ConnectionChain

Scenario Script Reference

Fujitsu Limited.

2024/1

Table of Contents

Introduction

Position of this book

This book describes the functionality of ConnectionChain (*1 also described as CC), a security technology that securely links different blockchains, and the API specification of the ConnectionChain. It is written for those who plan or develop applications and services using ConnectionChain. You should also have a basic understanding of the Internet and Web APIs before reading this book.

(*1) Fujitsu Press Release: "Fujitsu Develops Security Technology to Safely Connect Blockchains"

Technology Introduction Page: ConnectionChain

Structure of this book

This manual is organized as follows.

Chapter	Contents
1. Function of ConnectionChain	Provides an overview of ConnectionChain, key features, and the flow of using the API. See if you want to get an overview of the ConnectionChain.
2. Interface (API Specification)	Describes the ConnectionChain API specification. Refer to this when you want to try to use the API of the ConnectionChain or develop an application.

1. Function of ConnectionChain

1.1 Functional Overview

ConnectionChain is a technology that allows securely linking different blockchains. Two or more independently operated blockchain systems (For example, blockchain for money flow management, commercial flow management) are connected through cooperative nodes. The central blockchain can then act as a Hub to manage the logic and execution trail of the collaboration (see Figure 1).

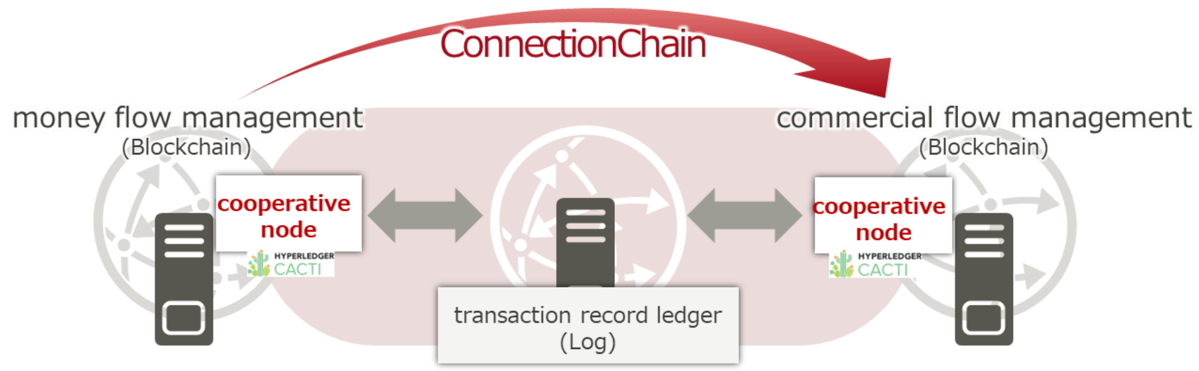


Figure 1: ConnectionChain Overview

The cooperative node in the figure is a connector that abstracts blockchain operations of various blockchain bases, and can be connected to various blockchain bases (*2). The source code for the cooperative node has been published (*3) in Hyperledger Cacti, a blockchain integration project run by the Linux Foundation.

Hyperledger Cacti site: https://www.hyperledger.org/use/cacti

Source code (GitHub): https://github.com/hyperledger/cacti

(*2) Fujitsu Research Portal (Research Portal) supports connectivity to the Data-e-TRUST and Ethereum Sepolia test net.

(*3) The cooperative node for Data-TRUST (CDL) is scheduled to be released in July 2023.

Figure 2 shows the functional structure of the ConnectionChain at the Research Portal. ConnectionChain consists of two features: **Multi-Scenario Function (Scenario Information Management and Scenario Control)**, and **Key Management Function**.

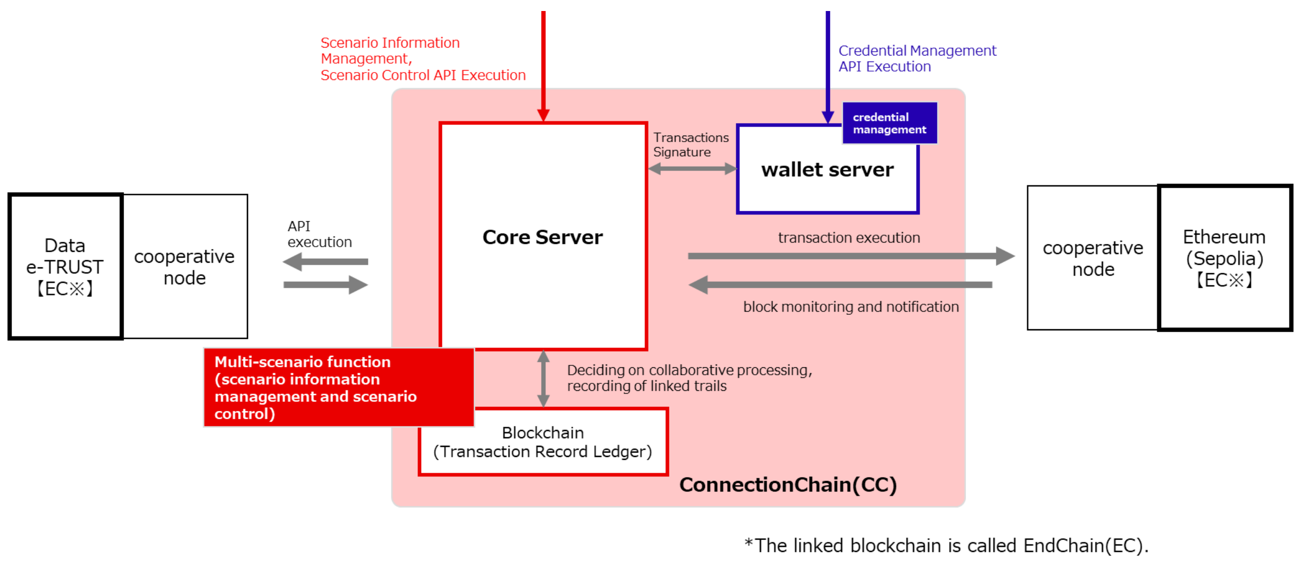

Scenario Information Management, Scenario Control API Execution
Credential Management API Execution
credential management
Transactions Signature
wallet server
Core Server
Data e-TRUST 【EC※】
cooperative node
API execution
transaction execution
block monitoring and notification
cooperative node
Ethereum (Sepolia) 【EC※】
Multi-scenario function (scenario information management and scenario control)
Deciding on collaborative processing, recording of linked trails
Blockchain (Transaction Record Ledger)
ConnectionChain(CC)
*The linked blockchain is called EndChain(EC).

