TIBCO iWay® Service Manager

Service Manager and Blockchain Solutions Development Guide

Version 8.0 and Higher
March 2021
DN3502252.0321

Contents

Chapter 1

Introducing Blockchain

This chapter provides an introduction to blockchain and an overview of integrating iWay Service Manager (iSM) with blockchain applications.

In this chapter:

What is a Blockchain?

A blockchain is a distributed record of transactions validated and stored at multiple peer locations in a network. The records form a consensus of replicated, shared, and synchronized information. For example, the debits and credits applied to a bank's accounts might be stored in the blockchain. The ledger (blockchain) is not the record of the accounts themselves, but rather the changes in account position.

The records stored in the blockchain are immutable, providing a high level of security and integrity. Each record is chained to the prior record, hashed, and encrypted. The records form a chain of transactions from the initial transaction to the most current. The records are duplicated among the peers in the network, and available to any authorized party for viewing. The result is a horizontal (peer storage) and vertical (historic chain) mesh that is virtually tamper proof. This results in a high level of trust, transparency, and reliability.

Once an item is recorded, it is irrefutable proof that this event occurred at a specific time and date, and (perhaps) between specific counterparties. The use of blockchains solves the problem of transferring value between parties (or entities) without the need to rely on a third intermediating party.

In the enterprise business world, a private blockchain provides a basis for non-repudiation and automates many of the existing labor-intensive processes required to settle transactions. Blockchains can reduce transaction and back office costs, the complexity of cross-enterprise business processes, fraud, and inefficiencies in the business to free up capital.

Blockchains can also take advantage of smart contracts. Smart contracts are self-executing contracts or agreements that are represented as software that can automatically trigger actions under certain conditions, such as when payments are made (or missed).

Smart contracts are implemented in chaincode, which implements the application in the peers. The peers run this code, and at a future point, validates the transaction to determine, for example, whether an asset should or should not be transferred.

In iWay Version 8, support for private blockchain maximizes the advantages of this modern approach to transaction interchange and non-repudiable storage.

Supported Blockchains

iWay Service Manager (iSM) supports Hyperledger Fabric **Permissioned or Private Blockchain (e.g. business applications)**. This type of blockchain requires users to be authorized to gain access (be invited) into the blockchain and limits the number of people who are granted access to maintain privacy of transactions. These are the only blockchain types currently supported by iSM.

Note: There are also **Permissionless or Public Blockchain (e.g. Bitcoin)**. This type of blockchain can be accessed by anyone and anyone can read/write to the blockchain. Currently, iSM does not address permissionless blockchain applications.

Blockchain Prerequisites

Before you continue, ensure that you have the following components installed on the platform(s) on which you will be integrating iWay Service Manager (iSM) with your blockchain:

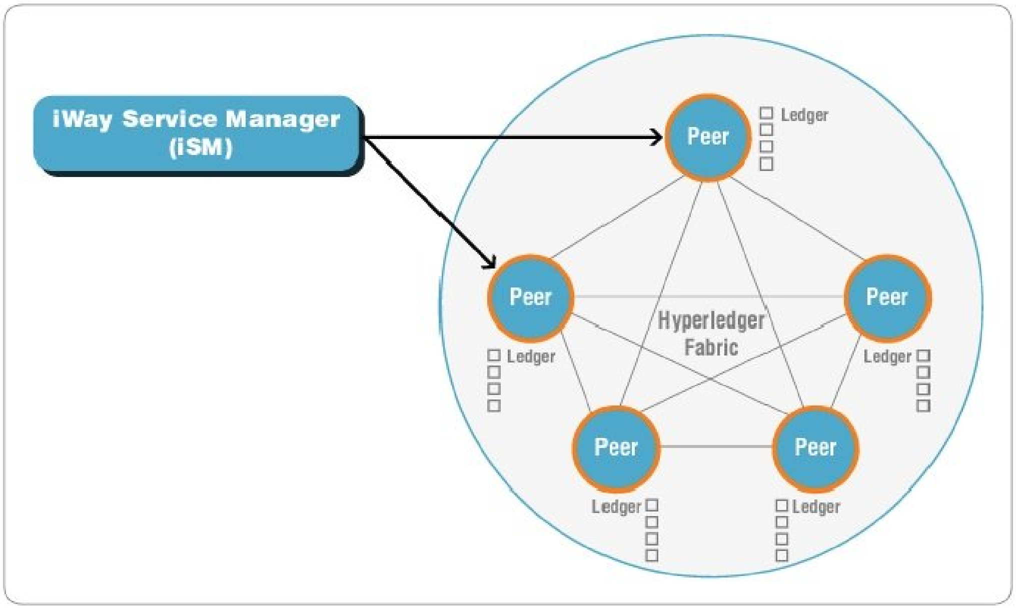
- ❑ iSM Version 8.0 or higher.
- ❑ Hyperledger Fabric Version 1.0.1 or higher.

