/oauth/manager**

   The OAuth Manager screen is displayed.
2. Click [**Manage Tokens**] to view values of issued tokens, and to disable or revoke tokens.

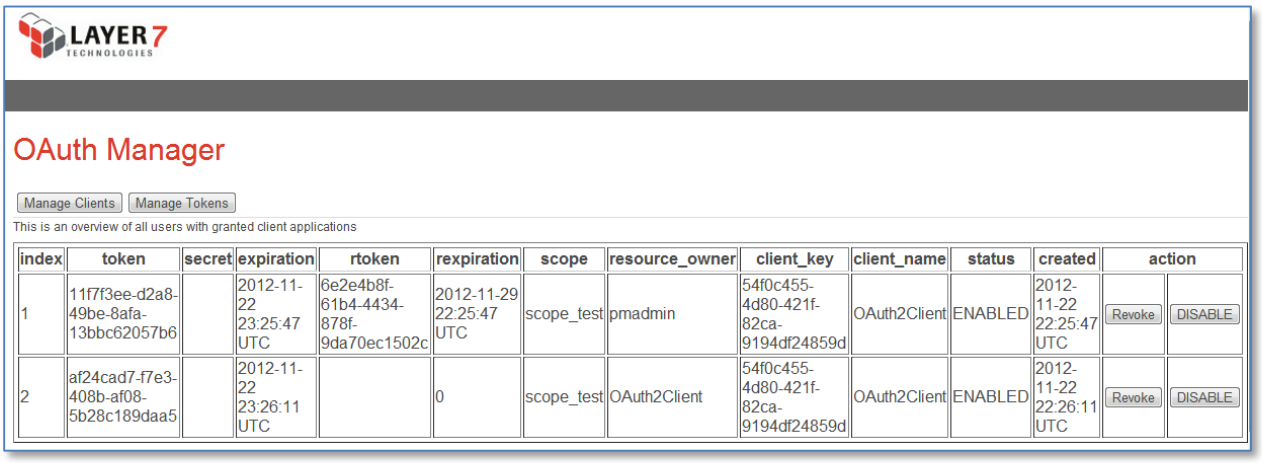

Figure 7: OAuth Manager - Manage Tokens

The following table describes each element on the OAuth Manager screen for tokens.

Table 5: Descriptions for OAuth Manager - Tokens section

Field	Description
token	The *access_token* issued to the client application when it was granted
secret	The secret of this *access_token* if available
expiration	The token is valid until this time
rtoken	A *refresh_token* if available
rexpiration	The expiration date of the refresh token
scope	The scope for this token
resource_owner	The user who has granted this token
client_key	In OAuth 1.0 it is the "oauth_consumer_key"; in OAuth 2.0 it is the "client_id"
client_name	The associated client name
status	Either "ENABLED" or "DISABLED". Disabled tokens will cause resource requests to be denied
created	The time this token was created

3. Select an action to perform:
   - **Revoke:** Click this to revoke (delete) the token.

     Note: Expired tokens are automatically deleted when a new token is issued. This is performed by the policy fragment "Delete expired tokens", which is invoked each time the "/oauth/tokenstore/*" endpoint is called. If you experience performance issues related to deleting expired tokens, disable this fragment in the oauth/tokens policy and call it in a different manner.
   - **DISABLE:** For enabled tokens, click this to disable the token.
   - **ENABLE:** For disabled tokens, click this to re-enable the token

Working with the OAuth 1.0 Toolkit

OAuth 1.0 Test Client

The test client is used to verify installation changes and to access OAuth 1.0/1.0a-secured API endpoints of platforms. This section describes how it works and how it can be configured.

Note the following security precautions when using the test client:

- The test client should *not* be installed on product systems.
- The test client should *not* be installed on a Gateway that is available on the internet.
- You should modify the test client to use your own specific client credentials.
- You should remove the test client from the OAuth Manager when it is no longer needed.

➢ *To run the test client:*

- Open a browser and navigate to this URL:

  https://<*Gateway_host*>**:8443/oauth/v1/client**

  The OAuth V1 Client welcome screen is displayed.

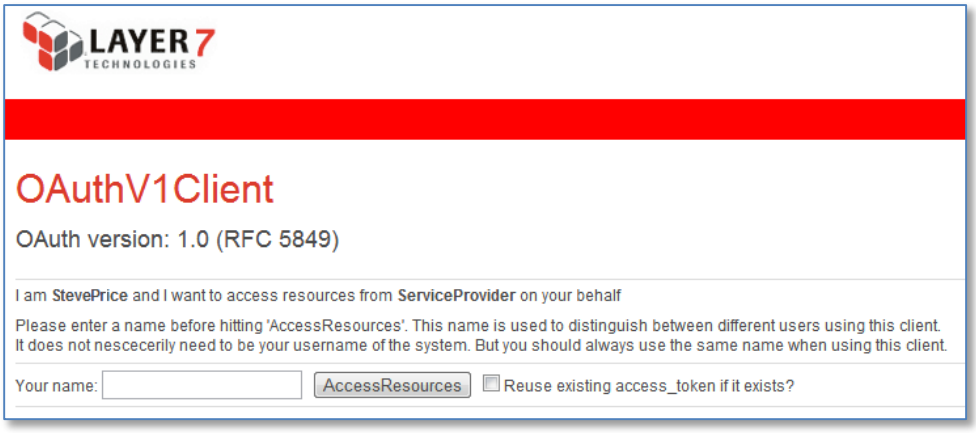

Figure 8: OAuth V1 Client

The following table describes each setting on the test client.

Table 6: Test Client setting descriptions

Setting	Description
Your name	The client will use this name to associate received tokens with the current user. This can be any name; it is not verified and is easily changed. This makes it useful if more than one developer is using the client at the same time. The default user name "system" is used if none was provided.

Setting	Description
AccessResources	The client will request a *request_token* at the authorization server.
Reuse existing access_token if it exists?	Select this check box to reuse an *access_token* for the current *consumer_key*, if the user has already received one from an earlier session. If the token exists, the client will directly access the resources. Clear this check box to not reuse an existing *access_token* and instead reissue a new one. **Notes:** (1) The OAuth token lifetime is configurable. (2) The client does not know whether the token has expired. If it has, the server will return either nothing or an error message and another session must be started.

Verifying the Test Client

Before running the test client for the first time, you should test that it is working properly.