Figure 2: Function Configuration of ConnectionChain

<Mechanism of Multi-Scenario Function>

This function enables cooperation between blockchains by writing a "Scenario Script" that represents a set of transactions for each linked blockchain (EC) in a pseudo language. The Research Portal supports API execution on Data e-TRUST and transaction execution on Ethereum (Sepolia testnet). Transaction processing according to the Scenario Script takes place on the Core Server and blockchain. Smart contracts deployed in the blockchain make collaborative processing decisions and record federated trails.

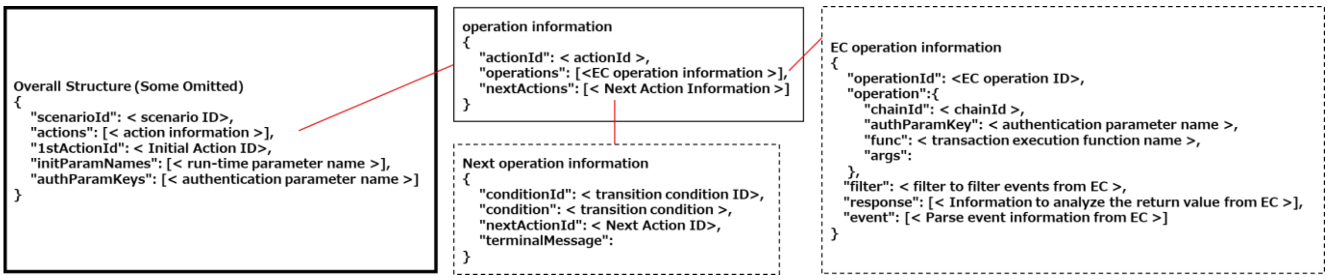
<Mechanism of Key Management Function>

The Key Management Function manages the private key of the account registered with the EC (Ethereum). The Core Server asks the Wallet Server to sign the transaction using the identifier (client ID) that identifies the key given by the end user when the cooperative processing is executed. Then, a signed transaction is obtained. The client ID is tied to the end user's Azure ID and can only be used to sign transactions with the private key.

See Sections 1.2 and 1.3 for a more practical explanation of the features and usage flow.

1. 2 Multi-Scenario Function (Scenario Information Management and Scenario Control)

The Multi-Scenario Function is a function that enables cooperation between blockchains by writing a Scenario Script that represents a set of transactions for each linked blockchain (EC) in a pseudo language. The Scenario Script supports writing in a common format independent of the EC type, and has the following format (Figure 3).



```
Overall Structure (Some Omitted)
{
   "scenarioId": < scenario ID>,
   "actions": [< action information >],
   "1stActionId": < Initial Action ID>,
   "initParamNames": [< run-time parameter name >],
   "authParamKeys": [< authentication parameter name >]
}
```

```
operation information
{
   "actionId": < actionId >,
   "operations": [<EC operation information >],
   "nextActions": [< Next Action Information >]
}
```

```
Next operation information
{
   "conditionId": < transition condition ID>,
   "condition": < transition condition >,
   "nextActionId": < Next Action ID>,
   "terminalMessage":
}
```

```
EC operation information
{
   "operationId": <EC operation ID>,
   "operation":{
      "chainId": < chainId >,
      "authParamKey": < authentication parameter name >,
      "func": < transaction execution function name >,
      "args":
   },
  "filter": < filter to filter events from EC >,
  "response": [< Information to analyze the return value from EC >],
  "event": [< Parse event information from EC >]
}
```

Figure 3: Scenario Script Format

The Scenario Script mainly consists of the scenario ID, Action Information (array), first Action ID, run-time parameter name, and authentication parameter name. The Action Information includes processing for EC (Transaction execution, reference, etc.), and when the Scenario is started, the processing defined in the Action Information is executed in order from the first Action ID. The runtime specification parameter name and the authentication parameter name are variable names for handling the value input at the start of the Scenario in this script, and the former is called the Instance Dynamic Value. Action Information includes EC Operation Information (array) and Next Action Information (array), and defines processing for EC (what kind of processing is performed for what kind of EC) and Transition Conditions (When and what actions to perform next based on the processing result (s) of EC) for determining next Action ID.When a Scenario is started, it follows the Scenario Script until the next Action ID is empty.

EC Operations execution (included in the first Action) -> Transition Condition judgment and next Action decision based on the EC Operation result

-> Execution of EC Operations (included in next Action) -> Decision of Transition Condition and decision of next Action based on EC Operation result ...

The following describes the general flow of API execution when using the multi-scenario function (Figure 4).

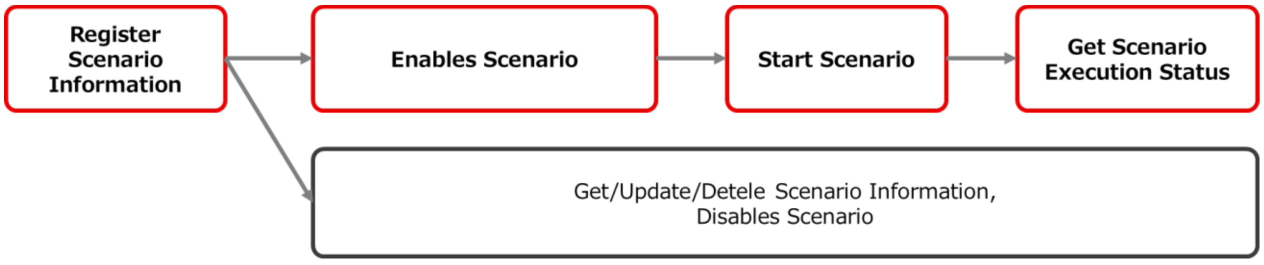

Register Scenario Information
Enables Scenario
Start Scenario
Get Scenario Execution Status
Get/Update/Detele Scenario Information, Disables Scenario

Figure 4: Flow of using the Multi-Scenario Function

When using this function, the user follows the flow of the red frame shown in the figure.
The user first executes the Scenario Information Registration API, and then registers the Scenario Script described above to CC. When a user activates a registered Scenario by executing Enables Scenario API, the Scenario is ready for execution. When the user executes the Scenario Start API, CC executes the processing defined in the Scenario Script. You can check the execution result by executing the Get Scenario Execution Status API.

1.3 Key Management Function

The Key Management Function manages the private key of the account registered with the EC (Ethereum in the Research portal). The flow of use and the mechanism of this function are described in Fig. 5.