  You can download Hyperledger Fabric from the following website:

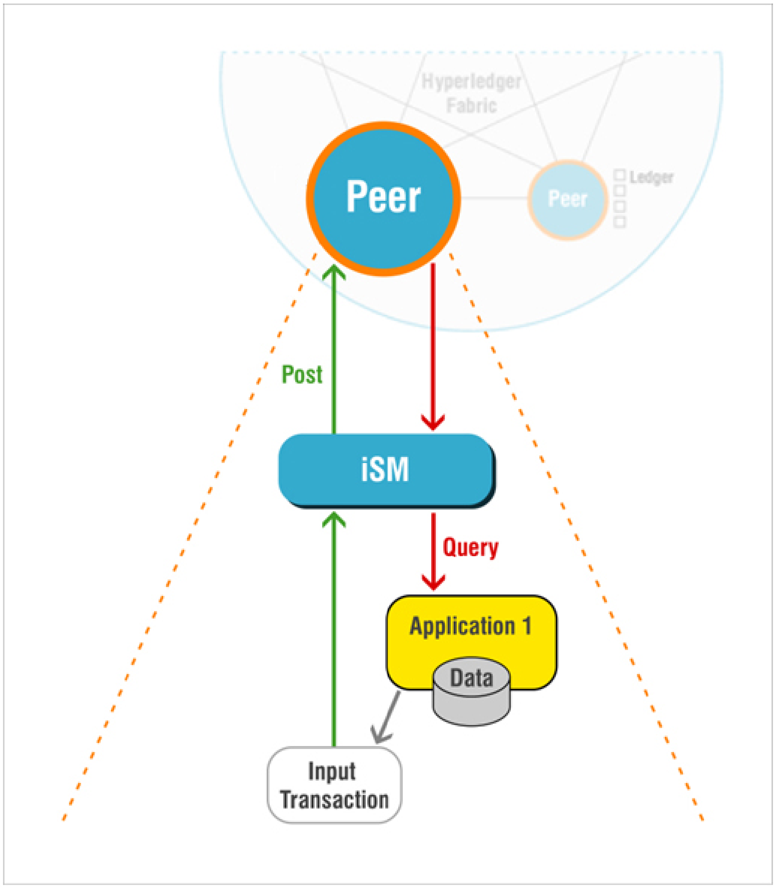
  https://github.com/hyperledger/fabric/tree/release
- ❑ A chaincode that is appropriate to your application.

How can iWay Service Manager Integrate With My Blockchain Application?

The blockchain workflow provides participants end-to-end visibility based on their level of permission. Detailed visibility of the workflow process is enhanced with real time exchange of events and documents.



This level of detail and transparency helps reduce fraud and errors, reduces the time required to complete the workflow, and improves the management of the workflow.



Note, the read on the specific peer in this diagram is only intended for illustration purposes. There is no inherent quality of the "local peer".