➢ *To verify the test client:*

1. Start the test client.
2. For the purposes of testing, you may click [AccessResources] with entering a name. You will see a notification stating that "system" will be used as the username. Click [OK] to confirm proceeding without a username. You are redirected to the authorization endpoint of the OAuth server, which also is the resource server. Before the redirect occurs, the client has already received a *request_token* issued by the request endpoint.
3. The next screen displays the login page of the Authorization server. Enter the login credentials of a user from the internal identity provider and then click [Grant].

   You will be redirected back to the client application. The client will receive the *authorized request_token* and a *verification code*. The client uses those values to exchange them for an *access_token*.
4. After the client receives the *access_token,* the download page is displayed. Click [Download] to receive the requested resources.

Figure 9: OAuth Client: Downloading the requested resources

At this point, the client has used the issued *access_token* and the test client verification is complete.

Figure 10: OAuth Client: Receiving the requested resources

Configuring the Test Client to access other OAuth protected services

The following configurations can be modified:

- Application name
- consumer_key, consumer_key_secret
- resource endpoint
- request, authorize and token endpoint

All modifications can be made in the policy using the Policy Manager in the /oauth/v1/client policy.

Working with the OAuth 2.0 Toolkit

All OAuth 2.0 features are implemented using policies in the Policy Manager, making it possible to customize the entire OAuth process. This chapter describes some of the more common customizations.

OAuth 2.0 Test Client

The test client is used to verify installation changes and to access OAuth 2.0-secured API endpoints of platforms. This section describes how it works and how it can be configured.

Note: Client management is not available via the OAuth Manager if the Layer 7 OAuth Toolkit has been integrated with the Layer 7 API Portal (see Figure 1 on page 2).

Note the following security precautions when using the test client:

- The test client should *not* be installed on product systems.
- The test client should *not* be installed on a Gateway that is available on the internet.
- You should modify the test client to use your own specific client credentials.
- You should remove the test client from the OAuth Manager when it is no longer needed.

Enabling the Client

➢ *To enable the OAuth 2.0 test client:*

1. Navigate to the following URL in a browser:

   https://< *Gateway_host* >**:8443/oauth/manager**

2. Click [**Manage Clients**].
3. Click [**List Keys**] of the client with the name "**OAuth2Client**"
4. Click [**Edit**]
5. Edit [**Callback URL**] and replace *"<YOUR_SSG>"* with the protocol, hostname and port of your Gateway; for example: *https://acmecorp.com:8443*

   Note that there are two replacements to be performed.
6. Click [**Save**].
7. Click [**Manage Client → List Keys**] and verify that the client key has been updated.

Tip: Click [**Manage Clients**] to list, delete, and register client applications. Click [**Manage Tokens**] to list, and revoke access tokens granted to client applications.

Running the Client

➢ *To run the OAuth 2.0 test client:*

- Navigate to the following URL in a browser:

  https://<*Gateway_host*>**:8443/oauth/v2/client/authcode**

  The OAuth Client Test Application screen is displayed. Each control on the screen is described in the following sections.

Additional notes about the client apps:

- On the left sidebar of this page, you can navigate between the other OAuth 2.0 Test Clients:

  Authorization Code
  Implicit
  Client Credentials
  Resource Owner Password Credentials
  SAML

- On the left sidebar of this page, you can navigate to OAuth Manager:

  OAuth Manager (This link only works if the protocol and port is the same as used with the client application; for example: https/8443.)

  - Each client app maintains its own token. Each time you initiate a new OAuth session, the current access token is overwritten.

    The access token in memory is used to call the API of your choice. In the case of SAML, a SAML token is also maintained in memory and overwritten each time you initiate a new one.

Tip: Each OAuth 2.0 Test Client tests its own grant type. If you are only using a subset of the available OAuth grant types, you can ignore the other test clients.

Getting an Access Token

➢ *To get an access token before calling an API:*

1. In the OAuth Test Client, click [**Initiate**]. The OAuth 2.0 Authorization Server page is displayed.

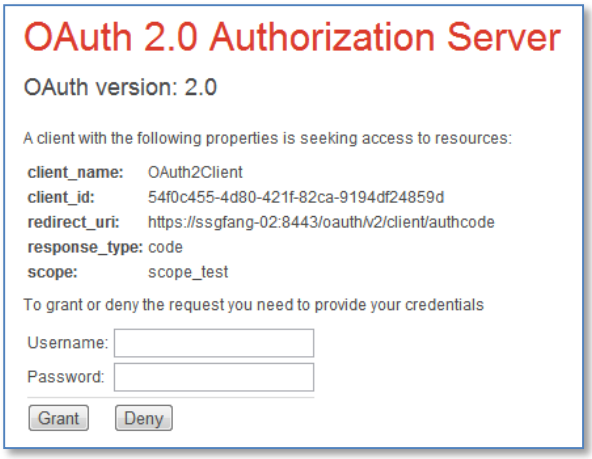

Figure 11: OAuth 2.0 Authorization Server

2. Enter your credentials and then click [**Grant**]. You will be redirected back to the client application with an access token and a refresh token.

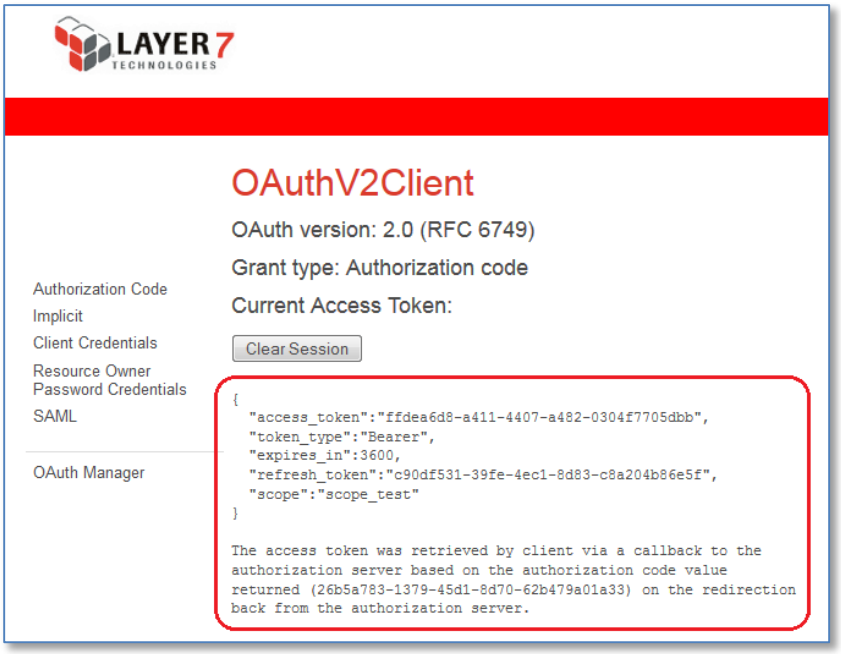

Figure 12: Current Access Token

Testing the Client

➢ *To test using the access token to call an API on the CA API Gateway:*

1. Enter a target URL in the **Target** field (see ❶).
2. Click [**Call API**] (see ❷). The client app will use the access token currently residing in memory as a credential to call the target API.

