   d. Click **Execute**. The Transaction Results shows returned values, and the API details field displays the detailed log of all blockchain processes performed from invoking the transaction.

9. Confirm whether the chaincode invoked successfully.
   a. Go to the Channels tab, and locate and click the channel the chaincode was installed on.
   b. Confirm that the Ledger pane is selected, and in the Query Ledger table, locate the block number indicating that an invocation occurred.
   c. Click the block and confirm that in the Transactions table you see “Success” in the **Status** column.
10. If needed, go to the Samples page and invoke any other operations on the chaincode.

Start and Stop Nodes

You can start or stop CA, orderer, peer, and the REST proxy nodes in your network. You can't start or stop the console node or remote peer nodes.

You can start and stop nodes depending upon the traffic in your network. For example, if network traffic is light, then you can stop unneeded peer nodes and orderer nodes.

You can also restart a node. See Restart a Node.

When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Channel tab and Chaincodes tab. If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. If you stop all peers joined to a channel, then the Channels tab lists the channel, but its information isn't available to view.
Before stopping a node for an extended period of time, you should transfer all this peer's responsibilities to other running peers, and then remove all the responsibilities this peer has.

- Check all other peers' gossip bootstrap address lists, remove the peer address, and add another running peer's address if needed. After peer configuration change, restart the peer.
- Check all channels' anchor peer lists, remove the peer from the anchor peer lists, and add another running peer to the anchor peer list if needed.
- If a channel or chaincode is only joined or instantiated in this peer, you should consider using another running peer to join the same channel and instantiate the same chaincode.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to start or stop, and click the node's **More Actions** button.
3. Click either the **Start** or **Stop** option. The node's status changes to either *up* or *down* and information is written to the node's log file.

Restart a Node

You can restart the CA, orderer, peer, and REST proxy nodes in your network. You can't restart the console node or remote peer nodes.

You should restart a node if it's not responding or running properly, or if you've updated a node's configuration. You can also start or stop a node. See Start and Stop Nodes.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to restart, and click the node's **More Actions** button.
3. Click **Restart**.

   The node's status changes to *restarting* and information is written to the log file.

Set the Log Level for a Node

If you’re an administrator, then you can specify the type of information you want to include in a node’s log files. For example, ERROR, WARNING, INFO, or DEBUG.

By default, every node’s log level is set to INFO. When developing and testing your network, Oracle suggests that you set the logging level to DEBUG. If you're working in a production environment, then use ERROR.

Only an administrator can change a node’s log level setting. If you're a user, then you can view a node’s log level settings.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the nodes table, locate the node you want to update, click its **More Actions** menu, and click **Edit Configuration**.

   If you have user permissions, then your console will have the **View** option that you click to see the node’s log level setting and other configuration settings.

   The Configure dialog is displayed.
3. In the **Log Level** field, select the log level you want to use.
4. Click **Submit**.

   If you changed the logging level on a peer node, then you need to restart the peer node. For all other node types, the logging level change immediately, and you don’t have to restart the console.

Manage Channels

This topic contains information about managing the channels in your network.

Topics

- What Are Channels?
- View Channels
- Create a Channel
- View a Channel’s Ledger Activity
- View or Update a Channel’s Organizations List
- Join a Peer to a Channel
- Add an Anchor Peer
- Change or Remove an Anchor Peer
- View Information About Instantiated Chaincodes
- Work With Channel Policies and ACLs

What Are Channels?

Channels partition and isolate peers and ledger data to provide private and confidential transactions on the blockchain network.

Members define and structure channels to allow specific peers to conduct private and confidential transactions that other members on the same blockchain network can't see or access. Each channel includes:

- Peers
- Shared ledger
- Chaincodes instantiated on the channel
- One or more ordering service nodes
- Channel policy definitions and ACLs where the definitions are applied

Each peer that joins a channel has its own identity that authenticates it to the channel peers and services. Although peers can belong to multiple channels, the information on transactions, ledger state, and channel membership is restricted to peers within each channel.

You can use the Oracle Blockchain Platform console or the Hyperledger Fabric SDK to create channels on your blockchain network. See View Channels.

View Channels

Members in your network use channels to privately communicate blockchain transactions information.

Use the Channel tab to view a list of the channels in your network, create and monitor channels, specify anchor peers, and upgrade the instantiated chaincodes used on your channels.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.

2. In the channel table, click the channel name you want information about. Note that if all peers joined to the channel are stopped, then the channel is listed but its information isn't available to view.

   The Channel Information page is displayed.

3. Click through the Channel Information page's panes to find information about the channel.

Section	What can I do in this pane?
Ledger	Get information about the channel’s ledger activity such as block number and the number of user transactions in the block. Click a block number to drill into information about its transactions. You can use the filter field to specify the summary information that you want to see (for example, information from the last day or last month), or use the custom option to enter start and end times. See View a Channel’s Ledger Activity.
Instantiated Chaincodes	View the list of chaincodes that have been instantiated on the channel.
Peers	View the list of peers that are joined to the channel. Use this section to set anchor peers for the channel.
Organizations	View the list of network members whose peers are using the channel to communicate.

Section	What can I do in this pane?
Channel Policies	View the list of the standard policies and any policies that you created for the channel. Use this section to add, modify, and delete policies.
ACLs	View the access control lists (ACLs) and the policies used to manage which organizations and roles can access the channel's resources.

Create a Channel

You can add channels to the network and specify which members can use the channel, and which peers can join the channel. You can’t delete channels.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
2. In the Channels tab, click **Create a New Channel**.
3. In the **Channel Name** field, enter a unique name for the channel. The channel's name can be up to 128 characters long.
4. In the **Application Capabilities** field, select 1_3 as the capabilities level for the channel. Don't select 1_1 - it was used in earlier versions of the product and selecting it will remove support for newer product features.
5. In the Organizations section, select any additional members that you want to communicate on the channel.

   If you’re working in a participant instance, you need to add the founder to your instance before the founder’s MSP ID displays in the Organization section. To add the founder organization, go to the Network tab and click the **Add Organization** button to upload the founder’s certificates.
6. In the **MSP ID ACL** section, specify the organizations that have access to the channel and permissions for each selected organization. Note that you can add more organizations to or delete them from the channel later, as needed.

   Your organization’s permissions are set to write (ReaderWriter) and you can't modify this setting. By default, other member’s permissions are set to write (ReaderWriter), but you can change them to read (ReaderOnly) if you don’t want the members to invoke chaincodes and to only read channel information and blocks on the channel.
7. (Optional) In the **Peers to Join Channel** field, select one or more peers. Note the following information:
   - Your instance has two VMs (Partition 1 and Partition 2) and Oracle recommends that you join one peer from each partition to the channel. This is because if one VM is unavailable that the channel can still process endorsements and commits. A peer’s name tells you which partition it's located in. For example, peer0–1 and peer1–1 are located in Partition 1. And peer0–2 and peer1–2 are located in Partition 2.
   - You can join a maximum of seven peers from Partition 1 and seven peers from Partition 2.
   - If your network contains participants, the participants’ peers don’t display in this list. Participants must use their consoles to join peers to the channel. A

participant can’t join its peers to the channel unless its organization was added to the channel’s MSP ID ACL section.

- If you want to create the channel only, then don’t select any peers. You can add peers to the channel later.

8. Click **Submit**.

   The channel table displays the new channel.

After you create the channel, you can:

- Instantiate a chaincode on the channel. See Instantiate a Chaincode.
- If the network contains participants, then they use their consoles to join member peers to the channel. See Join a Peer to a Channel.

View a Channel’s Ledger Activity

Use the ledger to find summary information and runtime statistics for transactions on a specific channel.

1. Go to the console and select the Channels tab.
2. In the channel table, click the channel name that you want transaction information about. In the Channel Information page, confirm that the Ledger pane is selected.
3. Use the Ledger Summary area to find at a glance information about the channel’s activity, such as the total number of blocks in the ledger’s chain and the total number of user transactions on the channel.
4. To see blockchain activity that occurred at a specific time such as for the last day or week, go to the filter dropdown menu to select the time range that you want. To locate and drill into a specific set of transactions, select **Custom** and enter search criteria in the **Start Time** and **End Time** fields, or click the calendar icon and pick the dates that you want. Click **Apply**.

   If you select a specific time period (for example, **Last day**) and then select it again to re-run the query, the query doesn’t re-run. To get the latest information, click the **Refresh** button.

   Note the following transaction types that can display for a block:

   - genesis — The transaction that runs the configuration block to initialize the channel.
   - data (sys) — The transaction that starts the chaincode’s container to make the chaincode available for use.
   - data — A chaincode transaction called for execution on the channel.

5. To find more information about a specific transaction, locate the transaction in the query ledger table and click it. The transactions table displays the transaction’s details.

Transaction Detail	Description
TxID	The unique alphanumeric ID assigned to the transaction. The TxID is constructed as a hash of a nonce concatenated with the signing identity's serialized bytes.
Time	The transaction’s time stamp (date and time that the transaction occurred).

Transaction Detail	Description
Chaincode	Displays the name of the chaincode that executed the transaction. This field can show the name of a chaincode that you wrote, installed, and instantiated, but can also show a system chaincode. System chaincode options are: • LSCC — For lifecycle requests, such as instantiate, install, and upgrade. • QSCC — For querying. This chaincode includes APIs for ledger query.
Status	Shows if the transaction succeeded or failed.

6. Click the triangle next to the TxID to view in depth information about the transaction, such as function name, arguments, validation results, response status, the initiator and the endorser.

   Note that if a transaction failed, then you can use the TxID to search error logs in the peer node or orderer nodes for more information.

View or Update a Channel’s Organizations List

You can view the list of the organizations that have access to the channel. If you created the channel, then you can change an organization’s permissions on the channel, and you can add organizations to or remove them from the channel

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels in your network.

2. In the channels table, locate the channel that you want information about, click the channels **More Actions** button, and click **Edit Channel Organizations**.

   The Edit Organizations page is displayed.

3. In the **MSP ID ACL** section, you can do the following:

   - Modify an organization’s permissions. The organization that created the channel is set to write (ReaderWriter). You can't change this setting.
   - If you’re the network founder, then clear an organization’s checkbox to delete it from the channel. If you’re a network participant, then use the **Delete** button to delete an organization from the channel. If you delete an organization from a channel, then the organization and its peers can no longer query, invoke, and instantiate a chaincode on the channel. And the removed organization’s peers can’t join the channel.
   - Click an organization’s checkbox to add the organization to the channel and set its permissions. By default, each member’s permissions is set to write (ReaderWriter), but you can change it to read (ReaderOnly) if you don’t want the member to invoke chaincodes and to only read channel information and blocks on the channel.

4. Click **Submit** to save the changes.

Join a Peer to a Channel

You can add a peer node to a channel so that the node can use it to exchange private transaction information with other peer nodes on the channel.

Note the following information:

- When you create a channel, you specify which local peer nodes can join the channel.
- If you're creating a network containing a participant, then you can select the participant as a member on the channel. Or you can add the participant after the channel is created.
- Your instance has two VMs (Partition 1 and Partition 2) and Oracle recommends that you join one peer from each partition to the channel. This is because if one VM is unavailable that the channel is still available for endorsements and commits. A peer's name tells you which partition it's located in. For example, peer0–1 and peer1–1 are located in Partition 1. And peer0–2 and peer1–2 are located in Partition 2.
- You can join a maximum of seven peers from Partition 1 and seven peers from Partition 2.

See Create a Channel.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the peer node that you want to add to a channel.
3. In the Node Information page, click the Channels pane to view the list of channels the peer is already using.
4. Click **Join New Channels**.

   The Join New Channels dialog is displayed.
5. Click the **Channel Name** field and from the list select the name of the channel to join. Click the field again to select another channel. Click **Join**.

Add an Anchor Peer

Each member using a channel must designate at least one anchor peer. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

You can designate one or more peers in your organization as an anchor peer on a channel. For a high availability network, you can specify two or more anchor peers. All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name you want to add anchor peers to.

The Channel Information page is displayed.

3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to designate as anchor peers and click their **Anchor Peer** checkboxes to select them.
5. Click the **Apply** button.

Change or Remove an Anchor Peer

(New in 19.2.1) You can change or remove a channel's anchor peers. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

Before you change or remove the channel's anchor peers, note the following information:

- To communicate on the channel, you must designate one or more peers in your organization as an anchor peer.
- For a high availability network, you can specify two or more anchor peers.
- All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name you want to remove anchor peers from.

   The Channel Information page is displayed.
3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to remove as anchor peers and clear their **Anchor Peer** checkboxes. Alternatively, to add another peer as an anchor peer, click its **Anchor Peer** checkbox to select it.
5. Click the **Apply** button.

View Information About Instantiated Chaincodes

You can view information about the chaincodes instantiated on the different channels in your network.

Some examples of when you need information about instantiated chaincodes are to determine if you need to upgrade the chaincode, or to find out which channels the chaincode was instantiated on.

1. Go to the console and select the Channels tab.
2. In the channels table, click the channel name with the chaincode that you want to view information for.
3. In the Channel Information page, confirm that the Instantiated Chaincodes pane is selected

4. In the chaincode table, you can:
   - Click the chaincode to go to the Chaincodes tab to learn more information about it, for example the peers that the chaincode is installed on and the channels that the chaincode is instantiate on.
   - In a chaincode’s More Actions menu, click **View Endorsement Policy** to find details about the chaincode’s endorsement policy, for example who must endorse the chaincode and the signed by expression string.
5. (Optional) If you see a channel listing without a chaincode, then you can go to the Chaincodes tab and instantiate a chaincode to the channel. See Instantiate a Chaincode.

Work With Channel Policies and ACLs

(19.1.3 and later versions only) This topic contains information about a channel's policies and ACLs.

Topics:

- What Are Channel Policies?
- Add or Modify a Channel's Policies
- Delete a Channel's Policies
- What Are Channel ACLs?
- Update Channel ACLs

What Are Channel Policies?

(19.1.3 and later versions only) A policy defines a set of conditions. The required parties must meet the policy's conditions before their signatures are considered valid and the corresponding request happens on the network.

The blockchain network is managed by these policies. Policies check the identity associated with a request against the policy associated with the resource needed to fulfill the request. Policies are located in the channel's configuration.

After you configure the channel's policies, you assign them to the channel's ACLs resources to determine which members are required to sign before a change or action can happen on the channel. For example, suppose you modified the Writers policy to include members from Organization A or Organization B. Then you assigned the Writers policy to the channel's cscc/GetConfigBlock ACL resource. Now only a member from Organization A or Organization B can call GetConfigBlock on the cscc component.

What Are the Policy Types?

There are two policy types: Signature and ImplicitMeta.

- **Signature** — Specifies a combination of evaluation rules. It supports combinations of *AND*, *OR*, and *NOutOf*. For example, you could define something like “An admin of org A and 2 other admins" or "11 of 20 org admins.”
  Note that when you modify the Oracle Blockchain Platform's default Admins policy, which was created as an ImplicitMeta policy, you'll use the Signature policy. Any new policies you create will be Signature policies.

- **ImplicitMeta** — This policy type is only valid in the context of configuration. It aggregates the result of evaluating policies deeper in the configuration hierarchy, which are defined by Signature policies. It supports default rules, for example "A majority of the organization admin policies."
  Oracle Blockchain Platform uses the ImplicitMeta policy type to create the Admins policy. When you modify the Admins policy, you'll use the Signature policy. You can't create or modify any policies using the ImplicitMeta policy. Oracle Blockchain Platform only supports modifying or creating policies using the Signature policy type.

When Are Policies Created?

When you add a channel to the network, Oracle Blockchain Platform created new default policies. The default policies are: Admins (ImplicitMeta policy), Creator, Writers, and Readers (Signature policies). If needed, you can modify these policies or create new policies.

Note the following important issues about channel policies:

- You can use the console to create a channel and set your organization's ACL to ReaderOnly. After you save the new channel, you can't update this ACL setting from the channel's Edit Organization option.

  However, you can use the console's Manage Channel Policies functionality to add your organization to the Writers policy, which overwrites the channel's ReaderOnly ACL setting.
- When you use the Hyperledger Fabric SDKs to create a channel, Fabric uses the ImplicitMeta policies as the default channel policies for Readers and Writers. When the channel uses these policies, the Oracle Blockchain Platform console can't guarantee that the administrative operations (for example, edit organization) will be successfully processed.

  To correct this issue, update the readers and writers policies to Signature policies, and define the policy rules as needed. See https://hyperledger-fabric.readthedocs.io/en/release-1.3/access_control.html
- When you use the Hyperledger Fabric SDKs or CLI to create a channel, the Creator policy isn't included in the configtx.yaml file. The Creator policy is required by Oracle Blockchain Platform to allow the channel creator to edit a channel's configuration. You must manually edit the configtx.yaml file and add the Creator policy.

Add or Modify a Channel's Policies

(19.1.3 and later versions only) You can add or modify a channel's policy to specify which members are required to perform a specific action on the channel. After you define policies, you assign them to the channel's ACLs.

Before you add or update policies, you need to understand how Oracle Blockchain Platform creates default channel policies. See What Are Channel Policies?

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name that you want to add policies to or modify policies for.

The Channel Information page is displayed.