Confederation of Applications

The blockchain in an actual usage scenario, provides a desired transaction record consistently across multiple parties that are interested in that information. Each member can be assured that all parties view the same information, although they may use this information for their own requirements. You can imagine this as the above diagram, but with access to the peers falling into a sector of application interest. The application-level data appropriate to that specific sector of interest may vary based on the specific purpose of the application, although all accept their input from the consistent blockchain data. The blockchain itself is the ledger of activity.

iSM works with and alongside your Hyperledger Fabric-based applications. Incorporating iSM into the blockchain workflow reduces process development cost and time as well as providing for faster implementation. iSM graphically builds your process flow and provides full integration and configuration via adaptors. The in-process flow query offers transaction intelligence and the ability to print logical reports.

iSM listeners provide input message protocols through dedicated event listeners.

Services help handle payloads and the Fabric facility offers services to support several means of posting transactions. iWay Service Manager (iSM) utilizes the following types of services to post to the blockchain:

- ❑ **Transaction Service.** Executes a transaction on a blockchain in the Hyperledger Fabric. This service supports **synchronous** and **asynchronous** posting. For more information, see *Transaction Service* on page 10.
- ❑ **Query Service.** Executes a read-only query in the Hyperledger Fabric. For more information, see *Query Service* on page 13.
- ❑ **Query Block Service.** Queries the Hyperledger Fabric for a specific block and returns this block. For more information, see *Query Block Service* on page 13.

Hyperledger Fabric Components

The Hyperledger Fabric facility in iWay Service Manager (iSM) includes access to the Fabric system, posting services, and query services. For more detailed information on these components, see *Hyperledger Fabric Component Reference* on page 15.

Channel Provider

Connection to the Hyperledger Fabric system is handled through a Channel provider. This provider is responsible for securing the connection and connection recovery. Separate Channel providers, each with a unique name or properties, can be defined as required. The posting and query services must be configured to refer to an available Channel provider.

For more information and an example, see *Hyperledger Fabric Channel Provider* on page 15.

Transaction Service

The Transaction service invokes chaincode (embedded logic that encodes rules for specific types of network transactions) to execute a transaction on a blockchain in the Hyperledger Fabric. It starts by sending a transaction proposal request to all the configured peers. If enough proposal responses are successful (and not too many have failed), then the transaction is sent to the configured Orderers. The Transaction service can be configured in two modes:

- Synchronous Posting
- Asynchronous Posting

For more information and several examples, see *Hyperledger Fabric Transaction Service* on page 18.

Synchronous Posting

Synchronous posting sends the transaction proposal to the peers, evaluates the response, and then sends the proposal responses to the Orderer. Synchronous posting requires a response from the Orderer in order for the process flow to proceed. The subsequent portion of the flow will be *aware* of the resultant success or rejection of the posted transaction.

The following is an outline of the synchronous posting process:

- Runs in the process flow (pflow) as part of its logic.
- iSM standard pflow services and transformers form the payload (transaction).
- The posting service (under the covers) handles the two-way communication within the network peers.
  - Posts the transaction to multiple peers.
  - Peer chaincode validates the transaction against the smart contract.
  - Chaincode sends results back to the posting service.
  - iSM internal logic can perform first level validation of results (allows performance improvements).
  - Upon validation by iSM, the peer results are sent to the Orderer.

- Fabric's Orderer
    - Ensures the network is consistent and has ultimate control of the network.
- Posting Service (the last step)
    - Makes results available to the flow and lets the flow continue.

Asynchronous Posting

Asynchronous posting to the blockchain enables the posting flow to proceed without waiting for the response from the Orderer, which provides an advantage. This enables the application to proceed and handle results of the post at a later time.

The asynchronous post service, like the synchronous post service, uses the Fabric provider to reach the Hyperledger system.

The Orderer sends the new block to the peer, the peer writes the new block, and then sends an event to all the parties that have registered with its event hub. Standard iSM Internal Queue operation presents these response messages to a process flow configured on the queue; that process flow handles both success and failure responses from the Orderer. If, however, you elect to shortcut error handling by configuring the posting service to perform basic validation of peer responses, these early rejections will not be sent to the internal queue as they have not yet reached the Orderer.