   The resulting response for this call will be displayed below the **Target** field (see ❸).

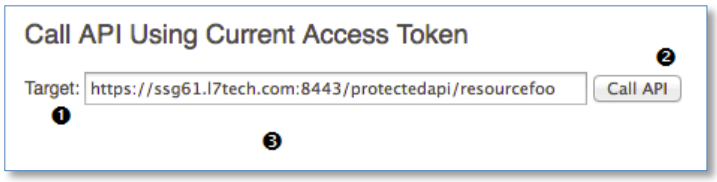

Figure 13: Using an access token to call an API

Refreshing a Token

➢ *To refresh an existing OAuth access token:*

Certain grant types support refresh tokens. This is indicated by the presence of a [**Refresh**] button.

- To refresh an existing OAuth access token, click the [**Refresh**] button.

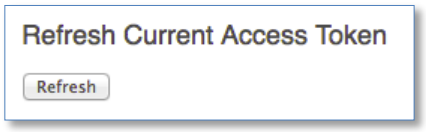

Figure 14: Refreshing an Access Token

Clearing the Current Session

➢ *To clear the current test client session:*

- Click the [**Clear Session**] button on the OAuth client page. This starts a new test and clears all the parameters in the clients.

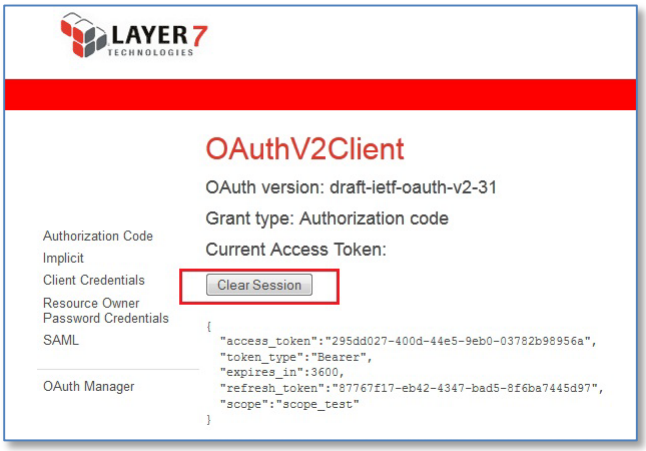

Figure 15: Clearing a session

Restricting the OAuth 2.0 Grant Types

By default, the OAuth Authorization Server enables the following OAuth grant types:

Authorization code
Implicit

Client Credentials
Resource owner password credentials
SAML

To restrict the different grant types that you want to support for your use cases, disable the branches that implement the ones you do not wish to support in the token endpoint policy. Set this to “At least one assertion must evaluate to true” with the comment “grant types” in endpoint.

/auth/oauth/v2/token

For more information on disabling a branch, see *Disabling an Assertion* in the *Layer 7 Policy Manager User Manual*.

Tip: In the Policy Manager, click [Show Comments] in the policy tool bar to see comments in the policy window.

Change Resource Owner Authentication

The authorization and token endpoint policies authenticate the resource owner. In the initial installation, the Internal Identity Provider is used to achieve this authentication. You may want to attach a different authentication source for your purpose.

For more information, refer to the following sections:

- *Authenticate User or Group Assertion* in the *Layer 7 Policy Authoring User Manual*
- *Working with Identity Providers* in the *Layer 7 Policy Manager User Manual*

Advanced Tip: If you change the way resource owner authentication is done by either the OAuth authorize or token endpoint policies, there may be downstream consequences. By default, these policies are set to authenticate against the Gateway's Internal Identity Provider using the Authenticate Against Identity Provider assertion. This assertion sets the *${request.authenticatedUser}* context variable at runtime and this variable is subsequently used by both the authorize and token endpoint policies to set another context variable representing the resource owner. If you change the resource owner authentication to another method that does not set the *${request.authenticatedUser}* context variable, then the OAuth authorize and token endpoint policies must be adjusted to set the resource owner context variables appropriately. (**Tip:** Use the Find command ([Ctrl]+F) in the policy window to search for “authenticatedUser” to find where these changes may need to be made.)

Adding SLA Rules

In addition to authenticating these various identities, you may want to add SLA rules associated with their use of the Authorization Server. Figure 16 shows an example of using the Apply Throughput Quota assertion to prevent a client from refreshing the access token more than 10 times per second.