3. In the Channel Information page, click the Channel Policies pane.
4. Do one of the following:
   - To add a new policy, click the **Create a New Policy** button. The **Create Policy** dialog is displayed. Enter a name in the **Policy Name** field and select Signature in the **Policy Type** field. Expand the **Signature Policy** section.
   - To modify an existing policy, click a policy's name. The **Update Policy** dialog is displayed.
5. Click the **Add Identity** button to add an organization. Or modify an existing signature policy as needed. Note the following information:

Field	Description
MSP ID	From the dropdown menu, select the organization that must sign the policy.
Role	Select the corresponding peer role required by the policy. Usually this will be member. You can find a peer's role by viewing its configuration information.
Policy Expression Mode	In most cases, you'll use **Basic**. Select **Advanced** to write an expression string using *AND*, *OR*, and *NOutOf*. See the Hyperledger Fabric documentation for information about how to write a valid policy expression string.
Signed By	Select how many members must sign the policy to fulfill the request.

6. If you're adding a new policy, then click **Create**. If you're modifying a policy, then click **Update**.

Delete a Channel's Policies

(19.1.3 and later versions only) You can delete a policy from a channel.

You can't delete a channel policy if it is assigned to an ACL. Before you try to delete a channel policy, confirm that the policy isn't assigned.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel that you want to delete a policy from.
   The Channel Information page is displayed.
3. In the Channel Information page, click the Channel Policies pane.
4. Locate the policy that you want to delete and click its **More Options** button.
5. Click **Remove** and confirm the deletion.

What Are Channel ACLs?

(19.1.3 and later versions only) Access control lists (ACLs) use policies to manage which organizations and roles can access a channel's resources.

Users interact with the blockchain network by targeting components such as the query system chaincode (qscc), lifecycle system chaincode (lscc), configuration system chaincode (cscc), peer, and event. These components are associated with specific resources (for example, GetConfigBlock or GetChaincodeData) that you can assign policies to at the channel level. These policies are a part of the channel's configuration.

A policy defines which organizations and roles can request a resource. When a request is made, the policy tells the system to check the requester's identity and determine if it's authorized to make the request. When you create a channel, Oracle Blockchain Platform includes the default Hyperledger Fabric ACLs with the channel. Oracle Blockchain Platform also creates four default policies (Admin, Creator, Writers, and Readers) for the channel. You can modify these policies or create new policies as needed. See What Are Channel Policies?

Update Channel ACLs

(19.1.3 and later versions only) You can update the channel's ACLs by assigning policies to the channel's resources. A policy defines which organizations and roles can request a resource

Before you update a channel's ACLs, you should understand what policies and ACLs are. See What Are Channel Policies? and What Are Channel ACLs?

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the name of the channel that you want to update ACLs for.
   The Channel Information page is displayed.
3. In the Channel Information page, click the ACLs pane.
4. In the Resources table, locate the resource that you want to update. Click the resource's **Expand** button and select the policy that you want to assign to the resource.
5. Modify the other resource's policies as needed.
6. Click **Update ACLs**.

Manage Certificates

This topic contains information about how to manage your network's certificates.

Topics:

- Typical Workflows to Manage Certificates
- Export Certificates
- Import Certificates to Add Organizations to the Network

- What's a Certificate Revocation List?
- View and Manage Certificates
- Revoke Certificates
- Apply the CRL

Typical Workflows to Manage Certificates

Here are the common tasks for managing your network's certificates.

Adding Organizations to the Network

You must be an administrator to perform these tasks.

Task	Description	More Information
Export or prepare an organization's certificates	The organization that wants to join the network either outputs or writes its certificates file and gives it to the founder.	Export Certificates Create a Fabric Organization's Certificates File Create an Organization's Third-Party Certificates File
Import member certificates	The founder imports the organization's certificates file to add the organization to the network.	Import Certificates to Add Organizations to the Network
View certificates	The founder can view and manage the network's certificates.	View and Manage Certificates

Revoking Certificates

You must be an administrator to perform these tasks.

Task	Description	More Information
Decide which certificates to revoke	View the certificates on your system to determine which ones to revoke to keep the network secure.	View and Manage Certificates
Select the certificates to revoke	Revoke the certificates in your CA.	Revoke Certificates
Apply CRL	Generates and applies an updated CRL to ensure that clients with revoked certificates can't access channels.	Apply the CRL

Export Certificates

Founders and participant organizations must import and export certificate JSON files to create the network.

Note the following information:

- For the founder to add a participant organization to the blockchain network, the participant must export its certificates file and make it available to the founder. The founder then uploads the certificates file to add the participant organization to the network.
- The certificate export file contains admincerts, cacerts, and tlscacerts.
- You might need to export certificates for blockchain or application developers. For example, a client application needs the TLS certificate to interact with peers or orderers.

For information about writing certificate files required to add Hyperledger Fabric or Third-Party organizations to the network, see Extend the Network.

1. Go to the console and select the Network tab.
2. In the Network tab, go to the Organizations table, locate the member that you want to export certificates for, and click its **More Actions** button.
3. Click **Export Certificates**.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. Specify where to save the file. Click **OK** to save the certificates file.
5. Send the certificates JSON file to the founder for import. See Import Certificates to Add Organizations to the Network.

Import Certificates to Add Organizations to the Network

To add an organization to the network, the founder must import a certificates file that was exported or prepared by the organization that wants to join the network.

You can import certificates for the following organization types.

Type	Description
Oracle Blockchain Platform Participant Organization	You can import a participant organization into a Oracle Blockchain Platform network. You upload the certificates that the participant organization exported from the console and sent to you. For information about creating certificates for upload and a list of the other steps that you need to perform to successfully set up a participant organization on the network, see Export Certificates.
Hyperledger Fabric Organization	You can import a Hyperledger Fabric organization into an Oracle Blockchain Platform network. To successfully upload a Fabric organization’s certificates file, you must modify the certificates file to replace all instances of \n with the newline character. See Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network.

Type	Description
Third-Party Certificate Organization	You can import an organization that is using certificates generated from a third-party CA server. To successfully upload a third-party organization’s certificates file, you must modify the certificates file to replace all instances of \n with the newline character. See Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network.

You must be an administrator to import certificates.

1. Go to the console and select the Network tab.
2. In the Network tab, click **Add Organizations**. The Add Organizations page is displayed.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
3. Click **Upload Organization Certificates**. The File Upload dialog is displayed.
4. Browse for and select the JSON file containing the certificate information for the organization you want to add to the network. Usually this file is named `certs.json`. Click **Open**.
5. (Optional) Click the plus (+) icon to locate and upload another organization’s certificate information.
6. Click **Add**. The organizations that you added are displayed in the Organization table.

   Note the following information for Oracle Blockchain Platform participant, Hyperledger Fabric, and third-party certificate organizations. Even though the founder uploaded the certificate information, the added organization can’t use the ordering service to communicate on the network until it imports the founder’s ordering service settings. The founder must export its ordering service settings and give the resulting file to the joining organizations for import. See one of the following:

   - For Oracle Blockchain Platform participants, see Export Ordering Service Settings.
   - For Hyperledger Fabric organizations, see Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network.
   - For third-party certificate organizations, see Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network.

What's a Certificate Revocation List?

You use a certificate revocation list (CRL) to help manage the certificates throughout your network.

A CRL is a list of digital certificates that the issuing Certificate Authority (CA) has revoked before their scheduled expiration date and should no longer be trusted and used on the network. For example, you should revoke any certificates that have been lost, stolen, or compromised.

After you use the Manage Certificates functionality to revoke certificates for users, Oracle Blockchain Platform creates the CRL. To ensure that the certificates are revoked throughout the network, you'll need to:

- Use the Apply CRL functionality after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

View and Manage Certificates

Use the console to view and manage the user certificates in your instance and any of the certificates you imported when building the network.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization's ID and click its **More Actions** button. Select **Manage Client Certificates**.

   Note that the Certificate Summary table will be empty until you add users to your instance. Also, the administrator's certificate doesn't display in this table. This is to prevent you from accidentally revoking the administrator's certificate.

   Organizations with third-party certificates or Hyperledger Fabric organization with revoked certificates won't display in this table. In such cases, you must use the native Hyperledger Fabric CLI or SDK to import the organization's certificate revocation list (CRL) file.

   The Certificates Summary dialog is displayed and shows a list of the certificates in your instance.
3. As needed, perform any of the following tasks:
   - Revoke certificates. See Revoke Certificates.
   - If you've revoked certificates and are working in a network with multiple members, then use Apply CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

Revoke Certificates

An organization can revoke certificates for any of its users. To ensure that the network remains secure, you should revoke certificates in case they're lost, stolen, or compromised.

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization's ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.
3. In the Certificates Summary dialog, locate and select the IDs of the users that you want to revoke certificates for.

4. Click Revoke and confirm that you want to permanently revoke certificates for the selected users.

   The users with revoked certificate display in the table and are added to the CRL.

5. If you’re working in a network with other members, then to ensure that their revoked certificates are cleaned up across the network, you must do the following:

   - If you’re working in a network with multiple members, then apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

Apply the CRL

If you're working in a network, then you must apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents members with revoked certificates from accessing the channel.

You must do the following tasks before applying the CRL:

- Revoke certificates. See Revoke Certificates

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization’s ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.

3. Click the Apply CRL button and confirm that you want to apply the CRL.

Manage Ordering Service Settings

This topic contains information about how founders and participants manage ordering service settings.

Topics:

- Export Ordering Service Settings
- Import Ordering Service Settings
- Edit Ordering Service Settings
- View Ordering Service Settings

Export Ordering Service Settings

If you’re a founder and are adding a participant to the network, then you must download your instance’s ordering service settings and give them to the participant for import. The participant isn’t able to communicate on the network until it has uploaded the founder’s ordering service settings.

For more information about the process you need to follow to create a network with multiple members, see Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

It isn’t common, but in some situations, the founder might expose a different ordering service to one or more network participants. In this case, the founder will export the updated ordering settings and the impacted participants must import the revised settings.

If the founder changes the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.

1. Go to the founder’s console and select the Network tab.
2. Go to the Organization table, locate the founder’s ID, click **More Actions**, and select **Export Ordering Settings**.
3. Specify the location where you want to save the JSON file that contains the ordering settings information. Click **OK**.
4. Send the ordering settings export file to the participants that need to import it into their instances.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.

Import Ordering Service Settings

If you’re a participant joining a network, then you must import the network founder’s ordering service settings file. You’ll not be able to communicate on the network until this settings file is uploaded into your instance.

For more information about the process you need to follow to create a network with multiple members, see Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

It isn’t common, but in some situations, the founder might expose a different ordering service to you and other participants. In this case, the founder will export the updated ordering settings and you’ll upload the revised settings into your instance. See Export Ordering Service Settings.

If the founder changes the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.

1. Go to the participant’s console and select the Network tab.
2. Click **Ordering Service Settings** and click **Import**.
3. In the Ordering Settings dialog, click **Upload Ordering Settings** and browse for and select the JSON file containing the ordering settings information. Usually this file is named `<founderinstancename>-orderer-settings.json`.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. In the Ordering settings dialog, click **Submit** to upload the ordering settings to the participant.

Edit Ordering Service Settings

You can update the ordering service settings for the founder instance.

Note the following important information about editing the ordering service settings:

- The updated settings are used when you create new channels and are not applied to existing channels.
- Separately you can update the ordering service settings for an individual existing channels. Go to the Channels tab, locate the channel, click the **More Actions** menu, and select **Update Ordering Service Settings**.
- If you change the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.
- It isn't common, but in some situations, you might expose a different ordering service to some of the network participants. In this case, you'll export the updated ordering settings and the required participants will import the revised settings. See Export Ordering Service Settings.

You must be an administrator to perform this task.

1. Go to the founder's console and select the Network tab.
2. Click the **Ordering Service Settings** button.

   The Ordering Service Settings dialog is displayed.
3. Update the settings as needed.

Field	Description
Batch Timout (ms)	Specify the amount of time in milliseconds that the system should wait before creating a batch. Enter a number between 1 and 3600000.
Max Message Count	Specify the maximum number of message to include in a batch. Enter a number between 1 and 4294967295.
Absolute Message Bytes	Specify the maximum number of bytes allowed for the serialized messages in a batch. This number must be larger than the value you enter in the Preferred Message Bytes field.
Preferred Message Bytes	Specify the preferred number of bytes allowed for the serialized messages in a batch. A message larger than this size results in a larger batch, but the batch size will be equal to or less than the number of bytes you specified in the Absolute Message Bytes field. Oracle recommends that you set this value to 1 MB or less. The value that you enter in this field must be smaller than the value you enter in the Absolute Message Bytes field.

4. Click **Update**.

   The updated settings are saved.

View Ordering Service Settings

You can view the ordering service settings that were imported into a participant's Oracle Blockchain Platform instance.

It isn't common, but in some situations, the founder might expose a different ordering service to network participants. In this case, the founder will export the updated ordering settings and you'll upload the revised settings into your participant instance. See Import Ordering Service Settings.

If the founder changes the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.

1. Go to the participant's console and select the Network tab.
2. Click **Ordering Service Settings** and click **View**.

   The Ordering Settings dialog is displayed.

4

Understand and Manage Nodes by Type

This topic contains information to help you understand the different node types and where you can get more information about how the nodes are performing in the network.

Topics:

- Manage CA Nodes
- Manage the Console Node
- Manage Orderer Nodes
- Manage Peer Nodes

Manage CA Nodes

This topic contains information about certificate authority (CA) nodes.

Topics

- View and Edit the CA Node Configuration
- View Health Information for a CA Node

View and Edit the CA Node Configuration

A certificate authority (CA) node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See CA Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the CA node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

View Health Information for a CA Node

You can check a certificate authority (CA) node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays the node's performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the CA node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node's performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

Manage the Console Node

This topic contains information about the console node.

Topics:

- View and Edit the Console Node Configuration
- View Health Information for the Console Node

View and Edit the Console Node Configuration

The console node's configuration determines how it performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See Console Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the console node and click its **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

View Health Information for the Console Node

You can check the console node’s metrics to see how it's performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the console node.

   The Node Information page is displayed.
3. Click the **Health Overview** pane to view the node’s performance metrics.

   Note the following information:

   - If the CPU Utilization percentage is too high, then it might be because too many users are trying to access the console at the same time, or that the console is having technical issues.
   - If the utilization percentages are consistently high, then contact Oracle Support

Manage Orderer Nodes

This topic contains information about orderer nodes.

Topics

- View and Edit the Orderer Node Configuration
- View Health Information for an Orderer Node

View and Edit the Orderer Node Configuration

An orderer node’s configuration determines how the node performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See Orderer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the orderer node that you want configuration information for, and click the node’s **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

View Health Information for an Orderer Node

You can check an orderer node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the orderer node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node’s performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support. If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.

Manage Peer Nodes

This topic contains information about peer nodes.

Topics

- View and Edit the Peer Node Configuration
- List Chaincodes Installed on a Peer Node
- View Health Information for a Peer Node
- Export and Import Peer Nodes

View and Edit the Peer Node Configuration

A peer node’s configuration determines how the node performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See Peer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the peer node that you want configuration information for, and click the node’s **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

List Chaincodes Installed on a Peer Node

You can view a list of the chaincodes and their versions installed on a specific peer node in your network.

If you don't see the chaincode or the chaincode version you were expecting, then you can install a chaincode or upgrade a chaincode to the peer node. You must be an administrator to install or upgrade a chaincode.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the peer node you want to see information for.

   The Node Information page is displayed.
3. Click the **Chaincodes** pane to view a list of chaincodes installed on the selected peer node.

View Health Information for a Peer Node

You can check a peer node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization, memory utilization, user transactions endorsed, and user transactions committed.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the peer node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health Overview** pane to view the node's performance metrics.

   Note the following information:
   - If the CPU Utilization and Memory Utilization percentages are too high, then it might be because the peer is overloaded with endorsement requests. Consider adding another peer or changing the endorsement policy.
   - If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.
   - The User Transactions Endorsed and User Transaction Committed metrics are collected and refreshed every ten minutes. The counts you see are cumulative.
   - If the utilization percentages are consistently high, then contact Oracle Support.

Export and Import Peer Nodes

If you want to run the blockchain transactions through the REST proxy, then after you've added a participant to the network, you must export its peer nodes and import them into the founder.

You need to do this export and import step because the REST proxy's end point configuration needs to know about the peers from both members. After you've completed this step then you'll have to update the founder and participants' REST proxy nodes to add the peers so that the requests can be routed as required by the endorsement policy.

See Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

1. Go to the participant's console and select the Nodes tab.
2. Click **Export/Import Peers** and select **Export**.

   The Export Nodes dialog is displayed.
3. In the Peer List field, select the peer nodes that you want to export. Click **Export**.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. To import, go to the founder's Oracle Blockchain Platform console and select the Nodes tab.
5. Click **Export/Import Peers** and select **Import**.

   The Import Remote Nodes dialog is displayed.
6. Click **Upload remote nodes configurations** and browse for and select the JSON file containing the node configuration information. Usually this file is named `<instance name>-exported-nodes.json`.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
7. Click the plus icon to upload another node configuration file for import.
8. Click **Import**.
9. To confirm that the nodes were added successfully, you can:
   - Go to the founder's Nodes tab and in the nodes table locate the names of the imported peer nodes. Note that the imported nodes type is Remote Peer. You can't view or edit a remote peer's configuration information.
   - Go to the founder's Network tab and click **Topology View** and locate the names of the imported peer nodes.