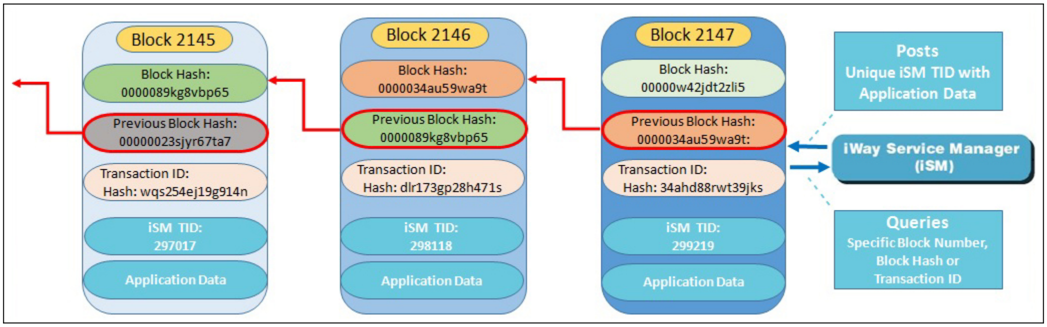
The following is an outline of the asynchronous posting process:

- Runs in the process flow (pflow) as part of its logic.
- iSM standard pflow services and transformers form the payload (transaction).
- The posting service (under the covers) handles the two-way communication within the network peers.
    - Posts the transaction to multiple peers.
    - Peer chaincode validates the transaction against the smart contract.
    - Chaincode sends results back to the posting service.
    - iSM internal logic can perform first level validation of results (allows performance improvements).
    - Upon validation by iSM, the peer results are sent to the Orderer.

Later in time...

- ❑ Fabric's Orderer
  - ❑ Ensures the transaction across the networked peers is consistent.
  - ❑ Has ultimate control of the network.
  - ❑ Emits an event to be trapped by iSM.
- ❑ iSM Fabric Asynchronous Response Handler
  - ❑ The Hyperledger Fabric listener receives the event from the event hub, formats it in XML or JSON, and sends it to an iSM Internal Queue for handling.

The following diagram illustrates how iSM posts a unique iSM TID (Transaction ID) along with application data for that transaction within a block.



The result of the selected query is returned to iSM from the blockchain for transaction analysis.

Understanding the Internal Queue

Standard configuration of the Internal Queue is described in the *iWay Cross-Channel Services Guide*. The only caveat is that you must not enable inhibit mode for the Internal Queue. Instead, iWay recommends configuring the Internal Queue to send a passivation throttle message to the posting channel should a high water mark be reached.

See the *iWay Cross-Channel Services Guide* for more information on configuring an Internal Queue, including the use of persistence to prevent loss of messages in the event that the server is shut down before all response messages have been handled.

The service offers specific asynchronous component configuration, including the name of the Internal Queue to receive the response. The Internal Queue listener can use configured routing services to select the proper flow for the specific message, or can use a single flow that differentiates the messages within the flow.

The message that reaches the Internal Queue for processing includes the time of posting, the parameters of the posting service, the name of the posting service, and so on. The response flow runs under the same iSM Transaction ID (TID) as that of the posting flow.

Query Service

This service invokes chaincode to make a read-only query in the Hyperledger Fabric. The service sends a query proposal request to all the configured peers, then it extracts the response payload from the first successful proposal response. The payload is a single ByteString value, but it can also be parsed by the service in XML or JSON, assuming the chaincode produced the result in this format.

For more information and an example, see *Hyperledger Fabric Query Service* on page 26.

Query Block Service

This service queries the Hyperledger Fabric for a specific block and returns it. The search criteria can be a block number, a block hash or a Transaction ID. The result is a BlockInfo response encoded in an XML or JSON parsed tree.

For more information and an example, see *Hyperledger Fabric Query Block Service* on page 29.