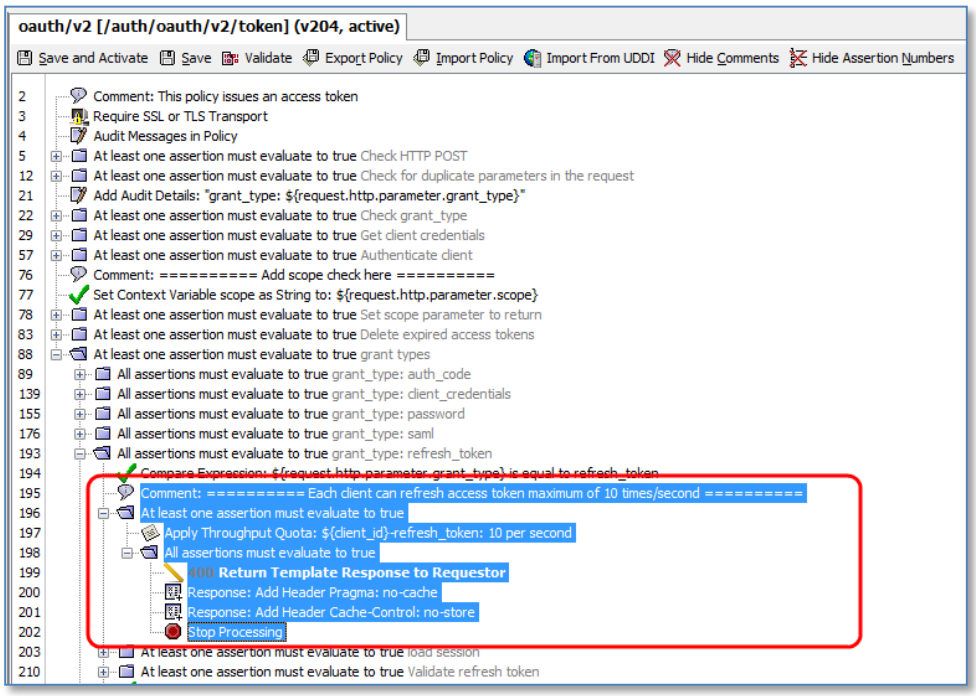

Figure 16: Adding SLA Rules sample

For more information, see “Apply Throughput Quota Assertion” in the *Layer 7 Policy Authoring User Manual*.

Chapter Four: Customizing the OAuth Toolkit

All OAuth Toolkit features are implemented using policies in the Policy Manager, making it possible to customize the entire OAuth process. This chapter describes some of the more common customizations.

Configuring a Corporate Brand

The OAuth toolkit comes with a simple policy-generated web page that is presented to end users as part of the Authorization process. This chapter describes how to replace this page with a template that conforms to a specific corporate style or theme. It is important that the pages presented by the Authorization server reflect your corporate style to help orient users by indicating the source of the page they are viewing.

The following steps describe how to create a themed template and configure the Gateway to use this template for the Authorization pages.

When the client is redirected to the authorization endpoint for the user login to grant/deny a request, the authorization server's website can be displayed.

- *To customize the authorization web page to configure a corporate brand:*

1. Create the HTML template. This can contain references to images and stylesheets. The easiest way to create this page is to take an existing HTML page and replace the actual content with the following tag:

   ```
   <!--oauth_content_placeholder-->
   ```

   This comment value will be replaced dynamically with a element containing all necessary values for the login/ grant/ deny dialogs.

```
<?xml version="1.0" encoding="UTF-8"?>¶
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"¶
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">¶
¶
<html>¶
    <head>¶
        <title>Pinnacle Shipping Web Services</title>¶
    </head>¶
    <body>¶
        <div class="outerBorder">¶
            <div class="header">¶
                <div class="banner">¶
                    <div class="banner_text">Auth Portal</div>¶
                </div>¶
            </div>¶
            <div id="main" class="main ">¶
                    <!--oauth_content_placeholder-->¶
            </div> ¶
        </div>¶
    </body>¶
```

Figure 17: Creating a branding template

When the authorize policy executes the downloaded web page, the above place holder will be searched and be replaced with a element containing the relevant markup. The element will include the login form, Grant and Deny buttons, as well as required error messages such as “Authentication Denied”.

Note: The HTML template can reference other resources provided that they are located in the same folder as the template. This means all images and style sheets must not be in subfolders.

---

2. Point the authorize policy to the web page by configuring the variables in policy fragment “OAuth 1.0 Context Variables” or “OAuth 2.0 Context Variables” (as appropriate to your installation).

   Set these variables to point to your website (for example: *http://myhost.com/website.html*):

   - **OAuth 1.0:** host_oauth_v1_server_website_baseuri
   - **OAuth 2.0:** oauth2_auth_template_path

   Set these variables to point to the host, including the protocol (for example, *http://myhost.com*):

   - **OAuth 1.0:** oauth_v1_server_website_baseuri
   - **OAuth 2.0:** host_oauth2_auth_server

Using the Customer's Identity Provider

For this initial installation, the internal identity provider built into the Gateway is used to authenticate users. You can change this to use the customer's identity provider (also known as *Identity Management System (IDMS)*). To do this, edit the policy for the authorization endpoint. The reference to the Internal Identity Provider can be changed to use the customer's identity provider. For example, Figure 18 illustrates changing the Internal Identity Provider to using an LDAP Identity Provider.

Advanced Tip: If you change the authentication to use the customer's identity provider rather than the Internal Identity Provider, consider the downstream consequences. Specifically, if a policy is modified to not use the Authenticate Against Identity Provider or Authenticate User or Group assertions, the value of the *${request.authenticatdUser}* context variable (which captures the name of the authenticated user) will need to be updated.

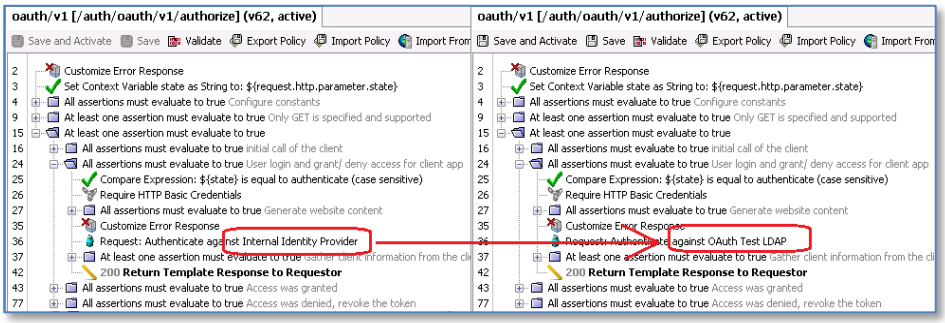

Figure 18: Changing the identity provider to the customer's identity provider

Note: The policy fragment shown Figure 18 is an example—your fragment and the line numbers may differ.

Configuring the Session Lifetime

To speed up the policy processing time, the OAuth Toolkit uses a combination of caching and session persistence to reduce the number of calls to the token store and client store endpoints. This functionality is provided by the "oauth/session" endpoint. The policy fragment "/oauth/validation/validate/v1/signature" uses the session endpoint to cache the *client_key* and *access_token* values.

The token properties are stored in the session, but the expiry is determined by the *expiration* property within the token. The cachAge controls how long the token will be in session and it also controls how long the client properties will be in session.

Customizing the Token Lifetime

It is possible to change the lifetime of tokens. The policy fragment "Token Lifetime Context Variables" contains the context variables that govern the lifetime of generated tokens. You can modify this policy as needed.