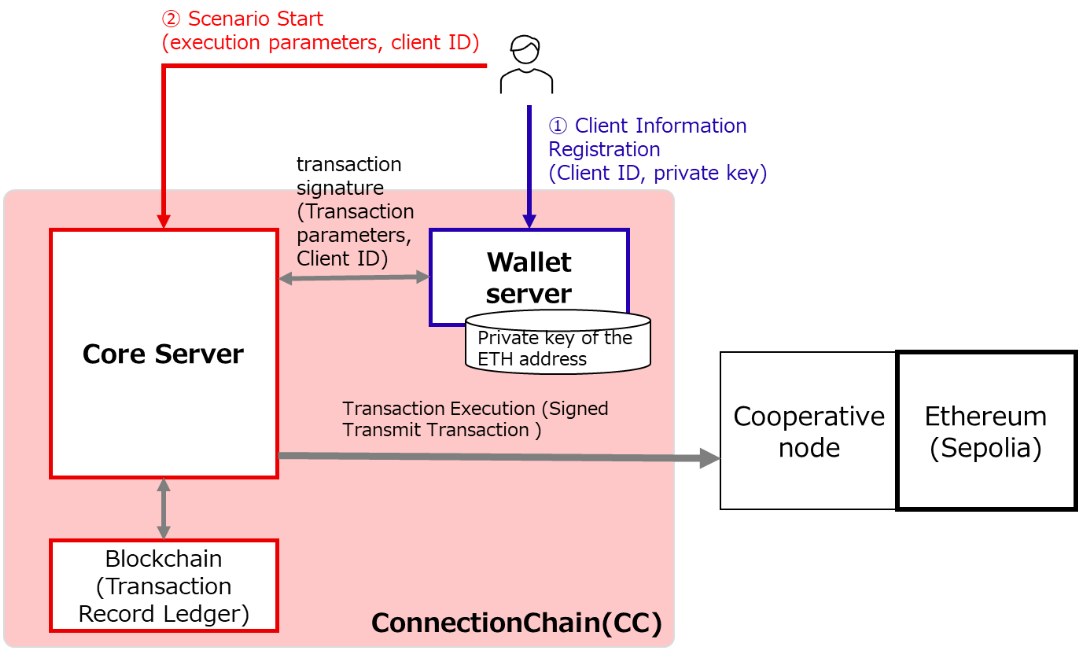

② Scenario Start
(execution parameters, client ID)
① Client Information Registration
(Client ID, private key)
transaction signature (Transaction parameters, Client ID)
Wallet server
Private key of the ETH address
Core Server
Transaction Execution (Signed Transmit Transaction)
Cooperative node
Ethereum (Sepolia)
Blockchain (Transaction Record Ledger)
ConnectionChain(CC)

Figure 5: Flow of using the Key Management Function

Users can use this function if the Scenario Script described in the previous section defines the process of executing Ethereum transactions from CC. In other words, it is a case where a transaction executed by Ethereum is signed by CC (*). In such a case, the user first registers the client information including the client ID and the private key of the Ethereum address with the wallet server (① in the figure 5). Export the private key of the Ethereum address from a wallet such as MetaMask (**Create an address for exclusive use of CC separately from the address used for services such as Ethereum mainnet.**). You need to keep the gas charge and the amount of money to execute the transaction by Ethereum at the registered address. The acquisition of gas, please use https://sepoliafaucet.com/. Refer to section 2.1 of the Environmental Value Application User Documentation for information on creating MetaMask wallets and connecting to networks.

When executing the Scenario, the user passes the client ID, which is the identifier of the key registered in ①, as the execution argument of the Scenario Start API described in Section 1.4 (② in the figure 5). When CC executes an Ethereum transaction during the execution of the Scenario, the transaction signature is requested to the Wallet Server using the submitted client ID. The Wallet Server authorizes the signature with the registered private key before signing the transaction. The signed transaction is then returned to the Core Server and sent to Ethereum.

(*) This does not apply to the case where only Data e-TRUST is linked and "EC-initiated Action Activation (described in Section 2.2.1)" is used to monitor transactions on Ethereum.

2. Interface (API Specification)

This chapter describes the API specification for ConnectionChain (Scenario Information Management, Scenario Control, and Key Management). Section 2.2 ~ 2.4 describes a list of APIs (Interface Details in OpenAPI Documentation). See Section 2.5 for the response at abnormal termination.

2. 1 Common Specification

Before using ConnectionChain, a user must be registered in our company (ID issued). When making an API request described in this manual, it is necessary to authenticate using the user ID, obtain the access token (< ACCESS_TOKEN >), and set the Authorization: Bearer < ACCESS_TOKEN > header in each API request. The user ID, authentication, and access token are described in detail on the Research Portal web page.

2. 2 Scenario Information Management

2. 2. 1 Register Scenario Information

Description	Register Scenario Information
Method	POST
Path	/cc_scenarios

(*1) Support for Dynamic Values

By specifying "@ + Instance Dynamic Value Name", the values entered during the execution of the Scenario can be used. For Instance Dynamic Value Names, variable names (initParamNames) that specify values to be set at the start of Scenario execution and variable names (paramName) that specify values returned or notified from EC can be used. If a Dynamic Value Name is specified, it is replaced with the corresponding value (Input at the start of Scenario execution and return or notification from EC) during Scenario execution.

Example) Instance Dynamic Value Name "username", for Instance Dynamic Value "taro"

Description of scenario: {"target": "@username"} → Replaced value: {"target": "taro"}

If the Instance Dynamic Value can be treated as a numerical value, it can be expressed as a calculation formula (Arguments (args), Transition Conditions, and filter values are primary targets).

Example) 2 * @value + (100 +&rate)/100

However, calculations that do not include Dynamic Values, exponential notation, remainder calculations, and calculations other than decimal are not supported.

Example) (2 ** 3 + 4)% 10

(*2) Support for Dynamic Value Partial Substitution

By enclosing Instance Dynamic Value Names with '(single quotes), a portion of the string can be replaced during a Scenario run.

Example) Instance Dynamic Value Name "username", for Instance Dynamic Value "taro"

Description of scenario: {"url": "http://x.x.x/'@username'/test"} → Replaced value: {"url": "http://x.x.x/taro/test"}

In addition, this substitution precedes the value conversion of Dynamic Values not enclosed in ', so it can be applied to those Dynamic Value names.

Example) If the Instance Dynamic Value Names are "user_admin", "user_normal", "userType"

If you write "@user_'@userType'", you can see Dynamic Values for user_admin or user_normal by switching userType to admin or normal.

(*3) Transition Conditions

{operationId: < EC Operation ID >, type: < operation type >, result. < field name > < comparison operator >} or a string concatenated with ||, &&. If || and && are mixed, enclose the combination clearly in ().