Fabric Listener

The Hyperledger Fabric listener connects to the event hubs and listens for block events. The listener returns the BlockEvent encoded in an XML or JSON parsed tree. In the Hyperledger Fabric, a BlockEvent is an abbreviated view of a BlockInfo with the TxReadWriteSetInfo missing but the event hub origin added. If necessary, the complete BlockInfo can be retrieved by BlockNumber later in a process flow with the help of the Hyperledger Fabric Query Block Service.

This listener does not interact with the asynchronous post requests, which are fielded by the provider and handed off to the configured Internal Queue.

For more information and an example, see *Hyperledger Fabric Listener* on page 42.

Chapter 2

Hyperledger Fabric Component Reference

This chapter provides a reference for predefined Hyperledger Fabric components that are available in iWay Service Manager (iSM).

In this chapter:

Hyperledger Fabric Channel Provider

Description:

The Hyperledger Fabric Channel provider holds the configuration to reconstruct a Hyperledger Fabric Channel on the client side. Other components can refer to the provider by name to gain access to the Channel. For example, Hyperledger Fabric Services use the Channel to execute transactions or make queries, and the Hyperledger Fabric listener uses the Channel to listen for events coming from the event hubs. Multiple components can share the same provider safely.

This provider can be created within the server tab of the console by clicking on the Fabric Channel Provider link in the left menu.

Parameters:

The following tables list and describe the parameters for the Hyperledger Fabric Channel provider.

User

Parameter	Description
User Name	The user must have been previously registered and enrolled in fabric-ca (or an equivalent member service).
MSPID	Membership service provider identifier.
Enrollment Certificate	Path to the enrollment certificate for that user, see: `$FABRIC_CA_CLIENT_HOME/msp/signcerts/cert.pem`
Enrollment Private Key	Path to the enrollment private key in PEM format for that user, see: `$FABRIC_CA_CLIENT_HOME/msp/keystore/key.pem`

The *User* group of parameters define the user credentials. The MSPID is chosen when the user is registered with Fabric CA (Certificate Authority), or an equivalent member service. The MSPID is usually an organization name, such that all users of a single organization share the same MSPID. The certificate and private key are generated when the user is enrolled. The provider requires the certificate and private key to be stored in PEM files. This is the same format produced by the command line utility fabric-ca-client when enrolling a user.

Fabric

Parameter	Description
Channel Name	The name of the channel.
Peer Endpoints	Comma separated list of peer definitions in the form `peerName@url`. For example: `peer0@grpc://host:7051`
Orderer Endpoints	Comma separated list of orderer definitions in the form `ordererName@url`. For example: `orderer0@grpc://localhost:7050`

Fabric	
Parameter	**Description**
Event Hub Endpoints	Comma separated list of event hub definitions in the form `eventHubName@url`. For example: `peer0@grpc://localhost:7053`
Transaction Wait Time	Transaction wait time, which uses the format of `[xxh][xxm]xx[s]`. For example, 1m30s is 90 seconds.

The *Fabric* group of parameters define the channel. If a list of endpoints becomes large, it might be more convenient to store it in a file and load it with the *_FILE(filepath)* function. In this case, each endpoint can appear on a separate line. Inside the file, a line starting with the # character is a comment and is ignored to the end of line.

Example:

The following example shows the parameter values that are used to reconstruct the channel in the End2endIT sample of the Hyperledger Fabric Java SDK. Replace *host.com* with the actual host name.

Parameter	Value
User Name	`user1`
MSPID	`Org1MSP`
Enrollment Certificate	`user1/cert.pem`
Enrollment Private Key	`user1/key.pem`
Channel Name	`foo`
Peer Endpoints	`peer0@grpc://host.com:7051`
Orderer Endpoints	`orderer0@grpc://host.com:7050`
Event Hub Endpoints	`peer0@grpc://host.com:7053`