Manage REST Proxy Nodes

This topic contains information to help you understand, set up, and manage the REST proxy nodes.

Topics

How's the REST Proxy Used?

The REST proxy maps an application identity to a blockchain member, which allows users and applications to call the Oracle Blockchain Platform REST APIs.

Instead of using the native Hyperledger Fabric APIs, Oracle Blockchain Platform can use the REST proxy to interact with the Hyperledger Fabric network. When you use the native Hyperledger Fabric APIs, you connect to the peers and orderer directly. However, the REST proxy allows you to query or invoke a Fabric chaincode through the RESTful protocol.

Add Enrollments to the REST Proxy

Enrollments allow users to call the REST proxy without an enrollment certificate. Enrollements require a new user group to be defined on your authentication server.

Adding Enrollments When Using Microsoft Active Directory as Your Authentication Server

Adding an enrollment to the REST Proxy requires a new user group to be added to Active Directory: `<Rest Proxy Client Users group name>_<custom enrolment name>`. You can then use the Blockchain Platform console to map the enrollment to this group.

1. Create a new Active Directory group called `<Rest Proxy Client Users group name>_<custom enrolment name>`.
2. Add any users needing to use the custom enrollment to this group.
3. Go to the Blockchain Platform console and select the Nodes tab.
4. In the Nodes tab, find the REST proxy node you want to add an enrollment to and open the **More Actions** menu.
5. Click **View or create enrollments** to see a list of the node’s current enrollments.
6. Click **Create New Enrollment**.
7. In the **User Name** field, enter `<custom enrolment name>` from the first step. Note that this is case-sensitive and must match the user group you created. Click **Enroll**.
   - The enrollment is created and displays in the Enrollments table.
   - The new enrollment is copied to each REST proxy node in the network.

Adding Enrollments When Using OpenLDAP or Oracle Internet Directory as Your Authentication Server

Adding an enrollment to the REST Proxy creates a new user role in the `OBP_<platform-name>_<instance-name>_REST_<custom-enrollment>` group on your LDAP server.
After the enrollment is created in the console, the Administrator uses the LDAP server to assign the required users to this role.

For information about how users access the REST resources, see REST API for Oracle Blockchain Platform.

1. Go to the console and select the Nodes tab.

2. In the Nodes tab, find the REST proxy node you want to add an enrollment to and open the **More Actions** menu.
3. Click **View or create enrollments** to see a list of the node’s current enrollments.
4. Click **Create New Enrollment**.

   The Create New Enrollment dialog is displayed.
5. In the **User Name** field, enter a name for the enrollment. Click **Enroll**.

   After you click **Enroll**:
   - The enrollment is created and displays in the Enrollments table.
   - The new enrollment is copied to each REST proxy node in the network.
   - A new user role in the `OBP_<platform-name>_<instance-name>_REST_<custom-enrollment>` group on your LDAP server.

View and Edit the REST Proxy Node Configuration

A REST proxy node’s configuration determines how the node performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See REST Proxy Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the REST proxy node that you want configuration information for, and click the node’s **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s **Proposal Wait Time (ms)** and **Transaction Wait Time (ms)** attributes as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.

View Health Information for a REST Proxy Node

You can check a REST proxy node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the REST proxy node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node’s performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

5 Extend the Network

This topic contains information to help founders add organizations to the blockchain network. This topic also contains information to help organizations join a network.

Topics

- Add Oracle Blockchain Platform Participant Organizations to the Network
- Add Fabric Organizations to the Network
- Add Organizations with Third-Party Certificates to the Network

Add Oracle Blockchain Platform Participant Organizations to the Network

This topic contains information about joining an Oracle Blockchain Platform participant organization to an Oracle Blockchain Platform network.

Topics:

- Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network
- Join a Network

Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network

Here are the tasks the founder and participants organizations need to perform to set up a blockchain network.

Adding Participant Organizations to a Blockchain Network

Task	Who Does This?	Description	More Information
Export the participant organization's certificates and import them into the network	Participant organization outputs certificates Founder organization uploads certificates	In the participant organization's instance, use the wizard to output the certificates into a JSON file and sends them to the founder organization. The founder uploads the certificates to add the participant to the network.	Join a Network Import Certificates to Add Organizations to the Network

Task	Who Does This?	Description	More Information
Download the founder organization's ordering service settings information and upload it to the participant organization	Founder organization downloads ordering service settings information Participant organization uploads ordering service setting information	In the founder's instance, download the ordering service settings information (JSON file). Then in the participant's instance, use the wizard to upload the ordering service settings. This enables the participant to communicate on the network.	Join a Network
Export and import the participant organization's peer nodes	Participant organization exports peers Founder organization imports peers	In the participant's instance, export the peers that you want to use on the network. Then in the founder's instance, import the peer nodes.	Export and Import Peer Nodes

Join Participant Organizations to the Channel and Set Anchor Peers

Task	Who Does This?	Description	More Information
Create a channel	Founder organization	In the founder's instance, create a channel that the founder and participants use to communicate. Add the founder's peers to the channel. You must select any newly added participants and assign them permissions on the channel. Note that instead of creating a new channel, you can add participants to an existing channel.	Create a Channel
Join participants to the channel	Participant organization	In the participant's instance, join the channel that was created in the founder's instance.	Join a Peer to a Channel
Set anchor peers on the founder and participants	Founder organization Participant organization	In the founder and participant instances, specify which peers you want to use as anchor peers. You must select at least one anchor peer for each member.	Add an Anchor Peer

Deploy the Chaincode Across the Blockchain Network

Task	Who Does This?	Description	More Information
Install the chaincode on the founder	Founder organization	In the founder's instance, upload and install the chaincode. Choose the peers to install the chaincode on.	Use Quick Deployment
Instantiate the chaincode and specify an endorsement policy on the founder	Founder organization	In the founder's instance, instantiate the chaincode to activate it on the network. An endorsement policy is required to specify the number of members that must approve chaincode transactions before they're submitted to the ledger.	Instantiate a Chaincode Specify an Endorsement Policy
Install the chaincode on the participant	Participant organization	In the participant's instance, install the chaincode that your network will use. Because you'll install the same chaincode that you installed and instantiated on the founder, you don't need to instantiate the chaincode on the participant. When the participant installs the chaincode, it's already instantiated.	Use Quick Deployment

Expose the Chaincode's REST API and Run Transactions

Task	Who Does This?	Description	More Information
Configure the founder's REST proxy node	Founder organization	In the founder's instance, modify the REST proxy node's attributes to specify the channel, chaincode, and peers that the network will use for transactions.	View and Edit the REST Proxy Node Configuration
Configure the participant's REST proxy node	Participant organization	In the participant's instance, modify the REST proxy node's attributes to specify the channel, chaincode, and peers that the network will use for transactions.	View and Edit the REST Proxy Node Configuration

Task	Who Does This?	Description	More Information
Invoke the chaincode and monitor network activity and ledger updates	Founder organization Participant organization	Begin using your network's chaincode for transactions. Both the founder and the participants can use their consoles to find out information about the activity on the network. Specifically, you can use the console's Channels tab to locate information about specific ledger transactions	Find Information About Nodes View a Channel's Ledger Activity

Join a Network

Participant organization are required to complete a wizard to join a blockchain network. The wizard displays the first time the participant organization opens its instance.

The wizard assists the participant organization with exporting the certificates to a JSON file to give to the network founder. The wizard also helps the participant import the founder's ordering service settings. For more information about the steps the founder and participant must complete to create a network, see Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

The participant's dashboard won't display until the wizard has been completed. If you're a network founder, then this wizard is never displayed.

1. Open the participant organization's console.

   The wizard that you'll use to join a network is displayed.

2. In the wizard, click **Export Certificates** and click the **Export** button.

   The Export dialog is displayed and includes the name of the JSON file the export will create.

3. Specify where to save the file. Click **OK** to save the certificates file.
4. Send the certificates JSON file to the network's founder. The network founder will import the participant's certificates file into the network.
5. Get the ordering services settings JSON file from the network founder. You'll import this file into your instance.
6. In the wizard, click **Import Ordering Service Settings**.
7. Click **Upload Ordering Service Settings**.

   The File Upload dialog is displayed.

8. In the File Upload dialog, browse for and select the JSON file containing the founder's ordering service settings information. Usually this file is named `<founderinstancename>-orderer-settings.json`. Click **Open**.

   The **Current Orderer Address** field updates with the address that Oracle Blockchain Platform extracted from the JSON file.

9. Click **Submit**.

   Your console's Dashboard tab is displayed.

Add Fabric Organizations to the Network

This topic contains information about joining Hyperledger Fabric organizations to an Oracle Blockchain Platform network.

Topics:

- Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network
- Create a Fabric Organization's Certificates File
- Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network

Here are the tasks that a Fabric organization and the Oracle Blockchain Platform founder organization need to perform to join a Fabric organization to the Oracle Blockchain Platform network.

Task	Who Does This?	Description	More Information
Create the certificate file for the Fabric organization	Fabric organization	Find the Fabric organization’s Admin, CA, and TLS certificate information and use it to compose a JSON certificates file.	Create a Fabric Organization's Certificates File
Upload Fabric organization's certificate file to the Oracle Blockchain Platform network	Founder organization	Use the console to upload and import the Fabric organization's certificate file to add the Fabric organization to the network.	Import Certificates to Add Organizations to the Network
Create a channel	Founder organization	Create a new channel and add the Fabric organization to it.	Create a Channel
Export the ordering service settings from founder	Founder organization	Output the founder’s ordering services settings to a JSON file and send the file to the Fabric organization.	Export Ordering Service Settings

Task	Who Does This?	Description	More Information

Compose orderer certificate file	Fabric organization	Create a file named orderer.pem that includes the tlscacert information. Go to the exported ordering service settings file and copy the tlscacert information. After you paste the tlscacert information into the orderer.pem file, you must replace all instances of \n with the newline character.
		
		The orderer.pem file must have the following format:
		
		`-----BEGIN CERTIFICATE-----`
		`...`
		`...`
		`...`
		`-----END CERTIFICATE-----`
			Create a Fabric Organization's Certificates File

Provide ordering service settings	Founder organization	Open the ordering service settings file and find the ordering service's address and port and give them to the Fabric organization. For example:
		
		`"orderingServiceNodes": [`
		`{`
		`"address": "grpcs://example_address:7777",`
		`...`
		`}]`
			NA

Add the Fabric organization to the network	Fabric organization	The Fabric organization copies certificates into its environment, sets environment variables, fetches the genesis block, joins the channel, and installs the chaincode.	Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

Create a Fabric Organization's Certificates File

For a Fabric organization to join an Oracle Blockchain Platform network, it must write a certificates file containing its admincerts, cacerts, and tlscacerts information. The

Oracle Blockchain Platform founder organization imports this file to add the Fabric organization to the network.

The Fabric certificates information is stored in PEM files located in the Fabric organization’s MSP folder. For example, `network_name_example/crypto-config/peerOrganizations/example_org.com/msp/`.

The certificate file must be in written in JSON and contain the following fields:

- **mspid** — Specifies the name of the Fabric organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization’s Admin certificates file: `Admin@example_org.com-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **cacert** — Contains the contents of the organization’s CA certificates file: `ca.example_org-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **tlscert** — Contains the contents of the organization’s TLS certificate file: `tlsca.example_org-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.

This is how the file needs to be structured:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "tlscacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
 }
}
```

Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

You must modify the Fabric organization’s environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For more information, see Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network.

- The Fabric organization’s certificate file was created and sent to the Oracle Blockchain Platform network founder.

- The network founder uploaded the certificates file to add the Fabric organization to the network.
- The network founder created a new channel and added the Fabric organization to it.
- The network founder downloaded its ordering service settings and sent them to the Fabric organization.
- The Fabric organization created the orderer certificate file.
- The network founder gave the ordering service address and port to the Fabric organization.

You must add the Fabric organization and install and test the chaincode.

1. Navigate to the Fabric network directory and launch the peer container.
2. Fetch the channel's genesis block with this command:

   ```
   peer channel fetch 0 mychannel.block -o ${orderer_addr}:$
   {orderer_port} -c mychannel --tls --cafile orderer.pem --logging-
   level debug
   ```

   Where:

   - `{orderer_addr}` is the Founder's orderer address.
   - `{orderer_port}` is the Founder's port number.
   - `-c mychannel` is the name of the channel that the Founder created. This is the channel where the Fabric organization will send and receive transactions on the Oracle Blockchain Platform network.
   - `orderer.pem` is the Founder's orderer certificate file.
3. Join the channel with this command:

   ```
   peer channel join -b mychannel.block -o ${orderer_addr}:$
   {orderer_port} --tls --cafile orderer.pem --logging-level debug
   ```

4. Install the chaincode with this command:

   ```
   peer chaincode install -n mycc -v 1.0 -l "golang" -p ${CC_SRC_PATH}
   ```

   Where `CC_SRC_PATH` is the folder that contains the chaincode.
5. Instantiate the chaincode with this command:

   ```
   peer chaincode instantiate -o  ${orderer_addr}:${orderer_port} --
   tls --cafile orderer.pem -C mychannel -n mycc -l golang -v
   1.0 -c '{"Args":["init","a","100","b","200"]}' -P <policy_string> --
   logging-level debug
   ```

6. Invoke the chaincode with this command:

   ```
   peer chaincode invoke -o ${orderer_addr}:${orderer_port}  --tls
   true --cafile orderer.pem -C mychannel -n mycc -c '{"Args":
   ["invoke","a","b","10"]}' --logging-level debug
   ```

7. Query the chaincode with this command:

```
peer chaincode query -C mychannel -n mycc -c '{"Args":
["query","a"]}'  --logging-level debug
```

Add Organizations with Third-Party Certificates to the Network

This topic contains information about joining organizations using third-party certificates to an Oracle Blockchain Platform network.

Topics:

- Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network
- Third-Party Certificate Requirements
- Create an Organization's Third-Party Certificates File
- Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network

Here are the tasks that an organization with third-party certificates and the Oracle Blockchain Platform founder need to perform to join the organization to an Oracle Blockchain Platform network.

Organization with certificates issued by a third-party certificate authority (CA) can join the Oracle Blockchain Platform network as participants. These participants are client-only organizations and have no peers or orderers. After joining the network, these participants can use an SDK or a Hyperledger Fabric command line interface (CLI) to start blockchain transactions on the network.

Task	Who Does This?	Description	More Information
Get the third-party certificates	Third-party certificates (participant) organization	Go to the third-party CA server and generate the required certificates files. Format the files as needed for import into the network.	Third-Party Certificate Requirements
Create the certificates file for import	Third-party certificates (participant) organization	Find the participant’s Admin and CA certificate information and use it to compose a JSON certificates file.	Create an Organization's Third-Party Certificates File
Upload a certificate file for the third-party (participant) organization	Founder organization	Use the console to upload and import the participant’s certificate file to add the participant to the network.	Import Certificates to Add Organizations to the Network

Task	Who Does This?	Description	More Information
Export the ordering service settings from network founder and provide them to the third-party (participant) organization	Founder organization	Output the founder's ordering services settings to a JSON file and send the file to the participant. Open the ordering service settings file and find the ordering service's address and port and give them to the participant. For example: `"orderingServiceNodes": [ { "address": "grpcs:// example_address:7777" ... }]`	Export Ordering Service Settings
Create the channel	Founder	Create a new channel and add the participant to it.	Create a Channel
Install and instantiate the chaincode	Founder	In the founder's instance, upload, install, and instantiate the chaincode. Choose the network peers to install the chaincode on.	Use Quick Deployment
Set up the third-party (participant) organization's environment	Third-party certificates (participant) organization	To query or invoke chaincodes, the participant must: • Add the founder's ordering service's address and port to the participant's environment. • Configure the environment to use Hyperledger Fabric CLI or SDKs. • Install the chaincode on peers.	Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

Third-Party Certificate Requirements

To successfully join the network, an organization must generate the required third-party certificates. The information in these certificates is used to create the organization's certificates file, which is then imported into the founder's instance.

Which Certificates Do Organizations Need to Provide?

You must generate the following certificates from your CA server:

- Client Public Certificate

- CA Root Certificate

What Are the Requirements for These Certificates?

The certificates must meet the following requirements:

- When generating the private key, you must use the Elliptic Curve Digital Signature Algorithm (ECDSA). This algorithm is the only accepted algorithm for Fabric MSP keys.
- The Subject Key Identifier (SKI) is mandatory and you must indicate it as x509 extensions in the extension file.
- You must convert the key files from the .key to the .pem format.
- You must convert the certificates from the .crt to the .pem format.

After confirming that you've outputted and updated the proper files, you can then create the certificates file for import into the Oracle Blockchain Platform network. See Create an Organization's Third-Party Certificates File.

Create an Organization's Third-Party Certificates File

To join an Oracle Blockchain Platform network, the organization must write a certificates file containing its admincert and cacert information. The network founder imports this file to add the organization to the network.

These organizations are client-only and have no peers or orderers. After joining the network, these organizations can use an SDK or a Hyperledger Fabric CLI to start blockchain transactions on the network.

Go to the certificates files that you generated from the CA Server to find the information that you need to create the certificates file. See Third-Party Certificate Requirements.

The certificates file must be in written in JSON and contain the following fields:

- **mspid** — Specifies the name of the organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization's Admin certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **cacert** — Contains the contents of the organization's CA certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.

This is how the file needs to be structured:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
 }
}
```

Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

You must set up the third-party organization's environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For information, see Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network.