<Operation Type>: REF/REQ/EVE

REF ⇒ Represent the response of the reference EC Operation (Operations that do not involve an EC state change, such as get balance or GET requests in the API)

REQ ⇒ Represent the response of the request EC Operation (Operations involving EC state changes, such as EC transaction execution or POST requests in the API)

EVE ⇒ Indicates event notification from EC when a request (including request EC operation) with event notification or filter setting is performed

< fieldname >: Name defined in the name area of response or event

<Comparison Operator>: You can specify "==", "!=", ">=", "<=", ">", and "<". When "> " or" < " is included, the comparison is performed as a numerical value, and when not included ("==", "!="), the comparison is performed as a character string.

<value>: Dynamic Value can be used. To specify an empty character, write "' ' (two single quotes) ".

Example) When you want to match the value of from (described in the name area of event) contained in the event notified from EC

{operationId: 1, type: EVE, result.from == 0x407d73d8a49eeb85d32cf465507dd71d507100c1}

Example) When you want to make a judgment based on a match between the value returned by the EC Operation and the value contained in the event notified from the EC

{operationId: 1, type: REQ, result.from == 0x407d73d8a49eeb85d32cf465507dd71d507100c1} &&
{operationId: 1, type: EVE, result.from == 0x407d73d8a49eeb85d32cf465507dd71d507100c1}

When the Transition Condition is to be judged only by "If the Operation request itself succeeds or fails, regardless of the result of the Operation", the field name is omitted and described as follows.

"result <Comparison Operators (== or !=)> <value (OK or NG)>"

Note that "OK" can be specified for the value if response or event is omitted for the EC Operation of the operationId (For "NG" statements, regardless of response or event).

Example) Determining success only, without reference to values returned directly by EC Operations

{operationId:1, type:REQ, result==OK}

■Notice (type: when dealing with EVE conditions)

In some cases, even though the transition to the next Action has already been made by the judgment of the Transition Condition, the judgment of the condition is made again, and the next Action is executed.

Completely cover "operation" described in "operations" to be judged and "synchronous result (REQ or REF)/asynchronous result (EVE)" which are the result of operation with AND ("&&").

Bad Example 1) Unintended behavior because the synchronization result (REQ) satisfies conditionId: 2 first, and the asynchronous result (EVE) also satisfies conditionId: 1 (Action ID "transfer " is executed twice).

nextActions: [

{conditionId: "1", condition: "{operationId: 1, type: REQ, < conditionA >} & & {operationId: 1, type: EVE, < conditionB >}", nextActionId: "transfer"},

{conditionId: "2", condition: "{operationId: 1, type: REQ, < conditionA >}", nextActionId: "transfer"}

]

=>conditionId: 2 is not required (**When both REQ and EVE operation types are used in a condition, they should not be written alone.**)

Bad Example 2) Unintended behavior because the synchronous result (REQ) satisfies conditionId: 1 and the asynchronous result (EVE) also satisfies conditionId: 2 (Action ID "transfer " is executed twice)

nextActions: [

{"conditionId":"1" , "condition" : "{operationId: 1, type: REQ, result == OK}" , "nextActionId" : "transfer"},

{"conditionId":"2" , "condition" : "{operationId: 1, type: EVE, result.to == xx}" , "nextActionId" : "transfer"}

]

=>The synchronous result (REQ) and the asynchronous result (EVE) are described by connecting "&&" as follows (**when both REQ and EVE operation types are used in the condition, they are not described independently**).

nextActions: [

{"conditionId" : "1" , "condition" : "{operationId: 1, type: REQ, result == OK} & & {operationId: 1, type: EVE, result.to == xx}" , "nextActionId" : "transfer"}

]

Bad Example 3) Unintended result because the asynchronous result (EVE) of operationId: 1 fills conditionId: 1, and the asynchronous result of operationId: 2 also fills conditionId: 2 (Action IDs "transferX" and "transferY" are both executed)

nextActions: [

{conditionId: "1", condition: "{operationId: 1, type: EVE, < conditionA >}", nextActionId: "transferX"},

{conditionId: "2", condition: "{operationId: 2, type: EVE, < conditionB >}", nextActionId: "transferY"}

]

=>If there are multiple EC operations that use asynchronous results in the same operation as described below, they shall be described by connecting them with "&&" (**not described separately when multiple conditions of Operation Type EVE are used**).

nextActions: [

```
    {conditionId: "1", condition: "{operationId: 1, type: EVE, < conditionA >} && {operationId: 2, type: EVE, < conditionB >}",
nextActionId: "transferX"}
    ]
```

(*4) EC-initiated Action Activation

< Function Description >

This function is used to execute EC transactions from applications other than CC after taking into account the risks and man-hours of depositing EC private keys in Fujitsu (see Key Management API) and executing transactions from CC. In CC, the transaction execution of EC can be monitored by the Scenario execution, and it can be connected to the processing to another EC.

< Specific Usage >

By describing specific/record information (event) for the result notified from EC as an event and variable name/value (filter) in Tx as an event notification condition as a set, CC can monitor transactions executed in EC. The filter description should include information that uniquely identifies the transaction, such as txid (Because there is a risk of filtering unexpected transactions and determining Transition Conditions). In the description of EC Operation Information, only chainId is defined, and in the Transition Condition, a condition whose Operation Type is EVE is described and judged. When activating the operation with an Ethereum origin, start the Scenario defining the activation of the operation with an EC origin until 25 blocks are committed (approximately 5 minutes) after executing the transaction with Ethereum, and set filter (because the number of blocks of Ethereum cached by CC is 25 blocks).

Example) Filter Ethereum transactions using the Instance Dynamic Value (txId) given in the Scenario start parameter to determine the from/to value.

```
{
    "operationId": "1",
    "operation": {
        "chainId": "Geth"
    },
    "filter": {"hash": "@txId"},
    "event": [{"name": "from" , "path": "from"} , {"name": "to" , "path": "to"}]
}
```

Input Specifications For EC

The following values contained in the Scenario Information have different input specifications depending on the EC type (See Input Specifications for EC below):

- Transaction execution function name (func)
- Arguments given to the transaction execution function (args)
- Identify or record information (response) for results returned directly by EC Operations
- Identification or logging information for the result reported as an event by the EC (event)
- Variable name or value (filter) in Tx that becomes the event notification condition

Each function or func in each table has a different value that can be specified for the Operation Type included in the Transition Condition (expressed by color).