The following table describes the token lifetime variables.

Table 7: Token lifetime context variables for OAuth 1.0

Variable	Description
oauth_v1_access_token_lifetime_s	Controls the lifetime of access tokens. Set to "**0**" (zero) to make the token invalid immediately. Default: **86400** (seconds)
oauth_v1_request_token_lifetime_s	Controls the lifetime of request tokens. Set to "**0**" (zero) to make the token invalid immediately. Default: **300** (seconds)
oauth_v1_consumer_key_lifetime_m	Controls the lifetime of OAuth consumer keys. The default of "**0**" (zero) means an indefinite lifetime. Default: **0** (minutes)

Table 8: Token lifetime context variables for OAuth 2.0

Variable	Description
oauth2_auth_code_lifetime_sec	Controls the lifetime of issued OAuth codes. Set to "**0**" (zero) to make the code invalid immediately. Default: **600** (seconds)
oauth2_access_token_lifetime_sec	Controls the lifetime of issued access tokens. Set to "**0**" (zero) to make the token invalid immediately. Default: **3600** (seconds)
oauth2_refresh_token_lifetime_sec	Controls the lifetime of issued refresh tokens. Default: **604800** (seconds)

Adding OAuth Authorization Capabilities to Existing API

To control access to an API using OAuth, include either the "Require OAuth 1.0 Token" or "Require OAuth 2.0 Token" policy fragment. These fragments verify that the request contains a valid OAuth access token.

The following illustration shows how to protect a resource with the OAuth 2.0 runtime authorization fragment.

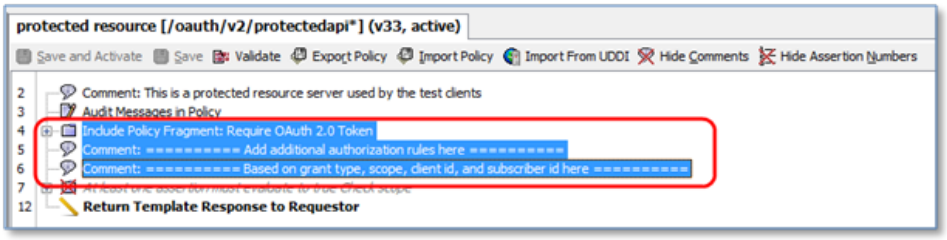

Figure 19: Example: Protecting a resource with the OAuth 2.0 Runtime Authorization Fragment

Additionally, the "Require OAuth 2.0 Token" policy fragment creates context variables that contain some of the attributes associated with the OAuth session for the token presented. This OAuth session information can be used from within the service policy to implement additional rules.

Table 9: Context variables set by the Require OAuth 2.0 Token policy

Context Variable	Description
${session.client_id}	The OAuth client ID to which the access token is issued.
${session.expires_at}	A timestamp (in long format) that represents a time after which the access token is no longer valid.
${session.scope}	The scope associated with the access token.
${session.subscriber_id}	The authenticated resource owner authenticated as part of the original OAuth handshake for this access token.

You can also add global OAuth Authorization rules directly within the "Require OAuth 1.0 Token" or "Require OAuth 2.0 Token" policy fragment. In this case, these rules apply the same way to all APIs that are protected with this OAuth fragment.

Appendix A: Related APIs for OAuth Toolkit

The OAuth Toolkit is using the APIs provided by the different components. The APIs can be used by any other third party client.

Three different APIs are provided and are required. All APIs support HTTP GET and HTTP POST (content-type: *application/x-www-form-urlencoded*). The need for SSL can be configured by simply modifying the policy using the Policy Manager. In this appendix, "HTTP" is used to keep the documentation consistent.

> **Tip:** Values within brackets followed by a "?" are optional parameters; for example: *(&parameter=value)?*

Clientstore API

This API allows CRUD operations using different endpoints and parameters to manage client applications and client keys (client_id, oauth_consumer_key).

All endpoints described will start here:

<Gateway_host_and_port>/ oauth/clientstore/*

For example:

http://my.securespan.gateway.com:8080/oauth/clientstore/store

API for registering a client application/client_key

The API will add the creation time.

1. */store?client_ident=<value>&name=<value>&org=<value>&registered_by=<value>&type=<value>(&description=<value>)?*
   - client_ident: the unique identifier of this client within the clientstore
   - name: the name given for this client
   - org: the organization that is registering this client
   - registered_by: the user who registers
   - type: for OAuth 2.0, select "confidential" or "public"
   - description: enter an optional description
2. */store?client_ident=<value>&client_key=<value>&secret=<value>&registered_by=<value>&status=<value>&scope=<value>&callback=<value>&environment=<value>&expiration=<value>*. This registers a new client_key.

- **client_ident:** the identifier of the associated client
- **client_key:** the registered client_key
- **secret:** the secret for this client_key
- **registered_by:** user who registers
- **status:** either ENABLED or DISABLED
- **scope:** used with OAuth 2.0
- **callback:** in OAuth 2.0 named as "redirect_uri". The URI a user client will be redirected to if it not contains "oob"
- **environment:** any value can be used to create client_key. This will be used in environments like "test", "integration" or "production"
- **expiration:** either "0" (forever) or a timestamp. The OAuth Toolkit uses milliseconds

3. */store?client_ident=<value>&name=<value>&org=<value>&registered_by=<value>&type=<value>&description=<value>&client_key=<value>&secret=<value>&status=<value>&scope=<value>&callback=<value>&environment=<value>&expiration=<value>*

   This registers a new client and a client_key. This is the default when using the OAuth Manager to register a new client.

The response will always be either one of the following.