- The third-party organization’s certificate file was created and sent to the Oracle Blockchain Platform network founder.
- The network founder uploaded the certificates file to add the third-party organization to the network.
- The network founder exported the orderer service's settings and gave the service's address and port to the third-party organization and the organization added them to the environment.
- The network founder created a new channel and added the third-party organization to it.
- The network founder installed and instantiated the chaincode.

Setup organization's Environment

Before the third-party organization can successfully use the Oracle Blockchain Platform network, it must set up its environment to use Hyperledger Fabric CLI or SDKs. See Welcome to Hyperledger Fabric.

Install the Chaincode

The third-party organization must install the chaincode on the peers. These peers must then be joined to the channel so that the chaincode can be invoked.

Instantiate the Chaincode

If needed, the third-party organizations can instantiate the chaincode on the channel. For example:

```
export  CORE_PEER_TLS_ENABLED=true
export  CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export  CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/Admin@customerorg1.com/msp
export  CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input###
CHANNEL_NAME=$1

echo "######### going to instantiate chaincode on channel $
{CHANNEL_NAME} ##########"
CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode instantiate
```

```
-o ${peer_host}:${port}  --tls $CORE_PEER_TLS_ENABLED --cafile
./tls-ca.pem -C ${CHANNEL_NAME}  -n obcs-example02 -v v0 -c '{"Args":
["init","a","100","b","200"]}'
```

Invoke the Chaincode

Third-party organizations use the Hyperledger Fabric CLI or SDKs to invoke the chaincode. For example:

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/User1@customerorg1.com/msp
export CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input ###
CHANNEL_NAME=$1

#### do query or invoke on chaincode ####

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode query -C
${CHANNEL_NAME} -n $2 -c '{"Args":["query","a"]}'

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode invoke -o
${peer_host}:${port} --tls $CORE_PEER_TLS_ENABLED --cafile ./tls-
ca.pem -C ${CHANNEL_NAME} -n $2 -c '{"Args":["invoke","a","b","10"]}'
```

6

Develop Chaincodes

This topic contains information to help you understand how to write and test chaincodes for use in Oracle Blockchain Platform.

Topics

Write a Chaincode

A chaincode is written in Go, Node.js, or Java and then packaged into a ZIP file that is installed on the Oracle Blockchain Platform network.

Chaincodes define the data schema in the ledger, initialize it, perform updates when triggered by applications, and respond to queries. Chaincodes can also post events that allow applications to be notified and perform downstream operations. For example, after purchase orders, invoices, and delivery records have been matched by a chaincode, it can post an event so that a subscribing application can process related payments and update an internal ERP system.

Resources for Chaincode Development

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write valid chaincodes.

- Welcome to Hyperledger Fabric. The Key Concepts and Tutorials sections should be read before you write you own chaincode.
- Go Programming Language. The Go compilers, tools, and libraries provide a variety of resources that simplify writing chaincodes.
- Package shim. Package shim provides APIs for the chaincode to access its state variables, transaction context and call other chaincodes. This documents the actual syntax required for your chaincode.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See What Are Chaincode Samples?

You can add rich-query syntax to your chaincodes to query the state database. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

Package and Zip a Go Chaincode

Once you've written your chaincode, place it in a ZIP file. You don't need to create a package for the Go chaincode or sign it — the Oracle Blockchain Platform install and instantiation process does this for you as described in Typical Workflow to Deploy Chaincodes.

If your chaincode has any external dependencies, you can place them in the vendor directory of your ZIP file.

Package and Zip a Node.js Chaincode

If you're writing a Node.js chaincode, you need to create a `package.json` file with two sections:

- The scripts section declares how to launch the chaincode.
- The dependencies section specifies the dependencies.

The following is a sample `package.json` for a Node.js chaincode:

```
{
    "name": "chaincode_example02",
    "version": "1.0.0",
    "description": "chaincode_example02 chaincode implemented in
Node.js",
    "engines": {
        "node": ">=8.4.0",
        "npm": ">=5.3.0"
    },
    "scripts": { "start" : "node chaincode_example02.js" },
    "engine-strict": true,
    "license": "Apache-2.0",
    "dependencies": {
        "fabric-shim": "~1.3.0"
    }
}
```

The packaging rules for a Node.js chaincode are:

- `package.json` must be in the root directory.
- The entry JavaScript file can be located anywhere in the package.
- If `"start" : "node <start>.js"` isn't specified in the `package.json`, `server.js` must be in the root directory.

Place the chaincode and package file in a zip file to install it on Oracle Blockchain Platform.

Package and Zip a Java Chaincode

If you're writing a Java chaincode, you can choose Gradle or Maven to build the chaincode.

If you're using Gradle, place the chaincode, build.gradle, and settings.gradle in a zip file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode zip package:

```
Archive:  example_gradle.zip
 Length      Date    Time    Name
---------  ---------- -----   ----
      610  02-14-2019 01:36   build.gradle
       54  02-14-2019 01:28   settings.gradle
        0  02-14-2019 01:28   src/
        0  02-14-2019 01:28   src/main/
        0  02-14-2019 01:28   src/main/java/
```

```
         0  02-14-2019 01:28   src/main/java/org/
         0  02-14-2019 01:28   src/main/java/org/hyperledger/
         0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/
         0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
      5357  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
SimpleChaincode.java
---------                     -------
      6021                     10 files
```

If you're using Maven, place the chaincode and pom.xml in a zip file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode zip package:

```
Archive:  example_maven.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
      3313  02-14-2019 01:52   pom.xml
         0  02-14-2019 01:28   src/
         0  02-14-2019 01:28   src/chaincode/
         0  02-14-2019 01:28   src/chaincode/example/
      4281  02-14-2019 01:28   src/chaincode/example/SimpleChaincode.java
---------                     -------
      7594                     5 files
```

Testing a Chaincode

After you write your chaincode, then you need to test it. See:

- Use a Mock Shim to Test a Chaincode
- Deploy a Chaincode on a Peer to Test the Chaincode

Installing and Instantiating a Chaincode

Once you've tested your chaincode, you can deploy it following the information in Typical Workflow to Deploy Chaincodes.

Upgrading a Chaincode

A chaincode may be upgraded any time by changing its version. The chaincode name must be the same or it would be considered a totally different chaincode.

1. Change the chaincode version
2. Follow the instructions in Upgrade a Chaincode to install and upgrade the new version of the chaincode.

Use a Mock Shim to Test a Chaincode

This method of testing involves using a mock version of the stub `shim.ChaincodeStubInterface`. With this you can simulate some functionality of your chaincode before deploying it to Oracle Blockchain Platform. You can also use this library to build unit tests for your chaincode.

Manually Vendor the Shim with a Chaincode

This applies to Oracle Blockchain Platform 19.1.1, 19.1.3, 19.2.1, and 19.2.3.

In Hyperledger Fabric, the fabric-ccenv image contains the `github.com/hyperledger/fabric/core/chaincode/shim` (shim) package. This allows you to package a chaincode without needing to include the shim. However, this may cause issues in future Hyperledger Fabric releases, and it may cause issues when using packages that are included with the shim.

Workaround: To avoid potential issues, you should manually vendor the shim package with the chaincode prior to using the `peer` command-line interface for packaging and installing a chaincode, or packaging or installing a chaincode. See https://jira.hyperledger.org/browse/FAB-5177.

Use a Mock Shim to Test a Chaincode

1. Create a test file that matches the name of the chaincode file.

   For example, if `car_dealer.go` is the actual implementation code for you smart contract, you would create a test suite called `car_dealer_test.go` containing all the tests for `car_dealer.go`. The test suite filename should be in the `*_test.go` format.

2. Create your package and import statements.

```
package main

import (
    "fmt"
    "testing"

    "github.com/hyperledger/fabric/core/chaincode/shim"
)
```

3. Create your unit test.

```
/*
* TestInvokeInitVehiclePart simulates an initVehiclePart
transaction on the CarDemo cahincode
 */
func TestInvokeInitVehiclePart(t *testing.T) {
    fmt.Println("Entering TestInvokeInitVehiclePart")

    // Instantiate mockStub using CarDemo as the target chaincode
to unit test
    stub := shim.NewMockStub("mockStub", new(CarDemo))
    if stub == nil {
        t.Fatalf("MockStub creation failed")
    }

    var serialNumber = "ser1234"

    // Here we perform a "mock invoke" to invoke the function
"initVehiclePart" method with associated parameters
    // The first parameter is the function we are invoking
    result := stub.MockInvoke("001",
        [][]byte{[]byte("initVehiclePart"),
            []byte(serialNumber),
            []byte("tata"),
            []byte("1502688979"),
            []byte("airbag 2020"),
            []byte("aaimler ag / mercedes")})
        // We expect a shim.ok if all goes well
        if result.Status != shim.OK {
            t.Fatalf("Expected unauthorized user error to be returned")
        }

        // here we validate we can retrieve the vehiclePart object we
    just committed by serianNumber
        valAsbytes, err := stub.GetState(serialNumber)
        if err != nil {
            t.Errorf("Failed to get state for " + serialNumber)
        } else if valAsbytes == nil {
            t.Errorf("Vehicle part does not exist: " + serialNumber)
        }
    }
```

Note:

Not all interfaces of the stub are implemented. Stub functions

- `GetQueryResult`
- `GetHistoryForKey`

are not supported, and attempting to call either of these will result in an error.

Deploy a Chaincode on a Peer to Test the Chaincode

After you create a chaincode, you must install, instantiate, and invoke it to test that it works correctly.

If you need help writing a chaincode, see Write a Chaincode.

Follow these steps to deploy and test your chaincode.

1. Identify the channel or create a new channel and add peers to it. See Join a Peer to a Channel.
2. Install the chaincode on the peers and instantiate it on the channel. See Use Quick Deployment.
3. Use the *Invoke* and *query* REST APIs to test the chaincode with cURL through the REST proxy. See REST API for Oracle Blockchain Platform for descriptions of each endpoint and correct cURL syntax to invoke each operation.
4. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
5. In the channel's **Ledger** pane, view the chaincode's ledger summary.

7

Deploy and Manage Chaincodes

This topic contains information to help you deploy (install, instantiate, upgrade, and enable in the REST proxy), monitor, and find information about the chaincodes on the network.

Topics

Typical Workflow to Deploy Chaincodes

Here are the common tasks for deploying chaincodes.

You must be an administrator to perform these tasks.

Task	Description	More Information
Use the wizard to fully or partially deploy a chaincode	For testing, use Quick Deployment to perform the deployment in one step, using default settings. For production, use Advanced Deployment to specify the deployment settings such as which peers to install the chaincode on and the endorsement policy you want to use. With Advanced Deployment you can instantiate the chaincode and enable it in the REST proxy now or later.	Use Quick Deployment Use Advanced Deployment

Task	Description	More Information
Instantiate a chaincode	Instantiate the chaincode after you've installed it.	Instantiate a Chaincode
Upgrade the chaincode	Upload and instantiate a newer version of a chaincode, or pick an older version of the chaincode to use.	Upgrade a Chaincode

Use Quick Deployment

Use the quick deployment option to perform a one-step chaincode deployment. This option is recommended for chaincode testing.

The quick deployment uses default settings, installs the chaincode on all peers in the channel, instantiates the chaincode using the default endorsement policy, and enables the chaincode in the REST proxy.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- You can use the advanced deployment option to put your chaincode into production on the network. See Use Advanced Deployment.
- You can't delete a chaincode from the network.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Quick Deployment**.

   The Deploy Chaincode (Quick) page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field enter a string value to specify the chaincode's version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).
5. Review the other default settings and modify them as needed.

6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy.
7. Click **Submit**.

   The chaincode is installed on the channel's peers, instantiated, and enabled in the REST proxy. The deployed chaincode's name is displayed in the Chaincode tab's table.

Use Advanced Deployment

Use the advanced deployment option to specify the parameters required to deploy a chaincode into a production environment. For example, you'll specify which peers to install the chaincode on and the endorsement policy to use.

With the advanced deployment wizard, you'll install the chaincode on the peers you select.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- You can use the quick deployment option for chaincode testing. Quick deployment is a one-step deployment that uses default settings, installs the chaincode on all peers in the channel, and instantiates the chaincode using a default endorsement policy. See Use Quick Deployment.
- You can't delete a chaincode from the network.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Advanced Deployment**.

   The Deploy Chaincode (Advanced) Step 1 of 3: Install page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field, enter the chaincode's version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).

5. Select one or more network peers to install the chaincode onto. To provide high availability, Oracle suggests that you choose the appropriate number of peers from each partition. Also, the peers you choose must be joined to the channel that you'll instantiate the chaincode on.
6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy. Click **Next**.

   The chaincode is installed and the Deploy Chaincode (Advanced) Step 2 of 3: Instantiate page is displayed.
7. Decide if you want to instantiate the chaincode now or later.
   - Click **Close** to close the wizard and instantiate later.
   - To instantiate now, select the channel to instantiate the chaincode on and the peers to instantiate the chaincode to. If required, enter initial parameters, an endorsement policy, transient map, and private data collections. Note the following information:
     – Instantiation compiles, builds, and initializes the chaincode on the peers.
     – If you leave the endorsement policy blank, then Oracle Blockchain Platform uses the default endorsement policy. The default endorsement policy gets an endorsement from any peer on the network.
     – When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

     Click **Next**.

   The chaincode is instantiated.

Update REST Proxy Settings for Running Chaincodes

If you're using Node.js or Java chaincodes which are dependent on external libraries, a few proxy settings must be updated.

You must complete these steps before you instantiate the chaincode. See Instantiate a Chaincode.

Node.js Chaincode

When your Node.js chaincode is instantiated, npm is used to install all dependency libraries from the internet, so in an Oracle Blockchain Platform instance, if there is such a dependancy, you need to ensure the `bcs/fabric-ccenv` image has internet access.

Set the HTTP proxy for the `bcs/fabric-ccenv` image following these steps:

1. Create a Docker file in any location in your VM with the following content (where http://hostname:port is your HTTP proxy access entry):

```
FROM bcs/fabric-ccenv:latest
ENV npm_config_proxy http://hostname:port
```

2. Build the image again:

```
docker build -f Dockerfile -t bcs/fabric-ccenv:latest
```

Java Chaincode

To configure the proxy to run Java chaincode:

1. Gradle-only: Create `gradle.properties` in a local directory, and add the following content in it:

   ```
   systemProp.http.proxyHost=[proxy host]
   systemProp.http.proxyPort=[proxy port]
   systemProp.https.proxyHost=[proxy host]
   systemProp.https.proxyPort=[proxy port]
   ```

2. Maven-only: Create `settings.xml` in a local directory, and add the following content in it:

   ```
   <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
   https://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository/>
   <interactiveMode/>
   <usePluginRegistry/>
   <offline/>
   <pluginGroups/>
   <servers/>
   <mirrors/>
   <proxies>
   <proxy>
   <id>httpproxy</id>
   <active>true</active>
   <protocol>http</protocol>
   <host>[proxy host]</host>
   <port>[proxy port]</port>
   </proxy>
   <proxy>
   <id>httpsproxy</id>
   <active>true</active>
   <protocol>https</protocol>
   <host>[proxy host]</host>
   <port>[proxy port]</port>
   </proxy>
   </proxies>
   <profiles/>
   <activeProfiles/>
   </settings>
   ```

3. Create a Docker file and add the following to it:

   ```
   FROM bcs/fabric-javaenv:latest
   COPY gradle.properties /root/.gradle/
   COPY settings.xml /root/.m2/
   ```

4. Create a new image:

   ```
   docker build -f dockerfile -t bcs/fabric-javaenv:latest
   ```

Instantiate a Chaincode

Instantiating a chaincode compiles, builds, and initializes the chaincode on the peers where the chaincode is installed. When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

Note the following information:

- You must install the chaincode on the required peers before you can instantiate it.
- If you're working on a channel that contains multiple members and have instantiated the chaincode on one member, then you don’t have to instantiate the chaincode on the other members where you installed the same chaincode. In such cases, the chaincode is already instantiated and running on all members on the channel.
- You can instantiate more than one chaincode on a channel.
- The process to instantiate the sample chaincodes is different than the instantiation process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- After you instantiate the chaincode, then you can optionally enable it in the REST proxy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click the arrow to expand the chaincode’s version list.
3. Locate the chaincode version and click its **More Actions** menu, and select **Instantiate**.

   The Instantiate Chaincode dialog is displayed.
4. Enter information about where and how to instantiate the chaincode.

Field	Description
Channel	Select the channel for the chaincode to run on.
Peers	Select the peer or peers you want to use the chaincode. This list shows the peers that you installed the chaincode onto.
Initial Parameter	Enter the input parameters that you want to pass to the chaincode. Go to the chaincode to find the initial parameters values.
Endorsement Policy	In this section, specify the number and role of members required to endorse the chaincode. If you don’t specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

Field	Description
Transient Map	The data that is passed into the chaincode is the transaction payload and the transient map. The transaction payload is recorded in the ledger and is visible to anyone who can access the ledger through the query system chaincode. Use a transient map to pass private data such as keys that you don't want stored in the ledger. In this section, provide the required keys and values. The information you provide is maintained on the peer node and is sent to the chaincode when a transaction is executed. If you're adding private data collections, then specify a transient map to pass the private data from the client to the peers for endorsement.
(New in v19.1.3) Private Data Collections	In this section, add one or more private data collections. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel you instantiate the chaincode on.

5. Click **Instantiate**.

   The chaincode is instantiated.

6. To confirm that the chaincode was instantiated, go to the Channels tab and click the name of the channel that you instantiated the chaincode on. Go to the Instantiated Chaincodes tab and confirm that the chaincode is listed in the summary table.

Specify an Endorsement Policy

You can add an endorsement policy when you instantiate a chaincode. An endorsement policy specifies the members with peers that must approve, or properly endorse, a chaincode transaction before it’s added to a block and submitted to the ledger.

Endorsement guarantees the legitimacy of a transaction. When you instantiate a chaincode on a channel, you can specify an endorsement policy. If you don’t specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

A member’s endorsing peers must have ReaderWriter permissions on the channel. When a transaction is processed, each endorsing peer returns a signed read-write set. After the client has enough endorsements to meet the endorsement policy requirements, then the client bundles the common read-write set with the signature from the endorsing peers and sends everything to the ordering service, which orders and commits the transactions into blocks and then to the ledger.

You can go to the Channels tab to view an instantiated chaincode’s endorsement policy. See View an Endorsement Policy. You can’t modify an instantiated chaincode's endorsement policy. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Endorsement Policy section. Click **Add Identity** to add members to the policy as needed.

Field	Description
MSP ID	From the dropdown menu, select the endorser peer’s organization.
Role	Select the corresponding peer role required by the endorsement policy. Usually this will be member. You can find a peer’s role by viewing its configuration information.
Policy Expression Mode	In most cases, you’ll use **Basic**. Select **Advanced** to provide an expression string. See the Hyperledger Fabric documentation for information about how to write a valid expression string.
Signed By	Select how many members with endorsing peers (peers with ReaderWriter permissions) on the channel must endorse the chaincode transactions to make them valid.

4. Complete the other fields on the Instantiate Chaincode page as needed.
5. Click **Instantiate**.

View an Endorsement Policy

You can view an instantiated chaincode's endorsement policy.

You might need to view an instantiated chaincode's endorsement policy to see how it was set up, how you need to choose transaction endorsers based on the policy, or to help resolve an endorsement failure.

You can’t modify the endorsement policy for an instantiated chaincode. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.
2. Locate the chaincode that you want to view endorsement policy information for and expand it in the table.
3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.
4. In the Instantiated on Channels tab, locate the channel that you want, click **More Actions**, and select **View Endorsement Policy**.

   The Chaincode Endorsement Policy page is displayed.

Find Information About Chaincodes

You can find information about the chaincodes in your network. For example, how many peers the chaincode is installed on and if the chaincode has been instantiated.

You can’t delete a chaincode from the network.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes and versions installed on the network.
2. In the chaincode table, locate the chaincode that you want information for and expand it to see information about its versions, path, how many peers it’s installed on, and how many channels it’s instantiated on.

Note the following information:

- When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Chaincodes tab.
- If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. To list the chaincode, start at least one peer node that has the chaincode installed on it.

3. Use the chaincode table as a starting point to perform chaincode-related tasks, such as instantiate, enable it in the REST proxy, and upgrade to a new version.

Manage Chaincode Versions

Each chaincode that you install or upgrade has a version number. Once installed, a chaincode and any of its versions can't be deleted.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes installed on the network.
2. Locate the chaincode that you want version information for and expand it to see a list of versions.
3. Click a version number. The Chaincode Version Information page is displayed.
4. Click the Installed on Peers pane to see which peers the chaincode is installed on. You can click the peer to view more information about it.
5. Click the Instantiated on Channels pane to see which channels the chaincode is instantiated on. You can click a channel to view more information about it.

   From this pane, you can also instantiate a specific version of the chaincode version. If the chaincode was instantiated on a channel, then you can view its endorsement policy.

   Note that you can instantiate different versions of a chaincode on different channels.
6. **(New in v19.1.3)** Click the Private Data Collections pane to view the private data collections that were added when the chaincode was instantiated.

Upgrade a Chaincode

If a developer modifies a chaincode's source, then you'll need to deploy it to a new version of the chaincode. If needed, you can revert back to an older version of a chaincode.

You can instantiate different versions of the same chaincode on different channels.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists all of the chaincodes installed on the network.
2. Locate the chaincode that you want to upgrade, click **More Actions**, and select **Upgrade**. The **More Actions** button only displays for chaincodes that have been instantiated.

The Upgrade Chaincode Step 1 of 2: Select a version page is displayed.

3. Select a version source. Note the following information:
   - Click **Select from existing versions** if you want to upgrade to a version that is already on the network. You might choose this option because the most current chaincode version contains errors and you need to temporarily use an older version until the chaincode can be fixed. Because the older version is on your system, the chaincode is already installed on the peers.
   - Choose **Install a new version** to upload the chaincode file. In the **Version** field enter a version number and in the **Target Peers** field, select the peers to install the chaincode on. In the **Chaincode Source** field, click **Upload Chaincode File** and browse for the chaincode ZIP file to upload.
4. Click **Next**.

   The Upgrade Chaincode Step 2 of 2: Upgrade page is displayed.
5. Decide if you want to instantiate the chaincode version now or later.
   - Click **Close** to close the wizard and upgrade later.
   - To upgrade now, select the channel to upgrade the chaincode on and the peers to instantiate the chaincode to. If required, enter initialize parameters, an endorsement policy, and transient map. See Specify an Endorsement Policy. Click **Next**.

   The chaincode is upgraded.

What Are Private Data Collections?

(19.1.3 and later versions only) Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data and to prevent the other organizations on the channel from seeing the data. Private data is distributed peer to peer and not by blocks, so the transaction data is kept confidential from the ordering service. Collections help you reduce the number of channels and their required maintenance on your network.

The primary components in a private data collection are:

- The private data that you specify in your private data collection definition. Private data is sent with the gossip protocol from peer to peer within the organizations that you specify in your policy. Private data is stored in a private database on the peer. The ordering service isn't used and can't see the private data.
- A hash of the data, which is endorsed, ordered, and written to each peer on the channel. This hash is evidence of the transaction and can be used for audit purposes.

When you instantiate a chaincode, you can associate it with one or more private data collections. Also when you instantiate a chaincode, you should specify a transient map to pass the private data from the client to the peers for endorsement. The collection definition specifies who can persist data, how many peers the data is distributed to, how many peers are required to disseminate the private data, and how long the private data is persisted in the private database.

Add Private Data Collections

(19.1.3 and later versions only) You can add private data collections to channels. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data within a transaction and to prevent the other organizations on the channel from seeing the data.

If you're going to use private data collections across the organizations in your network, then you need to configure anchor peers. Anchor peers facilitate private data gossip among the organizations. See Add an Anchor Peer.

You specify the private data collections when you instantiate the chaincode.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Private Data Collections section and add the collection definition as needed.

Field	Description
Collection Name	Enter the collection's name. You'll reference this name in the chaincode.
Policy	Create the policy to specify which organizations are included in the collection and which peers can store the private data. Each member listed in the policy must be included in an `OR` signature policy list. To support read/write transactions, the private data distribution policy must contain more organizations than the chaincode endorsement policy because peers must have the private data to endorse transactions. For example, in a channel with ten organizations, five of the organizations are included in a private data collection policy, but the endorsement policy requires three organizations to endorse a transaction.

Field	Description
Peers Required	Enter the number of peers that each endorsing peer must distribute private data to before the peer signs the endorsement and returns the proposal response. Oracle recommends that you set this value to 1 or more peers to: • Ensure redundancy of the private data on multiple peers in the network. • Ensure that private data is available if the endorsing peers become unavailable. Note that setting this value to 0 means that distribution isn't required. However, if the **Max Peer Count** field is set to greater than 0, then private data distribution might still occur.
Max Peer Count	Enter the maximum number of peers that the current endorsing peer attempts to distribute the data to. This is to ensure redundancy so that peers are available between endorsement time and commit time to pull the private data if an endorsing peer isn't available. If you set this value to 0, then the private data isn't distributes at the time of endorsement. This causes private data pulls against the endorsing peers on all authorized peers at commit time.
Block to Live	Enter the length in number of blocks that you want data to reside on the private database. The data is purged when the number of blocks is reached. Set this value to 0 if you never want to purge the data. Note that a peer can fail to pull private data from another peer if a private data collection's blocktolive value is less than 10, and its requiredPeerCount and maxPeerCount are less than the total number of peers in the channel. This is a known Hyperledger Fabric issue. See https://jira.hyperledger.org/browse/FAB-11889.

4. Click **Add New Collection** and your collection's information is displayed in the private data collection table.
5. If needed, specify other collections.
6. Complete the other fields on the Instantiate Chaincode page as needed.
7. Click **Instantiate**.

View Private Data Collections

(19.1.3 and later versions only) You can view information about a chaincode's private data collections.

After you instantiate a chaincode, you might need to view its private data collections to see how they were defined.

You can’t modify the private data collections for an instantiated chaincode. If you need to change the private data collections, then you need to upgrade the chaincode and specify new private data collections.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.
2. Locate the chaincode that you want to view private data collections for and expand it in the table.
3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.
4. In the Private Data Collections tab, locate the collection that you want to view.

8 Develop Blockchain Applications

Blockchains require smart contracts (chaincode) to update the ledger. In addition, you will also require a client application that utilizes either the Oracle Blockchain Platform REST API or native Hyperledger Fabric SDK to interact with the blockchain directly. There are other operational and administrative tasks to consider, namely the creation of peers and channels and installation of chaincode.

Topics

Before You Develop an Application

Before you write an application, download and use the sample applications, and ensure that you've the correct certificates and privileges to run an application.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See:

- What Are Chaincode Samples?
- Explore Oracle Blockchain Platform Using Samples

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write applications. The Key Concepts and Tutorials sections should be read before you write you own application: Welcome to Hyperledger Fabric.

Prerequisites for Application Development

A user ID and password for the application user must exist in your LDAP server. Depending on the functions in the application, this user must have the following:

- To install and instantiate chaincode:
  - You must have administrative access in order to install or deploy chaincode.
  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer and orderer nodes crypto folders.
  - You must export the admin credentials similarly to how you exported the certificates (from the action menu, select **Export Admin Credential**). This will download a ZIP file containing the signed certificate and keystore files that need to be placed in your application in the peer and orderer nodes crypto folders.
- To run operations against an installed and instantiated chaincode:

  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer node crypto folders.
  - You must export the tlscacerts certificate for the orderer node as described in Export Ordering Service Settings so that it can be placed in your application.
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.
- To run functions against a REST API endpoint:
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.

Use the Hyperledger Fabric SDKs to Develop Applications

Applications use a software development kit (SDK) to access the APIs that permit queries and updates to the ledger. You can install and use the Hyperledger Fabric SDKs to develop applications for Oracle Blockchain Platform.

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. See REST API for Oracle Blockchain Platform.

However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

SDK Versions

- If your Oracle Blockchain Platform founder instances were created using 19.2.3 or 19.3.2, they support V1.4 of the Hyperledger Fabric SDKs.

Installing the Hyperledger Fabric SDK for Node.js

Information about how to use the Fabric SDK for Node.js can be found here: Hyperledger Fabric SDK for Node.js documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Node.js SDK from this tab.
- If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform.

Installing the Hyperledger Fabric SDK for Java

Information about how to use the Fabric SDK for Java can be found here: Hyperledger Fabric SDK for Java documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Java SDK from this tab.

- If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform.

Install a build tool such as Apache Maven.

Structuring your Application

Your Java application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /ordererer
        Contains the certificates required for the application to act 
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act 
on the peer node
    /src
      chaincode.go if installing and deploying chaincode to the 
blockchain
  /java
    pom.xml or other build configuration files
    /resources
      Any resources used by the Java code, including artifacts such as 
the endorsement policy yaml file and blockchain configuration properties
    /src
      Java source files
```

Your Node.js application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act 
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act 
on the peer node
    /src
      chaincode.go if installing and deploying chaincode to the 
blockchain
  /node
    package.json file
    application.js
    /app
      Any javascript files called by the application
      /tools
```

Running the application

You’re now ready to run and test the application. In addition to any status messages returned by your application, you can check the ledger in the Oracle Blockchain Platform console to see your changes:

1. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
2. In the channel’s **Ledger** pane, view the chaincode’s ledger summary.

Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform

There's an incompatibility between an OCI infrastructure component and the Node.js and Java SDKs provided with Fabric. Follow the steps in this topic to correct this problem.

Methods of updating the Fabric SDKs

There are two ways of updating the SDK:

- Using Oracle scripts to download and install the Node.js SDK or Java SDK which will patch the code as it installs.
- Manually as described in this topic.

To use the scripts, on the console’s **Developer Tools** tab, select the **Application Development** pane. The links to download both the Node.js SDK and Java SDK have updates built in which will patch the code as it installs.

- Fabric Java SDK: We’ve created an updated `grpc-netty-1.15.0.jar` file, which is the module referenced by the Java SDK which requires modifications.
- Fabric Node.js SDK: We have created the `npm_bcs_client.sh` script to replace the standard Fabric `npm install` operations that users would perform to download and install the Node.js Fabric client package. The script runs the same npm command, but it also patched the needed component and rebuilds it.

Note that the Go SDK doesn’t have the same incompatibility issue and doesn’t need to be updated.

Manually updating the Fabric Node.js SDK

Do the following to rebuild the `grpc-node` module to connect the peers and orderers with grpcs client (via tls).

1. Install `fabric-client` without executing the grpc module's build script:

   ```
   npm install --ignore-scripts fabric-client
   ```

2. Change the code to avoid an `alpn` error from server side.
   - Change the target code of `node_modules/grpc/deps/grpc/src/core/lib/security/security_connector/security_connector.cc`
   - Change function `ssl_check_peer` to something similar to:

     ```
     static grpc_error* ssl_check_peer(grpc_security_connector* sc,
                                       const char* peer_name, const
     tsi_peer* peer,
     ```

```
                                  grpc_auth_context**
auth_context) {
  /* Check the ALPN. */
  const tsi_peer_property* p =
      tsi_peer_get_property_by_name(peer,
TSI_SSL_ALPN_SELECTED_PROTOCOL);
   if (false) {
    return GRPC_ERROR_CREATE_FROM_STATIC_STRING(
        "Cannot check peer: missing selected ALPN property.");
  }
   if (p != nullptr && !grpc_chttp2_is_alpn_version_supported(p-
>value.data, p->value.length)) {
    return GRPC_ERROR_CREATE_FROM_STATIC_STRING(
        "Cannot check peer: invalid ALPN value.");
  }

  ...
}
```

   - Change the target code of `node_modules/grpc/binding.gyp`.
   - Change the variable `grpc_alpn` to false:

     ```
     'grpc_alpn%': 'false'
     ```

3. Rebuild `grpc`

   ```
   npm rebuild --unsafe-perm --build-from-source
   ```

You can now install any other modules you need and run the project.

Manually updating the Fabric Java SDK

For `fabric-sdk-java`, do the following steps to rebuild the `grpc-netty` package to connect the peers and orderers with grpcs client (via tls). `grpc-netty` is a sub-project of `grpc-java`.

1. Install project dependencies:

   ```
   mvn install
   ```

2. Download grpc-java source code:

   ```
   git clone https://github.com/grpc/grpc-java.git
   ```

3. Change the code to avoid an `alpn` error from the server side.
   - Change the target code of `grpc-java_root/netty/src/main/java/io/grpc/netty/ProtocolNegotiators.java`
   - Change function `userEventTriggered` to something similar to:

     ```
     private static class BufferUntilTlsNegotiatedHandler extends
     AbstractBufferingHandler
           implements ProtocolNegotiator.Handler {
     ```

```
    ...

    @Override
    public void userEventTriggered(ChannelHandlerContext ctx,
Object evt) throws Exception {
      ...
          if (handler.applicationProtocol() == null
              ||
NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
            // Successfully negotiated the protocol.
            logSslEngineDetails(Level.FINER, ctx, "TLS
negotiation succeeded.", null);
      ...
  }
```

4. Build the project to generate the target patched package. Use gradle to build the `grpc-java` project. Or you can just rebuild the `grpc-netty` sub-project in the grpc netty directory `gradle build`.

   After the build is done, you can get the target patched jar package in the directory `grpc-java\netty\build\libs\grpc-netty-1.15.0.jar`.

5. Add the patched package into your Maven local repository.

   Replace official grpc-netty jar package with the patched package in either of the following two ways:

   - Use Maven to install the package by local file:

     ```
     mvn install:install-
     file -Dfile=local_patched_grpc_netty_package_root/grpc-
     netty-1.15.0.jar -DgroupId=io.grpc -DartifactId=grpc-netty -
     Dversion=1.15.0 -Dpackaging=jar
     ```

     You must keep the target `groupid`, `artifactid`, and `version` the same as the package you want to replace.

   - Manually replace your package. Go to the local Maven repository, find the directory where the target package is located, and replace the package with patched package.

6. Run the project.

Use the REST APIs to Develop Applications

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

Any application using the REST APIs requires the following:

- The chaincode name and version.
- The REST server URL and port, and the user ID and password for the REST node.

- Functions to invoke transactions against or query the ledger.

See REST API for Oracle Blockchain Platform for information on the existing operations, including examples and usage syntax.

Structuring your Application

Your REST API application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act 
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act 
on the peer node
    /src
  /REST
    Application script containing REST API calls
```

9

Work With Databases

This topic contains information to help you understand how to query the state database and how to create and configure a rich history database.