Red: REF (reference EC Operation)

Blue: REQ (request EC Operation), EVE (request with event notification)

Ethereum Testnet (chainId: Geth)

Function/func	args				response	filter, event
	Parameter	Required	Type	Description		
Web 3.eth function call/call	fcn	○	string	Function name of web3.eth function (*) Function name that can be called by web3.eth. <function name> on web3js. Can be called as long as the function takes Array [string] as an argument (e.g. getBalance) Do not specify functions that change the state of the EC, such as sendSignedTransaction. Reference: web3js Documentation	Set path by referring to the return value of the web3.eth function Reference: web3js Documentation Example:For getBalance [{"name" : "amount" , "path" : ""}] (Balance is returned and stored in the amount variable)	This function is not described because it is not intended to perform transactions (reference only).
	args		Array [string]	The above argument to the web3.eth function Example:[" 0x407d73d8a49eeb85d32cf465507dd71d507100c1"]		
Remittance/send	from	○	string	The source address must be preceded by 0x and must be in lowercase Example: 0x407d73d8a49eeb85d32cf465507dd71d507100c1	The transaction receipt will be returned, so if you want to use it in response, write as follows. *path is prefixed with [{"name" : "Status" , "path" : "transactionReceipt.status"}] (the transaction status is stored in the status variable) Reference: Format of the transaction receipt (the value returned by the getTransactionReceipt function)	You can filter transactions and assign them to specific & event variables according to the format of the transaction reported to CC (see below). *Any value assigned to the event variable is converted to a string. Address values shall be written in lower case. *Note that even if event and filter are described, they cannot be guaranteed until the transaction is finalized (mainly for recording purposes). Successful execution of transactions and smart contracts can be detected synchronously and judged by writing “result == OK ” in the condition. ■Example of filter description (narrowed down to the contents of to) {"to" : "0x407d73d8a49eeb85d32cf465507dd71d507100c1 "} ■Sample description of event (value is assigned to amount variable) [{"name" : "amount" , "path" : "value"}] ■Format of transactions notified to CC Reference: web3.eth.getTransaction { "hash": "<Transaction ID>", "nonce": "<Nonce>", "blockHash": "<block hash>", "blockNumber": "<block number>", "transactionIndex": "<Tx index>", "from": "<source address>", "to": "<Destination Address>", "value": "<Remittance Amount (wei)>", "gasPrice": "<Gas Price>", "gas": "<Fees>", "Inputs": "<Transaction input data>"} *Input decoding not supported (event describes undecoded data)
	to	○	string	The destination address Address must be preceded by 0x and must be all lowercase		
	amount	○	string	Transfer amount (unit: wei) Exponential notation is also available Example: 3.78 *10^14, 3.78e14		
	gas		string	Gas consumption (in gas) Example (GasLimit during remittance): 21000		
Send Contract / contractSend	abis	○	Array [object]	Contract ABI Example: [{"constant": true, "inputs": [{"internalType": "bytes4", "name": "interfaceId", "type": "bytes4"] }],...},...] Can also be stored in instance dynamic values. In this case, add (escape) "¥¥¥" immediately before every "" in the ABI, convert it into a string, and store it in the dynamic value. Dynamic Value Example: "[{¥¥¥" constant¥¥¥": true,¥¥¥" inputs¥¥¥"Type: {¥¥¥" internalType¥¥¥":¥¥¥" bytes4¥¥¥",¥¥¥" name¥ ¥":¥¥¥" interfaceId¥¥¥",¥¥¥" type¥¥¥":¥¥¥" bytes4¥¥¥"}], ...}, ..."		

	from	○	string	Source address address must start with 0x and be all lowercase		
	to	○	string	Contract address address must start with 0x and be in lowercase Example: 0x06012c8cf97bead5deae237070f9587f8e7a266d		
	method	○	string	Contract Function Name Example: issueNFT		
	params	○	Array [string]	Contract Execution Arguments. For functions with no parameters, specify an empty array. Not supported for arguments other than Array [string] Example: ["token0" , "0x407d73d8a49eeb85d32cf465507dd71d507100c1"]		
	options		object	Describe the parameters (Transaction execution parameters, such as gasPrice, gas, and value) to be included when executing the contract. Data, nonce is not specified because it is set internally in CC Example: {"gas" : "210000" , "value" : "10000"}		
contract execution (call) / contractCall	abis	○	Array [object]	Same as contractSend	Follow the return value of contract execution. All returned values are converted to strings ■Specifying a Single Return Value by Setting an Empty Character for path Example: [{"name" : "value1" , "path" : ""}] ■ Specifying Multiple Return Values by Setting an Array Number for path Example: [{"name" : "value1" , "path" : "payload[0]"} , {"name" : "value2" , "path" : "payload[1]"}]	This function is not described because it does not execute transactions.
	to	○	string			
	method	○	string			
	params	○	Array [string]			
	options		object	List the parameters (from, gasPrice, gas) to include when the contract is called. Example: {"gas": "210,000"} Reference: Web3.eth.contract.call		

Data e-TRUST (chainId: CDL)

For more on terms used in CDL and API specifications, see "Fujitsu Computing as a Service Data e-TRUST API Reference Manual"

Function/func	args				response	filter, event
	Parameter	Required	Type	Description		
Register history (function to execute "POST/trail_registration") / registerHistoryData	eventId		string	History ID It is registered as a header part (cdl: Lineage element). If not specified, the system automatically grants it. String (Alphabet [a-z] [A-Z], number [0 -9], _, -) Maximum characters: 100 characters	Registered history JSON data history JSON format, see "Appendix A JSON format for the trail and audit function" in "Fujitsu Computing as a Service Data e-TRUST Function Specification"	Not Supported
	lineageId		string	Lineage ID It is registered as a header part (cdl: Lineage element). String (Alphabet [a-z] [A-Z], number [0 -9], _, -) Maximum characters: 100 characters		
	tags		object	List of local data names It is registered as a local data part (cdl: Tags element).		
	properties		object	Additional Properties It is registered as a global data part (cdl: Event element).		
Lineage acquisition (Function to execute "GET/trail _ acquisition/{cdleventid}") / getLineage	eventId	○	string	History ID for which Lineage is acquired	Lineage JSON Data (an array of historical JSON data sets) Historical JSON format, see "Appendix A JSON format for the trail and audit function" in "Fujitsu Computing as a Service Data e-TRUST Function Specification"	
	direction		string	Lineage search direction Specify one of the following:. The default value is BACKWARD. ・BACKWARD: lineages backward ・FORWARD: get next historical lineage ・BOTH: Get Lineage in both directions		
	depth		string	depth of the Lineage Specify one of the following:. The default value is "-1". ・0: Get history only for the specified history ID ・Integer: The specified number of previous and next histories is obtained from the specified history ID. ・-1: Get all lineages including the history of the specified history ID		
History search (search target: header part) "POST/trail_search_headers" / searchByHeader	searchType	○	string	Matching Types Used in Searches Specify one of the following three search methods ・exactmatch: exact match search ・partialmatch: partial match search ・regexpmatch: regular expression search	History group whose header matches the search condition (array of history JSON objects) History JSON format, see "Appendix A JSON format for the trail and audit function" in "Fujitsu Computing as a Service Data e-TRUST Function Specification"	
	fields	○	object	Search Criteria Specify the attribute name and value in the following format: (Multiple choices allowed) {"attribute": "value"}		
History search (search target: global data part) (function to execute "POST/trail_search_globaldata") / searchByGlobalData	searchType	○	string	Matching Types Used in Searches Specify one of the following three search methods ・exactmatch: exact match search ・partialmatch: partial match search ・regexpmatch: regular expression search	History group (array of history JSON objects) whose global data part matched the search criteria History JSON format, see "Appendix A JSON format for the trail and audit function" in "Fujitsu Computing as a Service Data e-TRUST Function Specification"	
	fields	○	object	Search Criteria Specify one or more attribute names and values in the following format: {"attribute": "value"}		