- status: 200, content-type: text/plain, body: Persisted
- status: 400, content-type: text/plain, body: Client value could not be persisted

API for deleting a client

- */delete?client_ident=<value>*
  - **client_ident:** will delete the associated client and all client_keys owner by this client

The response will always be either one of these:

- status: 200, content-type: text/plain, body: {no-of-clients} client(s) deleted
- status: 400, content-type: text/plain, body: client could not be deleted

API for revoking a client_key

- */revoke?client_key=<value>* OR */revoke?client_ident=<value>*
  - If client_key: this client_key gets deleted
  - If client_ident: all client_keys for this client will be deleted

The response will always be either one of these:

- status: 200, content-type: text/plain, body: {no-of-clients} client_key(s) deleted
- status: 400, content-type: text/plain, body: client_key(s) could not be deleted

API for updating a client

1. */update?client_ident=<value>&name=<value>&type=<value>&description=<value>&org=<value>*
    - **client_ident:** the client identified by the client_ident will be updated
    - **name:** the new name (may be empty)
    - **type:** the new type (may be empty)
    - **description:** the new description (may be empty)
    - **org:** the new organization (may be empty)
2. */update?client_key=<value>&status=<value>&scope=<value>&callback=<value>&environment=<value>&secret=<value>*
    - **client_key:** update the values of the given client_key
    - **status:** the new status
    - **scope:** the new scope
    - **callback:** new callback (redirect_uri)
    - **environment:** set the new environment
    - **secret:** new secret

The response will always be one of these:

- status: 200, content-type: text/plain, body: {no-of-clients} client(s) updated
- status: 200, content-type: text/plain, body: {no-of-clients} client_key(s) updated
- status: 400, content-type: text/plain, body: client values could not be updated

API for requesting values of a given client

These APIs will use the given parameters as the search parameter in selecting only certain clients. All of these APIs accept the additional parameter “format”. “Format” can either take the value “xml” (which is the default) or “json”. If it is “json”, the values will be returned as a JSON message.

1. */get*
    - returns all values of all registered clients. No keys are included
2. */get?name=<value>&*
    - **name:** the name of the client as a filter
3. */get?name=<value>&org=<value>*

- **name:** the name of the client as a filter
- **org:** the organization of the client

4. */get?org=<value>*
   - **org:** the organization of the client
5. */get?registered_by=<value>*
   - **registered_by:** all clients registered by the given parameter
6. */get?registered_by=<value>&org=<value>*
   - **registered_by:** all clients registered by the given parameter
   - **org:** the organization
7. */get?client_key=<value>*
   - **client_key:** client values that "own" the given client_key
8. */get?client_ident=<value>&*
   - **client_ident:** the unique identifier identifying a certain client

The response will always be one of these:

- status: 200, content-type: text/xml, body:

```
<values xmlns=
"http://ns.l7tech.com/2012/11/otk-clientstore">
  <value index="value">
    <client_ident>value</client_ident>
    <name> value </name>
    <type> value </type>
    <description> value </description>
    <organization> value </organization>
    <registered_by> value </registered_by>
    <created> value </created>
  </values>
</values>
```

```
<values>
  <value>*
</values>
```

```
@index: represents the no.
of the value element in the
resulting list of value
elements. It is not a
constant value for this
value element
```

- status: 200, content-type: application/json, body:

```
{"values": {"value": [
    {
        "organization": "value",
        "index": value,
        "created": value,
        "description": "value",
        "name": "value",
        "type": "value",
        "client_ident": value,
        "registered_by": "value"
    }
]}}
```

```
{"values": ""}
```

- status: 400, content-type: text/plain, body: Non-valid parameter list for this operation

API for requesting values of a given client_key

These APIs will use the given parameters as a search parameter in selecting only certain client_keys. All of these APIs accept the additional parameter "format".

"Format" can either take the value "xml" (which is the default) or "json". If it is "json", the values will be returned as a json message.

1. */getKey*
   - returns all values of all client_keys
2. */getKey?client_key=<value>*
3. */getKey?name=<value>*
   - **name:** the name of the client used for this key
4. */getKey?name=<value>&org=<value>*
   - **name:** the name of the client used for this key
   - **org:** the name of the organization used for this key
5. */getKey?org=<value>*
   - **org:** the name of the organization used for this key
6. */getKey?client_ident=<value>*
   - **client_ident:** the identifier of the client issued for this key

The response will always be one of these:

- status: 200, content-type: text/xml, body:

```
<values xmlns=
"http://ns.l7tech.com/2012/11/otk-clientstore">
  <value index="value">
    <client_ident>value</client_ident>
    <client_name> value </client_name>
    <client_key> value </client_key>
    <secret> value </secret>
    <scope> value </scope>
    <callback> value </callback>
    <environment> value </environment>
    <expiration> value </expiration>
    <status> value </status>
    <created> value </created>
    <created_by0
  </value>
</values>
```

```
@index: represents the no.
of the value element in the
resulting list of value
elements. It is not a
constant value for this
value element
```

- status: 200, content-type: application/json, body:

```
{"values": {"value": [
    {
        "created_by": "value",
        "index": value,
        "environment": value,
        "scope": "value",
        "client_key": "value",
        "created": "value",
        "status": value,
        "expiration": "value",
        "callback": "value",
        "secret": "value",
        "client_ident": "value",
        "client_name": "value"
    }
]}}
```

```
{"values": ""}
```

- status: 400, content-type: text/plain, body: Non-valid parameter list for this operation

API for requesting values of a given client, client_key at once

These APIs will use the given parameters as search parameter to select certain clients and client_keys. All of these APIs accept the additional parameter “format”. “Format” can either take the value “xml” (which is the default) or “json”. If it is “json”, the values will be returned as a json message.

- */getAll*

“/getAll” can always be used with the same parameters as for “/get” and “getKey”. It will respond with the content of the selected client and client_key.

The response will always be one of these:

- status: 200, content-type: text/xml, body:

```
<values xmlns=
"http://ns.l7tech.com/2012/11/otk-clientstore">
  <clients>
    <values> … </values>
  </clients>
  <keys>
    <values> … </values>
  </keys>
</values>
```

“<clients>” contains the same “values” element as explained for client results

- status: 200, content-type: application/json, body:

```
{"values": {
    "keys": {"values": {"value": [...]}},
    "clients": {"values": {"value": [...]}}
}}
```

- status: 400, content-type: text/plain, body: Non-valid parameter list for this operation

Tokenstore API

This API allows CRUD operations using different endpoints and parameters to manage tokens.

All endpoints described will start here:

<Gateway_host_and_port>/ oauth/tokenstore/*

For example:

http://my.securespan.gateway.com:8080/oauth/tokenstore/store

API for registering a token. Additionally the API will add the creation time.

1. */store?token=<value>&expiration=<value>&client_key=<value>&client_name=<value>&callback=<value>&scope=<value>(&resource_owner=<value>)?(&secret=<value>)?.* Registers a temporary token.