Topics:

- Query the State Database
- Create the Rich History Database

Query the State Database

This topic contains information to help you understand how to query the state database.

Topics:

- What's the State Database?
- Supported Rich Query Syntax
- State Database Indexes
- Differences in the Validation of Rich Queries

What's the State Database?

The blockchain ledger’s current state data is stored in the state database.

When you develop Oracle Blockchain Platform chaincodes, you can extract data from the state database by executing rich queries. Oracle Blockchain Platform supports rich queries by using the SQL rich query syntax and the CouchDB find expressions. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

Hyperledger Fabric doesn’t support SQL rich queries. If your Oracle Blockchain Platform network contains Hyperledger Fabric participants, then you need to make sure to do the following:

- If your chaincodes contain SQL rich query syntax, then those chaincodes are installed only on member peers using Oracle Blockchain Platform.
- If a chaincode needs to be installed on Oracle Blockchain Platform and Hyperledger Fabric peers, then use CouchDB syntax in the chaincodes and confirm that the Hyperledger Fabric peers are set up to use CouchDB as their state database repository. Oracle Blockchain Platform can process CouchDB.

How Does Oracle Blockchain Platform Work with Berkeley DB?

Oracle Blockchain Platform uses Oracle Berkeley DB as the state database. Oracle Blockchain Platform creates relational tables in Berkeley DB based on the SQLite extension. This architecture provides a robust and performant way to validate SQL rich queries.

For each channel chaincode, Oracle Blockchain Platform creates a Berkeley DB table. This table stores state information data, and contains at least a key column named `key`, and a value column named `value` or `valueJson`, depending on whether you're using JSON format data.

Column Name	Type	Description
`key`	TEXT	Key column of the state table.
`value`	TEXT	Value column of the state table.
`valueJson`	TEXT	JSON format value column of the state table.

Note that the `valueJson` and `value` columns are mutually-exclusive. So, if the chaincode assigns a JSON value to a key, then the `valueJson` column will hold that value, and the value column will be set to null. If the chaincode assigns a non-JSON value to a key, then the `valueJson` column will be set to null, and the value column will hold the value.

Example of a State Database

These are examples of keys and their values from the Car Dealer sample's state database:

key	value	valueJson
abg1234	null	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
abg1235	null	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
ser1236	null	{"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
bra1238	null	{"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
dtrt10001	null	{"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979

State Database Indexes

The state database can contain a large amount of data. In such cases Oracle Blockchain Platform uses indexes to improve data access.

Default Indexes

When a chaincode is deployed, Oracle Blockchain Platform creates two indexes.

- Key index — Created on the key column.
- Value index — Created on the value column.

Custom Indexes

In some cases, you might need to create custom indexes. You define these indexes using any expression that can be resolved in the context of the state table. Custom indexes created against Berkeley DB rely on the SQLite syntax, but they otherwise follow the same CouchDB implementation provided by Hyperledger Fabric.

Note that you can use custom indexes to dramatically improve the performance of WHERE and ORDER BY statements on large data sets. Because using custom indexes slows down data insertions, you should use them judiciously.

Each custom index is defined as an array of expressions, which support compound indexes, expressed as a JSON document inside one file (note that there's one index per file). You must package this file with the chaincode in a folder named “indexes” in the following directory structure: `statedb/relationaldb/indexes`. See How to add CouchDB indexes during chaincode installation.

Example Custom Indexes

The custom index examples in this section use the Car Dealer sample.

Example 1 —This example indexes the use of the `json_extract` expression in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … ORDER BY json_extract(valueJson, '$.owner')
```

Example 2 — This example indexes the compound use of the two `json_extract` expressions in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.docType')",
"json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

Example 3 — This example creates two indexes: the index described in Example 1 and the index described in Example 2. Note that each JSON structure needs to be included in a separate file. Each file describes a single index: a simple index like Example 1, or a compound index like Example 2.

Index 1: `{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}`

Index 2: `{"indexExpressions": ["json_extract(valueJson, '$owner')", "json_extract(valueJson, '$.docType')"]}`

In the following example, Index 2 is applied to the `AND` expression in the `WHERE` portion of the query, while Index 1 is applied to the `ORDER BY` expression:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto' ORDER BY
json_extract(valueJson, '$.owner')
```

JSON Document Format

The JSON document must be in the following format:

```
{"indexExpressions": [expr1, ..., exprN]}
```

For example:

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

Differences in the Validation of Rich Queries

In some cases, the standard Hyperledger Fabric with CouchDB rich query and the Oracle Berkeley DB rich query behave differently.

In standard Hyperledger Fabric with CouchDB, each key and value pair returned by the query is added to the transaction's read-set and is validated at validation time and without re-executing the query. In Berkeley DB, the returned key and value pair isn’t added to the read-set, but the rich query's result is hashed in a Merkle tree and validated against the re-execution of the query at validation time.

Native Hyperledger Fabric doesn’t provide data protection for rich query. However, Berkeley DB contains functionality that protects and validates the rich query by adding the Merkle tree hash value into the read-set, re-executing the rich query, and at the validation stage re-calculating the Merkle tree value. Note that because validation is more accurate in Oracle Blockchain Platform with Berkeley DB, chaincode invocations are sometimes flagged for more frequent phantom reads.

Supported Rich Query Syntax

Oracle Blockchain Platform supports two types of rich query syntax that you can use to query the state database: SQL rich query and CouchDB rich query.

Topics:

- SQL Rich Query Syntax
- CouchDB Rich Query Syntax

SQL Rich Query Syntax

The Berkeley DB JSON extensions are in the form of SQL functions.

Before You Begin

Note the following information:

- You can only access the channel chaincode (<STATE>) that you’re executing your query from.
- Only the SELECT statement is supported.
- You can’t modify the state database table.
- A rich query expression can have only one SELECT statement.

- The examples in this topic are just a few ways that you can write your rich query. You've access to the usual full SQL syntax to query a SQL database.
- You've access to the JSON1 Extension (SQLite extension). See JSON1 Extension and SQL As Understood by SQLite.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

How to Refer to the State Database in Queries

The state database table name is internally managed by Oracle Blockchain Platform, so you don't need to know the state database's physical name when you write a chaincode.

Instead, you must use the `<STATE>` alias to refer to the table name. For example: `select key, value from <STATE>`.

Note that the `<STATE>` alias is **not** case-sensitive, so you can use either `<state>`, `<STATE>`, or something like `<StAtE>`.

Retrieve All Keys

Use this syntax:

```
SELECT key FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following list of keys:

key

abg1234

abg1235

ser1236

bra1238

dtrt10001

Retrieve All Keys and Values Ordered Alphabetically by Key

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM  <state> ORDER BY
key
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

serialNumber	details
abg1234	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}

serialNumber	details
abg1235	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
bra1238	{"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
dtrt10001	{"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979
ser1236	{"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}

Retrieve All Keys and Values Starting with “abg”

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM <state> WHERE key
LIKE 'abg%'SELECT key, value FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following results:

serialNumber	details
abg1234	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 2020", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"}
abg1235	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 4050", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"}

Retrieve All Keys with Values Containing a Vehicle Part Owned by "Detroit Auto"

Use this syntax:

```
SELECT key FROM <state> WHERE json_extract(valueJson, '$.docType') =
'vehiclePart' AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following list of keys:

key
abg1234
abg1235
ser1236
bra1238

Retrieve Model and Manufacturer for all Cars Owned by "Sam Dealer"

Use this syntax:

```
SELECT json_extract(valueJson, '$.model') AS model,
json_extract(valueJson, '$.manufacturer') AS manufacturer FROM
<state> WHERE json_extract(valueJson, '$.docType') = 'vehicle' AND
json_extract(valueJson, '$.owner') = 'Sam Dealer'
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

model	manufacturer
a coupe	Detroit Auto

CouchDB Rich Query Syntax

Use the information in this topic if you're migrating your chaincodes containing CouchDB syntax to Oracle Blockchain Platform, or if you need to write chaincodes to install on Hyperledger Fabric peers participating in an Oracle Blockchain Platform network.

If you're writing a new chaincode, then Oracle recommends that you use SQL rich queries to take advantage of the performance benefits that Oracle Blockchain Platform with Berkeley DB provides.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

Unsupported Query Parameters and Selector Syntax

Oracle Blockchain Platform doesn't support the `use_index` parameter. If used, Oracle Blockchain Platform ignores this parameter, and it will automatically pick the indexes defined on the StateDB in question.

Parameter	Type	Description
use_index	json	Instructs a query to use a specific index.

Retrieve All Models, Manufacturers, and Owners of Cars, and Order Them by Owner

Use this expression:

```
{
  "fields": ["model", "manufacturer", "owner"],
  "sort": [
    "owner"
   ]
}
```

Retrieve Model and Manufacturer for All Cars Owned by “Sam Dealer”

Use this expression:

```
{
  "fields": ["model", "manufacturer"],
  "selector": {
    "docType"  : "vehicle",
     "owner" : "Sam Dealer"
  }
}
```

Create the Rich History Database

This topic contains information to help you specify an Oracle database connection and choose channels to create the rich history database. You’ll use this database to make analytics reports and visualizations of your ledger’s activities.

Topics:

- What's the Rich History Database?
- Rich History Database Tables and Columns
- Create the Oracle Database Cloud Service Connection String
- Enable and Configure the Rich History Database
- Modify the Connection to the Rich History Database
- Choose the Channels that Write Data to the Rich History Database

What's the Rich History Database?

The rich history database is external to Oracle Blockchain Platform and contains data about the blockchain ledger’s transactions on the channels you select. You use this database to create analytics reports and visualization about your ledger’s activities.

For example, using the rich history database, you could create analytics to learn the average balance of all of the customers in your bank over some time interval, or how long it took to ship merchandise from a wholesaler to a retailer.

Internally, Oracle Blockchain Platform uses the Hyperledger Fabric history database to manage the ledger and present ledger transaction information to you in the console. Only the chaincodes can access this history database, and you can’t expose the Hyperledger Fabric history database as a data source for analytical queries. The rich history database uses an external Oracle database and contains many details about every transaction committed on a channel. This level of data collection makes the rich history database an excellent data source for analytics. For information about the data that the rich history database collects, see Rich History Database Tables and Columns.

You can only use an Oracle database such as Oracle Autonomous Data Warehouse or Oracle Database Cloud Service with Oracle Cloud Infrastructure to create your rich history database. You use the Oracle Blockchain Platform console to provide the connection string and credentials to access and write to the Oracle database. Note that the credentials you provide are the database’s credentials and Oracle Blockchain Platform doesn’t manage them. After you create the connection, you’ll select the channels that contain the ledger data that you want to include in the rich history database. See Enable and Configure the Rich History Database.

You can use any analytics tool, such as Oracle Analytics Cloud or Oracle Data Visualization Cloud Service, to access the rich history database and create analytics reports or data visualizations.

Create the Oracle Database Cloud Service Connection String

You must collect information from the Oracle Database Cloud Service deployed on Oracle Cloud Infrastructure to build the connection string required by the rich history database. You must also enable access to the database through port 1521.

Find and Record Oracle Database Cloud Service Information

The information you need to create a connection to the Oracle Database Cloud Service is available in the Oracle Cloud Infrastructure Console.

1. From the Infrastructure Console, click the navigation menu in the top left corner, and then click **Database**.
2. Locate the database that you want to connect to and record the **Public IP** address.
3. Click the name of the database that you want to connect to and record the values in these fields:
   - **Database Unique Name**
   - **Host Domain Name**
   - **Port**
4. Find a username and password of a database user with permissions to read from this database, and make a note of these. For example, the user SYSTEM.

Enable Database Access Through Port 1521

Add an ingress rule that enables the rich history database to access the database through port 1521.

1. In the Oracle Cloud Infrastructure home page, click the navigation icon and then under **Databases** click **DB Systems**.
2. Click the database that you want to connect to.
3. Click the **Virtual Cloud Network** link.
4. Navigate to the appropriate subnet, and under **Security Lists**, click **Default Security List For <Target Database>**.

   The Security List page is displayed.
5. Click **Edit All Rules**.
6. Add an ingress rule to allow any incoming traffic from the public internet to reach port 1521 on this database node, with the following settings:
   - **SOURCE CIDR**: 0.0.0.0/0
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All

- **DESTINATION PORT RANGE**: 1521
- **Allows**: TCP traffic for ports: 1521

Build the Connection String

After enabling access to the Oracle database, use the information you collected to build the connection string in the Configure Rich History dialog.

Construct the connection string like this: *<publicIP>*:*<portNumber>*/*<database unique name>*.*<host domain name>*

For example, 123.213.85.123:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com

Enable and Configure the Rich History Database

Use the console to provide database connection information and select the channels with the chaincode ledger data that you want to write to the rich history database. By default channels aren't enabled to write data to the rich history database.

Note the following information:

- Each blockchain network member configures its own rich history database.
- You must use an Oracle database. No other database types are supported.

1. Enter connection and credential information for the Oracle database that you want to use to store rich history information.
   a. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.

      The **Configure Rich History** dialog is displayed.
   b. Enter the user name and password required to access the Oracle database.
   c. In the **Connection String** field, enter the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.
      - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
      - If you're using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Cloud Service Connection String.
      - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`
   d. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload the required credential wallet ZIP file. This file contains client credentials and is generated from the Oracle autonomous database.

   e. Click **Save**.

2. Enable rich history on the channels that contain the chaincode data that you want to write to the rich history database.
   a. Go to the console and select the **Channels** tab.
   b. Locate the channel that contains the chaincode data that you want to write to the rich history database. Click its **More Options** button and select **Configure Rich History**.

      The Configure Rich History dialog is displayed.
   c. Click the **Enable Rich History** checkbox. Click **Save**.

Modify the Connection to the Rich History Database

You can change the rich history database's connection information.

1. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.
2. If needed, update the user name and password required to access the Oracle database.
3. If needed, in the **Connection String** field, modify the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.
   - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
   - If you're using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Cloud Service Connection String.
   - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`
4. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload or re-upload the required credential wallet file. This file contains client credentials and is generated from the Oracle autonomous database.
5. Click **Save**.

Choose the Channels that Write Data to the Rich History Database

You can add channels to write chaincode ledger data to the rich history database, or you can stop channels from writing data to the rich history database.

You must specify information to connect to the rich history's database before you can select channels that write to the rich history database. See Enable and Configure the Rich History Database.

1. Go to the console and select the **Channels** tab.
2. To add or remove a channel, locate the channel that you want to modify access for. Click its **More Options** button and select **Configure Rich History**.

   The Configure Rich History dialog is displayed.
3. To add the channel, click the **Enable Rich History** checkbox. To remove the channel, clear the **Enable Rich History** checkbox.
4. Click **Save**.

Rich History Database Tables and Columns

The rich history database contains three tables for each channel: history, state, and latest height. You'll query the history and state tables when you create analytics about your chaincodes' ledger transactions.

History Table

The *<instanceName><channelName>*_hist table contains ledger history. The data in this table tells you the chaincode ID, key used, if the transaction was valid, the value assigned to the key, and so on.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

Column	Datatype
chaincodeId	VARCHAR2 (256)
key	VARCHAR2 (1024)
txnIsValid	NUMBER (1)
value	VARCHAR2 (4000)
valueJson	CLOB
blockNo	NUMBER NOT NULL
txnNo NUMBER	NOT NULL
txnId	VARCHAR2 (128)
txnTimestamp	TIMESTAMP
txnIsDelete	NUMBER (1)

State Table

The *<instanceName><channelName>*_state table contains data values replicated from the state database. You'll query the state table when you create analytics about the state of the ledger.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

Column	Datatype
chaincodeId	VARCHAR2 (256)
key	VARCHAR2 (1024)
value	VARCHAR2 (4000)
valueJson	CLOB
blockNo	NUMBER
txnNo	NUMBER

Latest Height Table

The *<instanceName><channelName>*_last table is used internally by Oracle Blockchain Platform to track the block height recorded in the rich history database. It determines how current the rich history database is and if all of the chaincode transactions were recorded in the rich history database. You can't query this database for analytics.

A

Node Configuration

This topic contains information to help you understand and configure your nodes. Each node type has different configuration options.

Topics:

- CA Node Attributes
- Console Node Attributes
- Orderer Node Attributes
- Peer Node Attributes
- REST Proxy Node Attributes

CA Node Attributes

A certificate authority (CA) node keeps track of identities and certificates on the blockchain network.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-1 CA Node Attributes

Attribute	Description	Default Value
Fabric CA ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can’t modify this ID.	ca
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.

Table A-1 (Cont.) CA Node Attributes

Attribute	Description	Default Value
Max Enrollments	Use this field to determine how many times the CA server allows a password to be used for enrollment on the network. Consider the following options: • -1 — If you use -1, then the server allows a password to be used an unlimited number of times for enrollment. • 0 — Use 0 to disable enrollment. No identity registrations is allowed. • 1 — Use 1 so that an enrollment ID’s password can be used only one time.	-1
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use INFO.	INFO

Console Node Attributes

The console node manages the performance of the console.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-2 Console Node Attributes

Attribute	Description	Default Value
Console ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	console
Local MSP ID	This is the assigned MSP ID for your organization. You can’t modify this ID.	NA
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO

Table A-2 (Cont.) Console Node Attributes

Attribute	Description	Default Value
Request Timeout (ms)	Specify the maximum amount of time in milliseconds that you want the console to attempt to contact the nodes before timing out.	600,000

Orderer Node Attributes

An orderer node collects transactions from peer nodes, bundles them, and submits them to the blockchain ledger. The node’s attributes determine how the node performs and behaves on the network.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-3 Orderer Node — General Attributes

Attribute	Description	Default Value
Orderer ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	orderer<number-partition>
Local MSP ID	This is the assigned MSP ID for your organization. You can’t modify this ID.	NA
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO

Table A-4 Orderer Node — Advanced Attributes — Kafka/Retry tab

Attribute	Description	Default Value
ShortInterval in seconds	How long to establish the Kafka cluster connection for the orderer to read or write to the channel. The system will re-attempt the connection using the ShortInterval limit until it exceeds the ShortTotal limit.	5

Table A-4 (Cont.) Orderer Node — Advanced Attributes — Kafka/Retry tab

Attribute	Description	Default Value
ShortTotal in milliseconds	How long the ShortInterval limit is repeated until the system uses the LongInterval limit to establish the Kafka cluster connection.	10
LongInterval in milliseconds	How long to establish the Kafka cluster connection for the orderer to read or write to the channel. If the system can't successfully connect using the ShortTotal limit, then the system uses the LongInterval to attempt the connection. The system will re-attempt the connection using the LongtInterval limit until it exceeds the LongTotal limit.	5
LongTotal in hours	The number of hours the LongInterval limit is repeated until the Kafka cluster connection fails.	12
NetworkTimeouts/DialTimeout in seconds	The maximum amount of time that the client will wait for the initial connection.	10
NetworkTimeouts/ ReadTimeout in seconds	The maximum amount of time that the client will wait for a response.	10
NetworkTimeouts/ WriteTimeout in seconds	The maximum amount of time that the client will wait for a transmit.	10
Metadata/RetryBackoff in milliseconds	The maximum amount of time to wait before retrying the metadata request to the Kafka cluster.	250
Metadata/RetryMax	The maximum number of metadata request attempts the system makes to the Kafka cluster.	3
Producer/RetryBackoff in milliseconds	The maximum amount of time to wait before retrying to post a message to the Kafka cluster.	100
Producer/RetryMax	The maximum number of attempts that the system makes to post a message to the Kafka cluster.	3

Table A-4 (Cont.) Orderer Node — Advanced Attributes — Kafka/Retry tab

Attribute	Description	Default Value
Consumer/RetryBackoff in seconds	The maximum amount of time to wait before retrying to reconnect the offsets channel or retrying failed offset fetch or commit requests.	2

Peer Node Attributes

A peer node reads, endorses, and writes transactions to the blockchain ledger. The node’s attributes determine how the node performs and behaves on the network.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-5 Peer Node — General Attributes

Attribute	Description	Default Value
Peer ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	peer<number-partition>
Local MSP ID	This is the assigned MSP ID for your organization. You can’t modify this ID.	Specific to your organization.
Role	Specifies if the peer’s role is Member or Admin. In most cases this field displays Member. This role is used by the chaincode’s endorsement policy. The endorsement policy specifies the MSP that must validate the identity of the signer peer and the signer peer’s role. The Admin role is normally assigned in situations where you want to further protect sensitive operations and make sure that those operations are endorsed by specific peers. The peers created with your instance were assigned the Member role.	Member
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.

Table A-5 (Cont.) Peer Node — General Attributes

Attribute	Description	Default Value
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO

Table A-6 Peer Node — Advanced Attributes — Gossip tab

Attribute	Description	Default Value
Bootstrap Peers	Provide the service name address and port that the peer uses to contact other peers during startup. This endpoint must match the endpoints of the peers in the same organization.	NA
Max Block Count to Store	Enter the maximum number of blocks to store in memory.	3
Max Propagation Burst Latency in milliseconds	Enter how many milliseconds between message pushes.	2
Max propagation burst size	Enter the number of messages to be stored until a push remote peer is triggered.	4
Propagate Iterations	Enter the number of times a message is pushed to the peers.	7
Propagate Peer Number	Enter how many peers to send messages to.	8
Pull Interval in seconds	Enter how many seconds between pull phases.	10
Pull Peer Number	Enter the number of peers to pull from.	9
Request State Info Interval in seconds	Enter how often to pull state information messages from the peers.	20
Publish state Info Interval in seconds	Enter how often to send state information messages to the peers.	21
Publish Cert Period in seconds	Enter how many seconds from startup that certificates are included in alive messages.	40
Dial Timeout in seconds	Enter how many seconds before dial times out.	10
Connect Timeout in seconds	Enter how many seconds until the connection times out.	20
Receive Buffer Size	Enter the size of the buffer for received messages.	20

Table A-6 (Cont.) Peer Node — Advanced Attributes — Gossip tab

Attribute	Description	Default Value
Send Buffer Size	Enter the size of the buffer for sending messages.	40
Digest Wait Time in seconds	Enter how many seconds to wait before the pull engine processes incoming digests.	15
Request Wait Time in seconds	Enter how many seconds to wait before the pull engine removes incoming nonce.	10
Response Wait Time in seconds	Enter how many seconds that the pull engine waits before it terminates the pull.	20
Alive Time Interval in seconds	Enter how often to check alive time.	15
Alive Expiration Timeout in seconds	Enter how many seconds to wait before the alive expiration times out.	12
Reconnect Interval in seconds	Enter how many seconds to wait before reconnecting.	9
Skip Block Verification	Click to skip block verification.	Selected

Table A-7 Peer Node — Advanced Attributes — Gossip/Election tab

Attribute	Description	Default Value
Membership Sample Interval in seconds	How often in seconds the peer checks its stability on the network.	3
Leader Alive Threshold in seconds	The number of seconds to elapse before the last declaration message is sent and before the peer determines leader election.	2
Leader Election Duration in seconds	The number of seconds to elapse after the peer sends the propose message and declares itself leader.	5

Table A-7 (Cont.) Peer Node — Advanced Attributes — Gossip/Election tab

Attribute	Description	Default Value
Leader	A channel's leader peer receives blocks and distributes them to the other peers within the cluster. Specify the mode that you want the peer to use to determine a leader. • **OrgLeader** — Select this option to use static leader mode and make the peer the organization leader. If you select this option and then add more peers to the channel, then you must set all peers to **OrgLeader**. • **UseLeaderElection** — Select this option to use dynamic leader election on the channel. Before an active leader is selected for the organization, the system must run the configuration transaction to add the organization to the channel, and then the system updates the new peers with the configuration transaction.	UseLeaderElection

Table A-8 Peer Node — Advanced Attributes — Event Service tab

Attribute	Description	Default Value
Buffer Size	Enter the maximum number of events that the buffer can contain. The system won't send the events that exceed this number.	100
Timeout in milliseconds	Enter in milliseconds the maximum time allowed for the business network to send an event.	1000

Table A-9 Peer Node — Advanced Attributes — Chaincode tab

Attribute	Description	Default Value
Startup timeout in seconds	Enter in seconds the maximum time to wait between when the container starts and the registry responds.	300

Table A-9 (Cont.) Peer Node — Advanced Attributes — Chaincode tab

Attribute	Description	Default Value
Execute timeout in seconds	Enter in seconds the maximum time that a chaincode attempts to execute before timing out.	30
Mode	Displays how the system runs the chaincode. This value is always net.	net
Keepalive in seconds	If you're using a proxy for communication, then enter in seconds the maximum amount of time to keep the connection between a peer and the chaincode alive.	0
Log Level	Specify the log level that you want to use for all loggers in the chaincode container. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO
(New in 19.2.1) Shim Level	Specify the log level that you want to use for the shim logger.	WARNING

REST Proxy Node Attributes

A REST proxy node allows you to query or invoke a chaincode through the RESTful protocol. The node’s attributes determine how the node performs on the network and which channel, chaincode, and peers are used in the transactions performed by the node.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-10 REST Proxy Node Attributes

Attribute	Description	Default Value
REST Proxy Name	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can’t modify this ID.	restproxy<node number>
Proposal Wait Time (ms)	Enter the number of milliseconds that the node waits for completion of the proposal process. If this number is exceeded, then the transaction times out.	60,000

Table A-10 (Cont.) REST Proxy Node Attributes

Attribute	Description	Default Value
Transaction Wait Time (ms)	Enter the number of milliseconds that the node waits for execution after the transaction is submitted. If this number is exceeded, then the transaction times out.	300,000
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use WARNING or ERROR.	INFO

B

Using the Fine-Grained Access Control Library Included in the Marbles Sample

Starting in v1.2, Hyperledger Fabric provided fine-grained access control to many of the management functions. Oracle Blockchain Platform now provides a updated version of the mables sample package on the Developer Tools tab of the console, implementing a library of functions that chaincode developers can use to create access control lists for chaincode functions. It currently only supports the Go language.

Topics

Background

The goal of this sample access control library is to provide the following:

- Provides a mechanism to allow you to control which users can access particular chaincode functions.
- The list of users and their entitlements should be dynamic and shared across chaincodes.
- Provides access control checks so that a chaincode can check the access control list easily.
- At chaincode deployment time, allows you to populate the list of resources and access control lists with your initial members.
- An access control list must be provided to authorize users to perform access control list operations.

Download the Sample

On the **Developer Tools** tab, open the **Samples** pane. Click the download link under **Marbles with Fine-Grained ACLs**. This package contains three sub-packages:

- `Fine-GrainedAccessControlLibrary.zip`:
  The fine-grained access control library. It contains functions in Go which can be used by chaincode developers to create access control lists for chaincode functions.
- `fgACL_MarbleSampleCC.zip`:
  The marbles sample with access control lists implemented. It includes a variety of functions to let you examine how to work with fine-grained access control lists, groups and resources to restrict functions to certain users/identities.
- `fgACL-NodeJSCode.zip`:

Node.js scripts which use the Node.js SDK to run the sample. `registerEnrollUser.js` can be used to register new users with the Blockchain Platform. `invokeQueryCC.js` can be used to run transactions against a Blockchain Platform instance.

Terminology and Acronyms

Term	Description
Identity	An X509 certificate representing the identity of either the caller or the specific identity the chaincode wants to check.
Identity Pattern	A pattern that matches one or more identities. The following patterns are suggested: • X.509 Subject Common Name – CN • X.509 Subject Organizational Unit – OU • X.509 Subject Organization – O • Group as defined in this library – GRP • Attribute – ATTR The format for a pattern is essentially just a string with a prefix. For example, to define a pattern that matches any identity in organization "example.com", the pattern would be "%O%example.com".
Resource	The name of anything the chaincode wants to control access to. To this library it is just a named arbitrary string contained in a flat namespace. The semantics of the name are completely up to the chaincode.
Group	A group of identity patterns.
ACL	Access Control List: a named entity that has a list of identity patterns, a list of types of access such as "READ", "CREATE", "INVOKE", "FORWARD", or anything the chaincode wants to use. This library will use access types of CREATE, READ, UPDATE, and DELETE (standard CRUD operations) to maintain its information. Other than those four as they relate to the items in this library, they are just strings with no implied semantics. An application may decide to use accesses of "A", "B", and "CUSTOM".

Fine-Grained Access Control Library Functions

The library package provides the following functions for Resources, Groups and ACLs as well as global functions.

Global Functions

Function	Description
Initialization(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (error) (error)	When the chaincode is instantiated, the Initialization function is called. That function will initialize the world state with some built in access control lists. These built in lists are used to bootstrap the environment. So there needs to be access control on who can create resources, groups, and ACLs. If the identify is nil, then use the caller's identify. After the bootstrap is done, the following entities are created: • A resource named ".Resources". A corresponding ACL named ".Resources.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access. • A group named ".Groups". A corresponding ACL named ".Groups.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access. • An ACL named ".ACLs". A corresponding ACL control list named ".ACLs.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access.
NewGroupManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*GroupManager, error)	Get the group manager that's used for all group related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.
NewACLManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ACLManager, error)	Get the ACL manager that's used for all ACL related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.
NewResourceManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ResourceManager, error)	Get the resource manager that's used for all resource related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.

Access Control List (ACL) Functions

Definition of ACL structure:

```
type ACL struct {
  Name string
  Description string
  Accesses []string  // CREATE, READ, UPDATE, and DELETE, or whatever 
the end-user defined
  Patterns []string    // identities
  Allowed bool          // true means allows access.
  BindACLs []string  // The list of ACL , control who can call the APIs 
of this struct
}
```

- **Accesses:** The Accesses string is a list of comma-separated arbitrary access names and completely up to the application except for four: CREATE, READ, UPDATE, and DELETE. These access values are used in maintaining the fine grained access control. Applications can use their own access strings such as `"register"`, `"invoke"`, or `"query"`, or even such things as access to field names such as `"owner"`, `"quantity"`, and so on.
- **Allowed:** Allowed determines whether identities that match a pattern are allowed access (true) or prohibited access (false). You could have an access control list that indicates Bob has access to `"CREATE"`, and another one that indicates group Oracle (of which Bob is a member) is prohibited from `"CREATE"`. Whether Bob has access or not depends upon the order of the access control lists associated with the entity in question.
- **BindACLs:** The BindACLs parameter will form the initial access control list.

ACL functions:

Function	Description
Create(acl ACL, identity *x509.Certificate) (error)	Creates a new ACL. Duplicate named ACL are not allowed. To create a new ACL, the identity needs to have CREATE access to the bootstrap resource named ".ACLs". If identity is nil, the default identity specified in newACLManager() is used.
Get(aclName string, identity *x509.Certificate) (ACL, error)	Get a named ACL. The identity must have READ access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
Delete(aclName string, identity *x509.Certificate) (error)	Delete a specified ACL. The identity must have DELETE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.

Function	Description
Update(acl ACL, identity *x509.Certificate) (error)	Update an ACL. The identity must have UPDATE access to the named resource, and the named ACL must exist. If identity is nil, the default identity specified in NewACLManager() is used.
AddPattern(aclName string, pattern string, identity *x509.Certificate) (error)	Adds a new identity pattern to the named ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
RemovePattern(aclName string, pattern string, identity *X509Certificate) (error)	Removes the identity pattern from the ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
AddAccess(aclname string, access string, identity *X509Certificate) (error)	Adds a new access to the named ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
RemoveAccess(aclName string, access string, identity *X509Certificate) (error)	Removes the access from the ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
UpdateDescription(aclName string, newDescription string, identity *X509Certificate) (error)	Update the description. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
AddBeforeACL(aclName string, beforeName string, newBindACL string, identity *X509Certificate) (error)	Adds a bind ACL before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.
AddAfterACL(aclName string, afterName string, newBindACL string, identity *X509Certificate) (error)	Adds a bind ACL after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.
RemoveBindACL(aclName string, removeName string, identity *X509Certificate) (error)	Removes the removeName ACL from the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.

Function	Description
GetAll(identity *x509.Certificate) ([]ACL, error)	Get all the ACLs. The identity must have READ access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.

Group Functions

Definition of `Group` structure:

```
type Group struct {
    Name string
    Description string
    Members []string     // identity patterns, except GRP.
    BindACLs []string    // The list of ACLs, controls who can access
this group.
}
```

Definition of `GroupManager` functions:

Function	Description
Create(group Group, identity *x509.Certificate) (error)	Create a new group. The identity must have CREATE access to bootstrap group ".Group". If identity is nil, the default identity specified in NewGroupManager() is used.
Get(groupName string, identity *x509.Certificate) (Group, error)	Get a specified group. The identity must have READ access to this group. If identity is nil, the default identity specified in NewGroupManager() is used.
Delete(groupName string, identity *x509.Certificate) (error)	Delete a specified group. The identity must have DELETE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
AddMembers(groupName string, member []string, identity *x509.Certificate) (error)	Add one or more members into the group. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
RemoveMembers(groupName string, member []string, identity *x509.Certificate) (error)	Remove one or more member from a group. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
UpdateDescription(groupName string, newDes string, identity *x509.Certificate) (error)	Update the description. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.

Function	Description
AddBeforeACL(groupName string, beforeName string, aclName string, identity *x509.Certificate) (error)	Adds an bind ACL to the group before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource. The identity must have UPDATE access to the named group. If identity is nil, the default identity specified in NewGroupManager () is used.
AddAfterACL(groupName string, afterName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the group after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the group. The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used.
RemoveBindACL(groupName string, aclName string, identity *x509.Certificate) (error)	Removes the named ACL from the bind ACL list of the named group. The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used.
GetAll(identity *x509.Certificate) ([]Group, error)	Get all groups. The identity must have READ access to these groups. If identity is nil, the default identity specified in NewGroupManager () is used.

Resource Functions

Definition of `Resource` structure:

```
type Resource struct {
     Name string
     Description string
     BindACLs []string      // The name list of ACL, controls who can
access this resource
}
```

Resource Functions:

Fuction	Description
Create(resource Resource, identity *x509.Certificate) (error)	Create a new resource. Duplicate named resources are not allowed. The identity needs to have CREATE access to the bootstrap resource named ".Resources" If identity is null, the default identity specified in NewResourceManager() is used.

Fuction	Description
Get(resName string, identity *x509.Certificate) (Resource, error)	Get a specified resource. The identity must have READ access to the resource. If identity is null, the default identity specified in NewResourceManager() is used.
Delete(resName string, identity *x509.Certificate) (error)	Delete a named resource. The identity must have DELETE access to the named resource. If identity is null, the default identity specified in NewResourceManager() is used.
UpdateDescription(resourceName string, newDes string, identity *x509.Certificate) (error)	Update the description. The identity must have UPDATE access to this resource. If identity is nil, the default identity specified in NewResourceManager() is used.
AddBeforeACL(resourceName string, beforeName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the resource before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource. The identity must have UPDATE access to the named resource. If identity is nil, the default identity specified in NewResourceManager() is used.
AddAfterACL(resourceName string, afterName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the resource after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the resource. The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
RemoveBindACL(resourceName string, aclName string, identity *x509.Certificate) (error)	Removes the named ACL from the bind ACL list of the named resource. The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
CheckAccess(resName string, access string, identity *x509.Certificate) (bool, error)	Check whether the current user has the specified access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
GetAll(identity *x509.Certificate) ([]Resource, error)	Get all resources. The identity must have READ access to these resources. If identity is nil, the default identity specified in NewResourceManager() is used.

Example Walkthough Using the Fine-Grained Access Control Library

This topic provides some examples of how this library and chaincode can be used. These all assuming `Init()` has been called to create the bootstrap entities and the caller of `Init()` and `invoke()` is "`%CN%frank.thomas@example.com`". The normal flow in an application will be to create some initial access control lists that will be used to grant or deny access to the other entities.

Initialization

Call `Initialization()` to create bootstrap entities when instantiating chaincode. For example:

```
import "chaincodeACL"
func (t \*SimpleChaincode) Init(nil, stub shim.ChaincodeStubInterface)
pb.Response
{
         err := chaincodeACL.Initialization(stub)
}
```

Create a new ACL

```
import "chaincodeACL"
...
{

**ACLMgr**  := chaincodeACL.NewACLManager(nil, stub) // Not specify
identity, use caller's identity as default.

// Define a new ACL
**newACL**  := chaincodeACL.ACL{

    "AllowAdmins",   // ACL name
    "Allow admins full access",  // Description
    []string{"CREATE","READ","UPDATE","DELETE"},    // Accesses allowed
or not
    true, // Allowed

[]string{"%CN%bob.dole@example.com","%OU%example.com,"%GRP%admins"}, //
Initial identity patterns
    ".ACLs.acl", // Start with bootstrap ACL

}

// Add this ACL with default identity (caller's identify here)
err :=  **ACLMgr**.Create( **newACL** , nil)

}
```

Now that we have a new ACL, we can use that to modify who can perform certain operations. So we'll first add this new ACL to the bootstrap group `.Groups` to allow any admin to create a group.

Add an ACL to a group

```
import "chaincodeACL"
…
{

  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub) // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.AddAfterACL(

    ".Groups",     // Bootstrap group name
    ".Groups.ACL", // Which ACL to add after
    "AllowAdmins", // The new ACL to add
    nil            // with default identity that's frank.thomas

)

}
```

This adds the `AllowAdmins` ACL to the bootstrap group `.Groups` after the initial bootstrap ACL. Thus this ensures that Frank Thomas can still perform operations on `.Groups` as the ACL granting him permission is first in the list. But now anyone that matches the `AllowAdmins` ACL can perform CREATE, READ, UPDATE, or DELETE operations (they can now create new groups).

Create a new group

Admins can now create a new group:

```
import "chaincodeACL"
...
{

...
  // Define a new group.
  **newGroup**  := chaincodeACL.Group{

      "AdminGrp",   // Name of the group
      "Administrators of the app",   // Description of the group

{"%CN%jill.muller@example.com","%CN%ivan.novak@example.com","%ATTR%role=
admin"},
      []string{"AllowAdmins"},   // The ACL for the group

    }

  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub)   // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.Create( **newGroup** ,
bob\_garcia\_certificate)   // Using a specific certificate
...
}
```

This call is using an explicit identity - that of Bob Garcia (using his certificate) - to try and create a new group. Since Bob Garcia matches a pattern in the `AllowAdmins` ACL and members of that ACL can perform CREATE operations on the bootstrap group `.Groups`, this call will succeed. Had Jim Silva - who was not in organization unit `example.com` nor in the group `AdminGrp` (which still doesn't exist) - had his certificate passed as the last argument, the call would fail as he doesn't have the appropriate permissions. This call will create a new group called "`AdminGrp`" with initial members of the group being jill.muller@example.com and ivan.novak@example.com or anyone with the attribute (ABAC) role=admin.

Create a new resource

```
import "chaincodeACL"
...
{

  ...
  **newResource**  :=  **chaincodeACL**.Resource{

      "transferMarble", // Name of resource to create

      "The transferMarble chaincode function", // Description of the
resource

      []string{"AllowAdmins"}, // Single ACL for now allowing admins

  }

  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil,
stub)  // Not specify identity, use caller's identity as default.
  err :=  **resourceMgr**.Create(resourceMgr, nil)   // Using caller's
certificate

  ...
}
```

This would create a new resource named `transferMarble` that the application might use to control access to the `transferMarble` chaincode function. The access is currently limited by the `AllowAdmins` access control list.

Check access for a resource

We can use this new resource in our chaincode to only allow admins to transfer a marble by modifying the `invoke()` method of the Marbles chaincode as follows:

```
import "chaincodeACL"
…
func (t \*SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {
  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil,
stub)   // Not specify identity, use caller's identity as default.

  function, args := stub.GetFunctionAndParameters()

  fmt.Println("invoke is running " + function)        // Handle
different functions

  if function == "initMarble" {   //create a new marble

      return t.initMarble(stub, args)}

  else if function == " **transferMarble**" { //change owner of a
specific marble

    **allowed** , err : =  **resourceMgr**. **CheckAccess**
("transferMarble", "UPDATE", nil)
    if  **allowed**  == true {

      return t.transferMarble(stub, args)

    else {

      return NOACCESS

    }

    } else if function == "transferMarblesBasedOnColor" { //transfer
all marbles of a certain color
    …

    }

}
```

Fine-Grained Access Control Marbles Sample

The marbles chaincode application lets you create assets (marbles) with unique attributes (name, size, color and owner) and trade these assets with fellow participants in a blockchain network.

This sample application includes a variety of functions to let you examine how to work with access control lists and groups to restrict functions to certain users.

- Overview of the Sample
- Pre-requisites and Setup
- Implement the Fine-Grained Access Control Marble Sample
- Testing the Access Control
- Sample Files Reference

Overview of the Sample

The test scenario included in the sample contains the following restrictions in order to manage assets:

- Bulk transfer of red marbles is only allowed by identities having the "`redMarblesTransferPermission`" Fabric attribute.
- Bulk transfer of blue marbles is only allowed by identities having the "`blueMarblesTransferPermission`" Fabric attribute.
- Deletion of marbles is only allowed to identities with "`deleteMarblePermission`" Fabric attribute.

These restrictions are enforced by implementing the following library methods in the `fgMarbles_chaincode.go` chaincode:

- Create a fine-grained ACL group named `bulkMarblesTransferGroup`. This group will define all the identities which can transfer marbles based on color (bulk transfers):

  ```
  createGroup(stub, []string{" bulkMarblesTransferGroup",
  "List of Identities allowed to Transfer Marbles in Bulk",
  "%ATTR%redMarblesTransferPermission=true,
  %ATTR%blueMarblesTransferPermission=true", ".ACLs"})
  ```

- Create a fine-grained ACL named `redMarblesAcl` which provides bulk transfer of red marbles access to `bulkMarblesTransferGroup`:

  ```
  createACL(stub, []string{"redMarblesAcl",
  "ACL to control who can transfer red marbles in bulk",
  "redMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL named `blueMarblesAcl` which provides bulk transfer of blue marbles access to `bulkMarblesTransferGroup`:

  ```
  createACL(stub, []string{"blueMarblesAcl",
  "ACL to control who can transfer blue marbles in bulk",
  "blueMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL named `deleteMarbleAcl` to restrict marble deletion based on "`canDeleteMarble=true`" Fabric attribute:

  ```
  createACL(stub, []string{"deleteMarbleAcl",
  "ACL to control who can Delete a Marble",
  "deleteMarblePermission", "%ATTR%deleteMarblePermission=true",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL resource named `marble`, operations on which are controlled using the various ACLs we created:

  ```
  createResource(stub, []string{"marble",
  "System marble resource",
  "deleteMarbleAcl,blueMarblesAcl,redMarblesAcl,.ACLs"})
  ```

Pre-requisites and Setup

In order to run the fine-grained access control version of the marbles sample, complete these steps:

1. Download the fine-grained access control version of the marbles sample. On the **Developer Tools** tab, open the **Samples** pane, and then click the download link under **Marbles with Fine-Grained ACLs**. Unzip this package - it contains zips of the marbles sample (`fgACL_MarbleSampleCC.zip`), Node.js files to run the sample (`fgACL-NodeJSCode.zip`), and the fine-grained access control library (`Fine-GrainedAccessControlLibrary.zip`).
2. Generate the chaincode package that will be deployed to Blockchain Platform:
   - Install govendor:

     ```
     go get -u github.com/kardianos/govendor
     ```

   - Unzip the contents of `fgACL_MarbleSampleCC.zip` to the `fgACL_MarbleSampleCC` directory. The contents of the `fgACL_MarbleSampleCC` directory would be: `fgACL_Operations.go`, `fgGroups_Operations.go`, `fgMarbles_chaincode.go`, `fgResource_Operations.go` and the `vendor` directory.
   - From a command line, go to the `fgACL_MarbleSampleCC` directory, and run `govendor sync`. This will download the required dependency (`github.com/op/go-logging`) and add it to the `vendor` directory.
   - Zip all the contents (the four Go files and the `vendor` directory) of the `fgACL_MarbleSampleCC` directory. Your chaincode is ready to be deployed to Blockchain Platform.
3. Install and instantiate the updated sample chaincode package (`fgACL_MarbleSampleCC.zip`) as described in Use Quick Deployment.
4. On the **Developer Tools** tab, open the **Application Development** pane, and then follow the instructions to download the Node.js SDK.
5. On the **Developer Tools** tab, open the **Application Development** pane, and then click **Download the development package**.
   a. Unzip the development package into the same folder with the Node.js files downloaded with the sample.
   b. In the `network.yaml` file, look for the `certificateAuthorities` entry and its `registrar` entry. The administrator's password is masked (converted to `***`) in the `network.yaml` when downloaded. It should be replaced with the administrator's clear text password when running this sample.
6. Register a new identity with your Blockchain Platform instance:

a. Create a new user in IDCS (referred to as `<NewIdentity>` in the following steps) in the IDCS mapped to your tenancy.
b. Give this user the `CA_User` application role for your instance.

Implement the Fine-Grained Access Control Marble Sample

The following steps will enroll your new user and implement the ACL restrictions using the provided Node.js scripts.

1. **Enroll the new user:**

   ```
   node registerEnrollUser.js <NewIdentity> <Password>
   ```

2. **Initialization:** Initialize the access control lists.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> ACLInitialization
   ```

3. **Create the access control lists, groups, and resources:** This creates the ACL resources described in the overview.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> createFineGrainedAclSampleResources
   ```

4. **Create your test marble resources:** This creates several test marble assets - blue1 and blue2 owned by tom, red1 and red2 owned by jerry, and green1 and green2 owned by spike.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> createTestMarbles
   ```

Testing the Access Control

In order to test that our access control lists are only allowing the correct users to perform each function, we'll run through some sample scenarios.

1. **Transfer a marble:** We're transferring marble `blue1` from tom to jerry. Since there are no restrictions on who can transfer a single marble, this should complete successfully.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarble blue1 jerry
   ```

2. **Transfer a marble as the administrative user:** We're transferring marble `blue1` from jerry to spike. Since there are no restrictions on who can transfer a single marble, this should also complete successfully.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarble blue1 spike
   ```

3. **Get history:** Now we'll query the history of the marble named `blue1`. It should return that it was transferred first to jerry then to spike.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> getHistoryForMarble blue1
   ```

4. **Transfer all red marbles:** The `redMarblesAcl` ACL should allow this transfer because the newly registered identity has the required `"redMarblesTransferPermission=true"` Fabric attribute, so the two red marbles should be transferred to tom.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red tom
   ```

5. **Transfer all red marbles as the administrative user:** The administrative identity doesn't have the `"redMarblesTransferPermission=true"` Fabric attribute, so the `redMarblesAcl` ACL should block this transfer.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red jerry
   ```

6. **Transfer all green marbles:** By default, only explicitly defined access is allowed. Because there isn't an ACL which allows for bulk transfer of green marbles, this should fail.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor green tom
   ```

7. **Delete a marble:** The `deleteMarbleAcl` ACL allows this deletion because the newly registered identity has the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> delete green1
   ```

8. **Delete a marble as the administrative user:** The `deleteMarbleAcl` ACL should prevent this deletion because the administrative identity doesn't have the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js < AdminIdentity > <Password> <ChannelName>
   <ChaincodeName> delete green2
   ```

Sample Files Reference

These tables list the methods available in the chaincode and application files included with the sample.

fgMarbles_chaincode.go

Function	Description
`initMarble`	Create a new marble
`transferMarble`	Transfer a marble from one owner to another based on name

Function	Description
`createTestMarbles`	Calls `initMarble` to create new sample marbles for testing purposes
`createFineGrainedAclSampleResources`	Creates the fine-grained access control list (ACL), groups, and resources required by our test scenario
`transferMarblesBasedOnColor`	Transfers multiple marbles of a certain color to another owner
`delete`	Delete a marble
`readMarble`	Returns all attributes of a marble based on name
`getHistoryForMarble`	Returns a history of values for a marble

fgACL_Operations.go

Methods	Parameters	Description
`getACL`	• `name`	Get a named ACL or read all ACLs. The user invoking the method must have READ access to the named ACL.
`createACL`	• `name` • `description` • `accesses` • `patterns` • `allowed` • `BindACLs` • `Identity_Certificate`	To create a new ACL, the user invoking the method needs to have CREATE access to the bootstrap resource named `".ACLs"`. Duplicate named ACLs are not allowed
`deleteACL`	• `name`	The user invoking the method must have DELETE access to the named ACL.
`updateACL`	• `name` • `description` • `accesses` • `patterns` • `allowed` • `BindACLs`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addAfterACL`	• `aclName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addBeforeACL`	• `aclName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addPatternToACL`	• `aclName` • `BindPattern`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.

Methods	Parameters	Description
`removePatternFromACL`	• `aclName` • `BindPattern`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`updateDescription`	• `aclName` • `newDesc`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`removeBindACL`	• `aclName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addAccess`	• `aclName` • `accessName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`removeAccess`	• `aclName` • `accessName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`ACLInitialization`	• none	This function is used to initialize the fine-grained ACL support.

fgGroups_Operations.go

Methods	Parameters	Description
`getGroup`	• `name`	If name=`"GetAll"`, it returns all the groups the identity has access to. Otherwise, it returns the individual group details (if accessible) based on name. The user invoking the method must have READ access to this group.
`createGroup`	• `name` • `description` • `patterns` • `bindACLs`	Returns `success` or `error`. The user invoking the method must have CREATE access to bootstrap group `". Group"`
`deleteGroup`	• `name`	The user invoking the method must have DELETE access to this group.
`addAfterGroup`	• `groupName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this group.
`addBeforeGroup`	• `groupName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this group.

Methods	Parameters	Description
`updateDescriptionForGroup`	• `groupName` • `newDesc`	The user invoking the method must have UPDATE access to this group.
`removeBindAclFromGroup`	• `groupName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to this group.
`addMembersToGroup`	• `groupName` • `pattern`	The user invoking the method must have UPDATE access to this group.
`removeMembersFromGroup`	• `groupName` • `pattern`	The user invoking the method must have UPDATE access to this group.

fgResource_Operations.go

Methods	Parameters	Description
`createResource`	• `name` • `description` • `bindACLs`	The user invoking the method needs to have CREATE access to the bootstrap resource named `". Resources"`. Duplicate named resources are not allowed.
`getResource`	• `name`	The user invoking the method must have READ access to the resource
`deleteResource`	• `name`	The user invoking the method must have DELETE access to the named resource
`addAfterACLInResource`	• `ResourceName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this resource
`addBeforeACLInResource`	• `ResourceName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this resource
`updateDescriptionInResource`	• `ResourceName` • `newDesc`	The user invoking the method must have UPDATE access to this resource
`removeBindACLInResource`	• `ResourceName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to this resource
`checkResourceAccess`	• `ResourceName` • `access`	Checks whether the current user invoking the method has the specified access to the named resource.