2.2.2 Update Scenario Information

Description	Update the Scenario Information. Only a part of the Scenario Information cannot be updated, and all the Scenario Information is overwritten. Can be executed only by the user who registered the scenario with the specified Scenario ID.
Method	PUT
Path	/cc_scenarios/

2.2.3 Get Scenario Information

Description	Get Scenario Information
Method	GET
Path	/cc_scenarios/

2.2.4 Delete Scenario Information

Description	Delete Scenario Information Can be executed only by the user who registered the Scenario with the specified Scenario ID.
Method	DELETE
Path	/cc_scenarios/

2.2.5 Enables/Disables Scenario

Description	Enables or Disables Scenario Can be executed only by the user who registered the Scenario with the specified Scenario ID. When the Scenario Information is updated, the Scenario becomes disabled and needs to be re-enable.
Method	PUT
Path	/cc_scenarios/<Scenario ID>/availability

2.3 Scenario Control

2.3.1 Start Scenario

Description	Start Scenario Use after enabling Scenario with the Scenario Enable/Disable API (Returns error if not enabled)
Method	POST
Path	/cc_states

2.3.2 Get Scenario Execution Status

Description	Get Scenario Execution Status
Method	GET
Path	/cc_states/<Scenario Exection ID>

2.4 Key Management function

2.4.1 Register Client Information

Description	Register Client Information in Wallet Server.
Method	POST
Path	/cc_auth/wallet/client

2.4.2 Update Client Information

Description	Update Client Information registered in Wallet Server associated with the caller's Azure user.
Method	PUT
Path	/cc_auth/wallet/client/<Client ID>

2.4.3 Get Client Information List

Description	Retrieve a list of client IDs associated with the calling Azure user from the Wallet Server.
Method	GET
Path	/cc_auth/wallet/client

2.4.4 Delete Client Information

Description	Delete Client Information associated with the calling Azure user from the Wallet Server. Specifying a nonexistent Client ID also results in success.
Method	DELETE
Path	/cc_auth/wallet/client/<Client ID>

2.5 Error Code/Message List

The error code/message list for each API execution of the ConnectionChain is described below. Error codes and messages may change due to future version upgrades.

2.5.1 Scenario Information Management and Scenario Control

When an error occurs in the execution of the API for Scenario Information Management and Scenario Control, a response body of the following format is returned.

```
 {
   "error":{
       "code":<error code>,
       "message":<error message>
}
```

Parameter	type	description
error	object	Error Description
code	int	Error code
message	string	Error messsage

< Error Code/Error Message List >

HTTP Status	error code	error message	Error Description/Response
404	1002	登録されていないシナリオ ID です。	Occurs if there is no Scenario Information corresponding to the given Scenario ID when update/get Scenario Information, Enable/Disable Scenario, or executing Start Scenario API. Confirm that Scenario Snformation corresponding to a target Scenario ID is registered, and to register the Scenario Information as necessary.
	1003	登録されていないシナリオ実行 ID です。	Occurs if the Scenario Execution Status corresponding to the target Scenario Execution ID does not exist when executing the Get Scenario Execution Status API. Verify that the Scenario Execution ID exists and execute the API.
	1004	有効化されていないシナリオです。	Occurs if the Scenario Information corresponding to the given Scenario ID is not enabled when starting the Scenario. Confirm that the Scenario is enabled by executing a Get Scenario Information API, and enable the Scenario.

422	2004	シナリオデータが指定されていません。	Occurs if no Scenario data is specified when registering or updating Scenario Information. Specify the Scenario data in the request body and execute the API.
	2005	シナリオ ID が指定されていません。	Occurs if a Scenario ID is not specified when registering Scenario Information or starting a Scenario. Specify the Scenario ID in the request body and execute the API.
	2006	シナリオデータに動作リストが定義されていません。	Occurs if Scenario data does not have actions array defined when registering or updating Scenario Information. Define the actions array in the request body and execute the API.
	2007	シナリオデータの動作情報に動作 ID が定義されていません。	Occurs if the actionId is not defined in the Scenario data when registering or updating Scenario Information. Define actionId for each Action Information of the request body and execute API.
	2008	シナリオデータの動作情報に EC 操作情報が定義されていません。	Occurs if the operations array is not defined in the Scenario data when registering or updating Scenario Information. Define the operations array in the request body and execute the API.
	2009	シナリオデータの動作情報に EC 操作 ID が定義されていません。	Occurs if operationId is not defined in Scenario data when registering or updating Scenario Information. Define operationId in each EC Operation Information of the request body and execute API.
	2010	シナリオデータの動作情報に EC 操作情報内容が定義されていません。	Occurs if operation is not defined in Scenario data when registering or updating Scenario Information. Define operation for each EC Operation Information in the request body and execute the API.
	2011	シナリオデータの EC 操作情報内容に ChainID が定義されていません。	Occurs if chainId is not defined in operation in Scenario data when registering or updating Scenario Information. Define a chainId for each operation in the request body and execute the API.
	2014	シナリオデータの EC 操作情報内容に認証用パラメータキーと Signer が両方指定されています。	Occurs if both authParamKey and signer are specified in Scenario data Operation when registering or updating Scenario Information. Execute the API with only signer or authParamKey (but not support signer).
	2015	シナリオデータの動作情報に次動作遷移条件 ID が定義されていません。	Occurs if the conditionId is not defined for an element in the nextActions array of Scenario data when registering or updating Scenario Information. Define a conditionId in each nextActions array in the request body to execute the API.