- **token:** the token, either a *request_token* (OAuth 1.0) or an authorization_code (OAuth 2.0) to be stored
- **expiration:** the date this token expires (the OAuth Toolkit is using ms)
- **client_key:** the *client_key* issued for this token
- **client_name:** the client issued for this token
- **callback:** the callback uri. "redirect_uri" in OAuth 2.0
- **scope:** the scope used with OAuth 2.0
- **resource_owner:** the *resource_owner* who granted this token. This is optional because the initial OAuth 1.0 *request_token* was not granted when it was issued
- **secret:** the secret for this token. This is optional because OAuth 2.0 does not have one.

2. */store?token=<value>&secret=<value>&expiration=<value>&client_key=<value>&status=<value>&temp_token=<value>*. Registers access_tokens used with OAuth 1.0.
   - **token:** the *access_token*
   - **secret:** the shared secret for this token
   - **expiration:** the date this token expires (the OAuth Toolkit is using ms)
   - **client_key:** the *client_key* issued for this token
   - **status:** either ENABLED or DISABLED. DISABLED tokens will cause a request to fail
   - **temp_token:** the temporary token exchanged for this token
   - This API will add the *client_name* and *resource_owner* of the *temp_token*
3. */store?token=<value>&expiration=<value>&client_key=<value>&resource_owner=<value>&status=<value>&scope=<value>&client_name=<value>&secret=<value>(&rtoken=<value>&rexpiration=<value>)?.* Registers access_token used with OAuth 2.0. Optionally also a given refresh_token (rtoken) and refresh_token expiration (rexpiration).
   - **token:** the *access_token*
   - **expiration:** expiration: the date the token expires (the OAuth Toolkit is using ms)
   - **client_key:** the *client_key* issued for this token
   - **resource_owner:** the *resource_owner* who granted this token
   - **status:** either ENABLED or DISABLED. DISABLED tokens will cause a request to fail

- **scope:** the scope
- **client_name:** the client name issued for this token
- **secret:** the secret for this token. This is optional because it is only used for token type “MAC”
- **rtoken:** the refresh token is available
- **rexpiration:** the refresh token expiration date (if available)

The response will always be one of these:

- status: 200, content-type: text/plain, body: persisted
- status: 400, content-type: text/plain, body: Token could not be persisted

API for updating a token

Tip: You can also update the status of an access token by submitting the parameters “status” and “token”.

1. */update?token=<value>&resource_owner=<value>*
   - **token:** the token to be updated. This is a temporary token (*request_token*).
   - **resource_owner:** the *resource_owner* assigned to the token
2. */update?token=<value>&expiration=<value>&verifier=<value>*
   - **token:** token to be updated. This is a temporary token (*request_token*)
   - **expiration:** when the token expires (in milliseconds)
   - **verifier:** the verifier assigned to this token

The response will always be one of these:

- status: 200, content-type: text/plain, body: {no-of-tokens} token(s) updated
- status: 400, content-type: text/plain, body: Tokens could not be updated

API for revoking token

This API works for long-living tokens only.

1. */revoke?client_key=<value>&resource_owner=<value>*
   - **client_key:** revokes the token of this *client_key*
   - **resource_owner:** the resource owner who granted the token to be revoked
2. */revoke?client_key=<value>*
3. */revoke?resource_owner=<value>*
4. /revoke?token=<value>
   - **token:** the token to be revoked

The response will always be one of these:

- status: 200, content-type: text/plain, body: {no-of-tokens} token(s) revoked
- status: 400, content-type: text/plain, body: Token could not be revoked

API for deleting a token

- */delete?temp_token=<value>|token=<value>|rtoken=<value>|client_key=<value>.* Only one of these parameters at once.
  - **temp_token:** the temporary token to delete
  - **token:** the access_token to delete. This will include an assigned refresh_token
  - **rtoken:** the *refresh token* to delete. This will include an assigned access_token
  - **client_key:** all tokens issued for this *client_key*, includes temporary and long living tokens

The response will always be one of these:

- status: 200, content-type: text/plain, body: {no-of-tokens} token(s) deleted
- status: 400, content-type: text/plain, body: Token could not be deleted

API for retrieving token values

These APIs will use the given parameters as search parameter to select only certain tokens. All of these APIs accept the additional parameter "format". "Format" can either take the value "xml" (which is the default) or "json". If it is "json", the values will be returned as a json message.

1. */get*
   - No parameters, all values of all long-living tokens will be returned
2. */get?token=<value>|rtoken=<value>|resource_owner=<value>|client_key=<value>|status=<value>.* One parameter at once only.
   - **token:** all values of the given token (which is an *access_token*)
   - **rtoken:** all token values of the given *refresh_token*
   - **resource_owner:** all token values the given *resource_owner* has granted
     a. **client_key:** all token values of tokens issued for the given *client_key*
     b. **status:** all token values according to the given status (ENABLED |DISABLED)
3. */get?client_key=<value>&resource_owner=<value>*
   - **client_key:** all token values of tokens issued for the given *client_key* in combination with the given *resource_owner*

The response will always be one of these:

- status: 200, content-type: text/xml, body:

```
<values xmlns=
"http://ns.l7tech.com/2012/11/otk-tokenstore">
  <value index="value">
  <token> value</token>
  <secret> value </secret>
  <expiration>value </expiration>
  <rtoken> value </rtoken>
  <rexpiration> value </rexpiration>
  <scope> value </scope>
  <resource_owner>. value </resource_owner>
  <client_key> value </client_key>
  <client_name> value </client_name>
  <status> value </status>
  <created> value </created>
</value>
```

```
<values>
  <value>*
</values>
```

```
@index: represents the no.
of the value element in the
resulting list of value
elements. It is not a
constant value for this
value element
```

- status: 200, content-type: application/json, body:

```
{"values": {"value": [
    {
        "index": value,
        "client_key": "value",
        "scope": "value",
        "rtoken": "value",
        "created": value,
        "status": "value",
        "expiration": value,
        "token": "value",
        "resource_owner": "value",
        "secret": "value",
        "rexpiration": value,
        "client_name": "value"
    }
]}}
```

```
{"values": ""}
```

- status: 400, content-type: text/plain, body: Non-valid parameter list for this operation

API for retrieving temporary token values

These APIs will use the given parameters as search parameter to select only certain temporary tokens. All of these APIs accept the additional parameter “format”. “Format” can either take the value “xml” (which is the default) or “json”. If it is “json”, the values will be returned as a json message.

