Hyperledger Composer Access Control Language

Hyperledger Composer includes an access control language (ACL) that provides declarative access control over the elements of the domain model. By defining ACL rules you can determine which users/roles are permitted to create, read, update or delete elements in a business network's domain model.

Network Access Control

Hyperledger Composer differentiates between access control for resources within a business network (business access control) and access control for network administrative changes (network access control). Business access control and network access control are both defined in the access control file (.acl) for a business network.

Network access control uses the system namespace, which is implicitly extended by all resources in a business network; and grants or denies access to specific actions as defined below, and is intended to allow for more nuanced access to certain network-level operations.

What does network access control allow or disallow?

Network access control affects the following CLI commands:

**Composer Network**

**composer network deploy**

Network access is required to use the CREATE operation for registries and networks.

**composer network download**

Network access is required to use the READ operation for registries and networks.

**composer network list**

Network access is required to use the READ operation for registries and networks.

**composer network loglevel**

Network access is required to use the UPDATE operation for networks.

**composer network ping**

Network access is required to use the READ operation on registries and networks.

**composer network undeploy**

Network access is required to use the DELETE operation on registries and networks.

**composer network update**

Network access is required to use the UPDATE or CREATE operation on registries, or the UPDATE operation on networks.

**Composer Identity**

**composer identity import**

Network access is required to use the UPDATE operation on identity registries or the CREATE operation on identities.

**composer identity issue**

Network access is required to use the UPDATE operation on identity registries or the CREATE operation on identities.

**composer identity revoke**

Network access is required to use the UPDATE operation on identity registries or the DELETE operation on identities.

**Composer Participant**

**composer participant add**

Network access is required to use the CREATE operation on participants or the UPDATE operation on participant registries.

Granting network access control

Network access is granted using the system namespace. The system namespace is always org.hyperledger.composer.system.Network for network access, and org.hyperledger.composer.system for all access. The following access control rules gives the **networkControl** participant the authority to use all operations with network commands.

Copy

rule networkControlPermission {

description: "networkControl can access network commands"

participant: "org.acme.vehicle.auction.networkControl"

operation: ALL

resource: "org.hyperledger.composer.system.Network"

action: ALLOW

}

The following access control rule will give all participants access to all operations and commands in the business network, including network access and business access.

Copy

rule AllAccess {

description: "AllAccess - grant everything to everybody"

participant: "org.hyperledger.composer.system.Participant"

operation: ALL

resource: "org.hyperledger.composer.system.\*\*"

action: ALLOW

}

Evaluation of Access Control Rules

Access control for a business network is defined by an ordered set of ACL rules. The rules are evaluated in order, and the first rule whose condition matches determines whether access is granted or denied. If no rule match then access is **denied**.

ACL rules are defined in a file called permissions.acl in the root of the business network. If this file is missing from the business network then all access is **permitted**.

Access Control Rule Grammar

There are two types of ACL rules: simple ACL rules and conditional ACL rules. Simple rules are used to control access to a namespace, asset or property of an asset by a participant type or participant instance.

For example, the rule below states that any instance of the org.example.SampleParticipant type can perform ALL operations on all instances of org.example.SampleAsset.

Copy

rule SimpleRule {

description: "Description of the ACL rule"

participant: "org.example.SampleParticipant"

operation: ALL

resource: "org.example.SampleAsset"

action: ALLOW

}

Conditional ACL rules introduce variable bindings for the participant and the resource being accessed, and a Boolean JavaScript expression, which, when true, can either ALLOW or DENY access to the resource by the participant.

For example, the rule below states that any instance of the org.example.SampleParticipant type can perform ALL operations on all instances of org.example.SampleAsset IF the participant is the owner of the asset.

Copy

rule SampleConditionalRule {

description: "Description of the ACL rule"

participant(m): "org.example.SampleParticipant"

operation: ALL

resource(v): "org.example.SampleAsset"

condition: (v.owner.getIdentifier() == m.getIdentifier())

action: ALLOW

}

Conditional ACL rules can also specify an optional transaction clause. When the transaction clause is specified, the ACL rule only allows access to the resource by the participant if the participant submitted a transaction, and that transaction is of the specified type.

For example, the rule below states that any instance of the org.example.SampleParticipant type can perform ALL operations on all instances of org.example.SampleAsset IF the participant is the owner of the asset AND the participant submitted a transaction of the org.example.SampleTransaction type to perform the operation.

Copy

rule SampleConditionalRuleWithTransaction {

description: "Description of the ACL rule"

participant(m): "org.example.SampleParticipant"

operation: READ, CREATE, UPDATE

resource(v): "org.example.SampleAsset"

transaction(tx): "org.example.SampleTransaction"

condition: (v.owner.getIdentifier() == m.getIdentifier())

action: ALLOW

}

Multiple ACL rules may be defined that conceptually define a decision table. The actions of the decision tree define access control decisions (ALLOW or DENY). If the decision table fails to match then by default access is denied.

**Resource** defines the things that the ACL rule applies to. This can be a class, all classes within a namespace, or all classes under a namespace. It can also be an instance of a class.

Resource Examples:

* Namespace: org.example.\*
* Namespace (recursive): org.example.\*\*
* Class in namespace: org.example.Car
* Instance of a class: org.example.Car#ABC123

**Operation** identifies the action that the rule governs. Four actions are supported: CREATE, READ, UPDATE, and DELETE. You can use ALL to specify that the rule governs all supported actions. Alternatively, you can use a comma separated list to specify that the rule governs a set of supported actions.

**Participant** defines the person or entity that has submitted a transaction for processing. If a Participant is specified they must exist in the Participant Registry. The PARTICIPANT may optionally be bound to a variable for use in a PREDICATE. The special value 'ANY' may be used to denote that participant type checking is not enforced for a rule.

**Transaction** defines the transaction that the participant must have submitted in order to perform the specified operation against the specified resource. If this clause is specified, and the participant did not submit a transaction of this type - for example, they are using the CRUD APIs - then the ACL rule does not allow access.

**Condition** is a Boolean JavaScript expression over bound variables. Any JavaScript expression that is legal with the an if(...) expression may be used here. JavaScript expressions used for the condition of an ACL rule can refer to JavaScript utility functions in a script file. This allows a user to easily implement complex access control logic, and re-use the same access control logic functions across multiple ACL rules.

**Action** identifies the action of the rule. It must be one of: ALLOW, DENY.

Examples

Example ACL rules (in evaluation order):

Copy

rule R1 {

description: "Fred can DELETE the car ABC123"

participant: "org.example.Driver#Fred"

operation: DELETE

resource: "org.example.Car#ABC123"

action: ALLOW

}

rule R2 {

description: "regulator with ID Bill can not update a Car if they own it"

participant(r): "org.example.Regulator#Bill"

operation: UPDATE

resource(c): "org.example.Car"

condition: (c.owner == r)

action: DENY

}

rule R3 {

description: "regulators can perform all operations on Cars"

participant: "org.example.Regulator"

operation: ALL

resource: "org.example.Car"

action: ALLOW

}

rule R4 {

description: "Everyone can read all resources in the org.example namespace"

participant: "ANY"

operation: READ

resource: "org.example.\*"

action: ALLOW

}

rule R5 {

description: "Everyone can read all resources under the org.example namespace"

participant: "ANY"

operation: READ

resource: "org.example.\*\*"

action: ALLOW

}

Rules are evaluated from top (most specific) to bottom (least specific). As soon as the Participant, Operation and Resource match for a rule then subsequent rules are not evaluated.

This ordering makes the decision table faster to scan for both humans and computers. If no ACL rule fires then the access control decision is DENY.