	2016	シナリオデータの動作情報に次動作遷移条件が定義されていません。	Occurs if condition is not defined for element in nextActions array of Scenario data when registering or updating Scenario Information. Define a condition in each nextActions array in the request body to execute the API.
	2017	シナリオ有効フラグが指定されていません。	Occurs when enabling or disabling Scenario without Scenario availability flag. Execute the API by specifying availability in the request body.
422	2050	シナリオデータに初期動作 ID が定義されていません。	Occurs if the Scenario data of 1stActionId is not defined when registering or updating Scenario Information. Define a 1stActionId in the request body and execute the API.
	2051	初期動作 ID で指定された動作情報がシナリオデータに存在しません。	Occurs if Action Information of actionId defined for 1stActionId in Scenario data does not exist in Scenario data when registering or updating Scenario Information. Define 1stActionId so that 1stActionId matches one of the actionIds contained in the request body and execute the API.
	2052	シナリオデータの動作情報に次動作情報が定義されていません。	Occurs if nextActions array in Scenario data is not defined when registering or updating Scenario Information. Define each nextActions array in the request body and execute the API.
	2053	シナリオデータの動作 ID が重複して定義されています。	Occurs if multiple pieces of Action Information with the same actionId are defined when registering or updating Scenario Information. Define the actionId for each Action Information of the request body so that it does not overlap, and execute the API.
	2054	シナリオデータの動作情報に EC 操作 ID が重複して定義されています。	Occurs if multiple operations with the same operationId are defined in the operations array of Scenario data when registering or updating Scenario Information. Define again so that operarionId of each EC Operation Information in the request body does not become a duplicate definition, and execute API.
	2055	シナリオデータの動作情報に次動作遷移条件 ID が重複して定義されています。	Occurs if multiple Transition Conditions with the same conditionId defined in nextActions array of Scenario data when registering or updating Scenario Information. Define each Transition Condition of the request body so that the conditionId does not overlap and execute the API.

	2056	シナリオデータの EC 操作結果特定用情報内で、同じ名前が複数回定義されています。	Occurs if the same name or paramName is defined more than once for an object in the response or event arrays in the operations array of Scenario data when registering or updating Scenario Information. Redefine each response or event object in the request body to have a unique name or paramName and execute the API.
	2057	認証用パラメータキー一覧で定義していない値が、EC 操作情報内容に使用されています。	Occurs if values that are not defined in authParamKeys are defined for authParamKey for each operation in the operations array of Scenario data when registering or updating Scenario Information. Define the value defined in authParamKeys for each EC operation information of the request body in authParamKey and execute the API.
	2058	シナリオデータの EC 操作情報として、取引実行関数またはフィルタ定義の指定が最低限必要です。	Occurs if neither func nor filter is defined for each operation in the operations array of Scenario data when registering or updating Scenario Information. Define a func or filter for each EC Operation Information of the request body and execute the API.
	2059	シナリオ開始時指定パラメータ名一覧で、同じ名前が複数回定義されています。	Occurs if the same value is defined more than once in initParamNames of Scenario data when registering or updating Scenario Information. Define a unique initParamNames for the request body and execute the API.
	2060	認証用パラメータキー一覧で、同じ名前が複数回定義されています。	Occurs if identical values for authParamKeys defined in Scenario data when registering/updating Scenario Information. Define a unique value for authParamKeys in the request body and execute the API.
	2061	シナリオデータの次動作情報で、未定義の動作 ID が指定されています。	Occurs if the actionId specified for nextActionId is not defined for an element in the nextActions array of Scenario data when registering or updating Scenario Information. Define a defined actionId (or an empty character) for nextActionId in the request body and execute the API.
	2062	シナリオデータの次動作情報で、現在の動作と同じ動作 ID が指定されています。	Occurs if the actionId specified for nextActionId is the same as the current actionId for an element in the nextActions array of Scenario data when registering or updating Scenario Information. Define an actionId (or an empty character) other than the current actionId for nextActionId in the request body and execute the API.

	2063	シナリオデータの次動作情報で、実行済みの動作と同じ動作 ID が指定されています。	Occurs if the actionId specified for nextActionId is the same as the previously executed actionId for an element in the nextActions array of Scenario data when registering or updating Scenario Information. For nextActionId of request body, define actionId (or empty character) other than actionId already executed and execute API.
	2064	シナリオデータの動作情報または次動作遷移条件で、未定義のパラメータ名が指定されています。	Occurs if an undefined Instance Dynamic Value Name (@ + string) is defined for each operation (chainId, func, args, filter) or nextActions array element in the operations array of Scenario data when registering or updating Scenario Information. Define Instance Dynamic Value Names in initParamNames of the request body and paramNames in the event, response array, and execute the API.
	2065	シナリオデータの次動作遷移条件が正しい形式で記述されていません。	Occurs if there is an error in the format of the element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define a well-formed condition in the request body and execute the API. Reference: Transition Condition Format {operationId: <EC Operation ID>, type: <operation type >, result. <field name> <comparison operator>} or a string concatenated with ||, &&. If || and && are mixed, enclose the combination clearly in ().
	2066	シナリオデータの次動作遷移条件で、operatonId 領域の書式が不正または記述されていません。	Occurs if there is an error in the operationId description for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define the request body condition according to the correct operationId format and execute the API.
	2067	シナリオデータの次動作遷移条件で、operatonId 領域の値が指定されていません。	Occurs if there is an operationId description error for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define the EC operation ID value following operationId: for the condition of the request body and execute the API.

	2068	シナリオデータの次動作遷移条件で、未定義の EC 操作 ID が operatonId 領域に指定されています。	Occurs if an undefined operationId is specified for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define the operationId value defined in the Action for the operationId in the condition of the request body, and execute the API.
	2069	シナリオデータの次動作遷移条件で、type 領域の書式が不正または記述されていません。	Occurs if there is an error in the description of type for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define the request body condition according to the correct type format and execute the API.
	2070	シナリオデータの次動作遷移条件で、type 領域の値が指定されていません。	Occurs if there is a type description error for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. For the condition of the request body, define the operation type after "type:" and execute the API.
	2071	シナリオデータの次動作遷移条件で、type 領域の値に[REF/REQ/EVE]以外が指定されています。	Occurs if type is a string other than [REF, REQ, EVE] for the element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Execute API by defining [REF, REQ or EVE] as type in condition of request body.
	2072	シナリオデータの次動作遷移条件で、result 要素の書式が不正または記述されていません。	Occurs if there is a description error in result for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Define the request body condition according to the correct result format and execute the API.
	2073	シナリオデータの次動作遷移条件で、result 要素に使用する演算子が不正または記述されていません。	Occurs if there is an error in the operator used for the result element for an element (condition) in the nextActions array of Scenario data when registering or updating Scenario Information. Execute the API by defining one of the following operators in the result in the condition of the request body: “==”, “!=”, “>=”, “<=” ,">”, or “<”.
	2074	シナリオデータの次動作遷移条件で、result 要素のフィールド名が記述されていません。	Occurs if there is a description error in the field name (followed by ".") of the result element for the element (condition) contained in the nextActions array of the Scenario data when registering or updating Scenario