1. /getTemp
   - **no parameters:** Will return all token values of all temporary tokens
2. */getTemp?token=<value>|resource_owner=<value>|client_key=<value>.* Only one parameter at once.
   - **token:** all token values of this temporary token
   - **resource_owner:** all token values of temporary tokens granted by this *resource_owner*
   - **client_key:** all temporary token values issued for given *client_key*
3. *getTemp?token=<value>&verifier=<value>|client_key=<value>&resource_owner=<value>.* Only one combination of parameters at once.

The response will always be one of these:

- status: 200, content-type: text/xml, body:

```
<values xmlns=
"http://ns.l7tech.com/2012/11/otk-tokenstore">

  <value index="value">
    <token> value </token>
    <secret> value </secret>
    <expiration> value </expiration>
    <scope> value </scope>
    <resource_owner> value </resource_owner>
    <client_key> value </client_key>
    <client_name> value </client_name>
    <created> value </created>
    <callback> value </callback>
    <verifier> value </verifier>
  </value>
```

```
<values>
  <value>*
</values>
```

```
@index: represents the no.
of the value element in the
resulting list of value
elements. It is not a
constant value for this
value element
```

- status: 200, content-type: application/json, body:

```
{"values": {"value": [
    {
"index": value,
    "client_key": " value ",
    "scope": " value ",
    "created": value,
    "expiration": value,
    "token": " value ",
    "callback": " value ",
    "verifier": " value ",
    "resource_owner": " value ",
    "secret": " value ",
    "client_name": " value "
    }
]}}
```

```
{"values": ""}
```

- status: 400, content-type: text/plain, body: Non-valid parameter list for this operation

OAuth Validation Point (OVP) API

There are several endpoints that are used to validate requests using OAuth. Each endpoint handles different topics within the OAuth dance.

The first group of endpoints is used during the process of issuing tokens, while the second group is used when clients request protected resources.

All endpoints described will start here:

<Gateway_host_and_port>/oauth/validation/*

For example:

http://my.securespan.gateway.com:8080/oauth/validation/v1/authorize

If an endpoint contains “v1”, then it is used by OAuth 1.0. “v2” is used by OAuth 2.0

OVP API used during the token issuing process

1. */oauth/validation/v1/authorize?token=<value>&expiration=<value>&verifier=<value>*
   Used to authorize a *request_token* and make it available in exchange for an *access_token*

- **token:** the temporary token to be authorized
- **expiration:** the new expiration date that will be used if the token is valid
- **verifier:** the verifier used if the token is valid

The validation includes the following validation steps:

- the expiration date has not expired
- a *resource_owner* must be assigned to the token
- a callback must be assigned to the token
- the token store is updated. The token can be exchanged for an *assess_token*

The response will always be one of these:

- status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <callback>value</callback>
</validation>
```

- status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

2. */oauth/validation/validate/v2/granttype?callback=<value>&client_key=<value>&token=<value>*

   Used to validate grant types (only auth_code validated here)

   - **callback:** the *redirect_uri* passed in by the client
   - **client_key:** the associated *client_key*
   - **token:** the temporary token (*authorization_code*)

   The validation includes the validation steps:

   - *client_key* must match the one that was used when the token was generated
   - the expiration date must not be expired
   - a callback must either be empty or equal to the one used when the token was generated

   The response will always be one of these:

   - status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <scope>the scope used when the token was generated</scope>
  <resource_owner>the resource_owner who granted this
token</resource_owner>
</validation>
```

- status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

3. */oauth/validation/validate/v2/refreshtoken?client_key=<value>&rtoken=<value>&scope=<value>*

   Used to validate a *refresh_token*

   - **client_key:** the *client_key* issued for this token
   - **rtoken:** the *refresh_token* to be validated
   - **scope:** the scope to be used

   The validation includes the validation steps:

   - *client_key* must match the one that was used when the token was generated
   - the expiration date must not be expired
   - the status must be ENABLED

   The response will always be one of these:

   - status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <scope>the if scope is empty the scope used when the token was generated , if
not the given one</scope>
  <resource_owner>the resource_owner who granted this token</resource_owner>
</validation>
```

   - status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

4. */oauth/validation/validate/v2/tokenrequest?client_key=<value>&secret=<value>*

   Used to validate a the client credentials when requesting an access_token

   - **client_key:** the *client_key*
   - **secret:** secret for the *client_key*

   The validation includes the validation steps:

   - the secret
   - the expiration date must not be expired
   - the status must be ENABLED

The response will always be one of these:

- status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <type>the type fo the client</type>
  <client_name>the name of the client</client_name>
  <scope> the scope used when the token was generated </scope>
</validation>
```

- status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

OVP API used when clients access resources

1. /oauth/validation/validate/v2/token (oauth parameters)

   - Used to validate “oauth” parameters. The validation depends on the token type. If it is “MAC” or “BEARER”, the expiration date and the status will be verified

   The response will always be one of these:

   - status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <client_key> the associated client key </client_key>
  <resource_owner>the resource owner who granted the token</resource_owner>
  <scope>the scope used when the token was generated</scope>
  <expiration>the tokens expiration date</expiration>
</validation>
```

   - status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

Note: The OAuth Validation Point caches validated tokens to improve performance. The default *cacheAge* context variable for the token at */oauth/validation/validate/v2/token* in SecureZone OVP is 60 seconds. This means that a revoked token may continue to be authorized for up to 60 seconds beyond revocation. Be sure to review this default to ensure that it conforms to the security policy at your organization.

---

2. /oauth/validation/validate/v1/signature (oauth parameters)

   Used to validate “oauth_signature”

   - The signature will be verified
   - The *client_key* will be verified
     - expiration date has not expired

- status is ENABLED

- The token will be verified
  - Expiration date has not expired
  - status is ENABLED (for *access_tokens* only)
- For authorized request token, the verifier must exist in the tokenstore

The response will always be one of these:

- status: 200, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>valid</result>
  <client_name>value</client_name>
</validation>
```

status: 401, content-type: text/xml, body

```
<validation xmlns="http://ns.l7tech.com/2012/11/otk-ovp">
  <result>invalid</result>
</validation>
```

Note: The OAuth Validation Point caches validated signatures to improve performance. The default *cacheAge* context variable for the signature at */oauth/validation/validate/v1/signature* in SecureZone OVP is 30 seconds. This means that an accepted signature will be cached for 30 seconds. Be sure to review this default to ensure that it conforms to the security policy at your organization.