			Information. Execute the API by defining the field name after "." in result in the condition of the request body.
	2075	シナリオデータの次動作遷移条件で、EC 操作結果特定用情報に未定義の名前が result 要素のフィールド名に指定されています。	Occurs if the field name (Write after ".") of the result element is not defined in the name part of response, event () for the element (condition) contained in the nextActions array of Scenario data when registering or updating Scenario Information. In the result in the condition of the request body, define the field name defined in response, event (name) in the Operation of the EC Operation ID defined in the operationId of the Transition Condition, and execute the API.
	2076	シナリオデータの次動作遷移条件で、フィールド名なしの result 要素に使用可能な演算子は[==/!=]、値は[OK/NG]のみです。	Occurs if the field name (Write after ".") of the result element is not defined for the element (condition) contained in the nextActions array of Scenario data when registering or updating Scenario Information. Execute the API by defining the result in the condition of the request body as one of the following operators: “==”, “!=”, “OK” or “NG”.
	2077	シナリオデータの動作情報または次動作遷移条件で、不正な計算式が記述されています。	Occurs if there is an error in the format of the calculation described in each operation in the operations array of Scenario data or in the element (condition) in the nextActions array when registering or updating Scenario Information. Define a well-formed equation for the request body and execute the API.
	2078	シナリオデータの EC 操作結果特定用情報で、パスの書式が不正です。	Occurs if the path format of the object in the response or event array in the operations array of Scenario data is invalid when registering or updating Scenario Information. Define a properly formatted path for each object in the response, event array of the request body and execute the API.
	2079	開始時指定パラメータ名一覧は配列で指定してください。	Occurs if the initParamNames of the Scenario data is not an array when registering or updating Scenario Information. Define an array in the initParamNames of the request body and execute the API.
	2080	認証用パラメータキー一覧は配列で指定してください。	Occurs if authParamKeys of Scenario data is not an array when registering or updating Scenario Information. Define an array in the request body's authParamKeys and execute the API.

	2081	シナリオデータの動作リストは配列で指定してください。	Occurs if the actions of the Scenario data is not an array when registering or updating Scenario Information. Define an array for actions in the request body and execute the API.
	2082	シナリオデータの EC 操作情報は配列で指定してください。	Occurs if the operations of the Scenario data is not an array when registering or updating Scenario Information. Define an array for operations in the request body and execute the API.
	2083	シナリオデータの EC 操作結果特定用情報は配列で指定してください。	Occurs if the response or event array in the operations array of scenario data is not an array when registering or updating scenario information. Define an array for response, event in the request body and execute the API.
	2084	シナリオデータの次動作情報は配列で指定してください。	Occurs if the nextActions array for Scenario data is not an array when registering or updating Scenario Information. Define an array in nextActions in the request body and execute the API.
	2085	シナリオデータの次動作遷移条件で、result 要素の値が記述されていません。	Occurs if the value of the element (condition) in the nextActions array of the Scenario data does not follow the operator in the result element when registering or updating Scenario Information. Execute the API by defining the value following the result operator in the request body condition.
	2100	シナリオ開始時指定パラメータの数が、シナリオデータで定義された数と一致しません。	Occurs if the number of parameters in the params array specified at the starting of the Scenario does not match the number in the initParamNames array defined in the Scenario data. Change the params at the starting of the Scenario or initParamNames of the Scenario data, and execute the API after combining the numbers of both.
	2101	認証用パラメータの数が、シナリオデータで定義された数と一致しません。	Occurs if the number of parameters in the authParams array specified at the starting of the Scenario does not match the number of authParamKeys arrays defined in the Scenario data. Change the authParams at the starting of the Scenario or the authParamKeys of the Scenario data, and execute the API after combining the numbers of both.
	2102	シナリオ開始時指定パラメータは文字列の配列で指定してください。	Occurs if the value specified in the params array at the starting of the Scenario is not an array of strings.

			Execute the API with an array of strings in the request body params.
	2103	認証用パラメータは配列で指定してください。	Occurs if the value specified in the authParams array at the starting of the Scenario is not an array of strings. Execute the API by specifying an array of strings for authParams in the request body.
400	2104	シナリオ開始情報が指定されていません。	Occurs if no request body is specified at the starting of the Scenario. Execute the API by specifying parameters in JSON format in the request body.
	2200	リクエストデータのサイズが上限を超えています。	Occurs if the request data exceeds the maximum size limit (50 MB) when each API is executed. Specify request data smaller than the upper limit and execute each API.
500	3000	内部エラーです。	Internal error. If the error persists after several runs at different times, you should contact person in charge of our company.
	3002	既に登録済みのシナリオ ID です。	Occurs if the Scenario Information corresponding to the specified Scenario ID has already been registered when registering Scenario Information. Change Scenario ID and execute Register Scenario Information API, or execure Update Scenario Information API.
	3003	既に登録済みの実行状況 ID です。	Occurs if the specified Scenario Execution ID is already registered when starting a Scenario. Change the stateId to execute the API.
403	4001	シナリオ情報を操作する権限がありません。	Occurs if the executing user does not have permission to execute the Scenario Information when updating, deleting, disabling, or enabling Scenario Information. Specify the access token of the user who registered the Scenario in the Authorization header and execute the API.
503	5000	他シナリオの実行によるロック中です。少し待ってから再度実行してください。	Occurs if execution cannot be accepted because another user is executing when starting the Scenario. Wait about 20 seconds and execute API again (You might have to wait 20 seconds or more depending on the execution status of other users.)
-	9000	不明なエラーです。	Occurs if the Core server returns an error code that is not defined. Contact a person of our company in charge.

2.5.2 Key Management

If an error occurs during execution of the Key Management API, the following error message is returned. If you make several correct requests according to the specifications and an error not listed in the table below occurs, please contact a person of our company in charge.

< List of error messages >

HTTP Status	Error message (response body)	Error Description/Response
400	{"result": "not exist!"}	At login, an error occurs if the client information for the specified client ID does not exist. Specify an existing client ID and execute the API. (Check if it exists by executing the client information list acquisition API)
	{"result": "invalid parameter!"}	Error in input parameter specification. (Required parameter missing or over size). Run the API with required parameters.
403	{"result": "not authorized!"}	An error occurs if you do not have permission to operate on the client ID when updating, deleting, or signing a transaction.Execute API by specifying the client ID registered by itself.
404	Not Found	An error occurs if the resource does not exist in the requested path. Verify that the path is correct and run the API.
500	{"result": "Already exist!"}	When registering client information, an error occurs if the client information for the specified client ID already exists. Specify the correct client ID and execute the API.
	{"result": "not exist!"}	When updating client information, an error occurs if no client information exists for the specified client ID. Specify an existing client ID and execute the API. (The existence of the client ID is confirmed by executing the client information acquisition API.)