Parameter	Value
Transaction Wait Time	`20`

In the example above, the file path *user1/cert.pem* must point to a file that contains the enrollment certificate for user1 as generated by the Membership Services Provider (i.e. fabric-ca). This is a sample enrollment certificate file:

```
-----BEGIN CERTIFICATE-----
MIIB8TCCAZegAwIBAgIUYe9xCEzh9YwhriG1YZnj4V9rYYMwCgYIKoZIzj0EAwIw
czELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh
biBGcmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMT
E2NhLm9yZzEuZXhhbXBsZS5jb20wHhcNMTcwNzI0MTYwNjAwWhcNMTgwNzI0MTYw
NjAwWjAQMQ4wDAYDVQQDEwV1c2VyMTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IA
BFq/22WC5z/yNt3GMyUVFLeQr4YsSMe4gnSbpUItTVjHLnz77kOmbdmgMQSoIZNm
jKL/gb3+j5fBfSe3A2TEwP+jbDBqMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8E
AjAAMB0GA1UdDgQWBBQMTqi2/YZgcz40dDqj74aBGTsNODArBgNVHSMEJDAigCD8
93awKgVgBAjQvp2XUq/Fn2SVC3IcrLNjtblaD+piFjAKBggqhkjOPQQDAgNIADBF
AiEArZi+gu56MY/VoKhXUOAhQ3qI8EO7KjduZEKi+C3LA+8CIDLcPlnFzuZ2jNnF
+GiSAA2BE6qc8kd0YHWNSnf8GrEd
-----END CERTIFICATE-----
```

In the example above, the file path *user1/key.pem* must point to a file that contains the private key for user1 as generated by the Membership Services Provider (for example, fabric-ca). The following is a sample private key file:

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIHMjJth72UerCb2S6UH2IyDcF7T4ZszKeA9+uGXGAEXxoAoGCCqGSM49
AwEHoUQDQgAEWr/bZYLnP/I23cYzJRUUt5CvhixIx7iCdJulQi1NWMcufPvuQ6Zt
2aAxBKghk2aMov+Bvf6Pl8F9J7cDZMTA/w==
-----END EC PRIVATE KEY-----
```

Hyperledger Fabric Transaction Service

Syntax:

```
com.ibi.agents.XDFabricAgent
```

Description:

This service invokes chaincode to execute a transaction on a blockchain in the Hyperledger Fabric. It starts by sending a transaction proposal request to all the configured peers. If enough proposal responses are successful (and not too many have failed), then the transaction is sent to the configured Orderers.

In synchronous mode, the service waits for the transaction event from the configured event hubs to make sure the transaction was committed to the blockchain. In asynchronous mode, the service returns immediately because a message is sent to an internal queue when the event finally arrives. An internal queue listener must be configured to read that internal queue to continue processing after the confirmation event.

The TransactionID is stored in the Special Register (SREG) named *fabric.TransactionID*. Upon error, the output is an error document. When successful in synchronous mode, the output document can be the input document, the chaincode return value, or the TransactionEvent. When successful in asynchronous mode, the output document is the input document.

Parameters:

The following tables list and describe the parameters for the Hyperledger Fabric Transaction service.

Main	
Fabric Channel Provider	The name of the configured Hyperledger Fabric Channel provider.
Chaincode Name	The name of the chaincode to call.
Chaincode Path	The path of the chaincode to call.
Chaincode Version	The version of the chaincode to call.
Minimum Successful Proposals	Minimum number of successful proposals required to send the transaction to the Orderers.
Maximum Failed Proposals	Maximum number of failed proposals tolerated. The transaction will not be sent to the Orderers if there are more failed proposals.
Return	Determines the contents of the output document. Ignored if *asynchronous mode* is selected.

In the *Main* group of parameters, the service refers to the Fabric Channel Provider by name to gain access to the reconstructed Channel. The chaincode name, path and version are determined when the chaincode is installed. The path is similar to a namespace and is not related to a path in the file system. The chaincode must already be installed before calling the Hyperledger Fabric Transaction service.

The Hyperledger Fabric Transaction service has a simple acceptance criterion to decide if the transaction is sent to the Orderers to be committed to the blockchain. There must be a minimum number of successful proposals and the number of failed proposals cannot exceed a maximum. This simple criterion is used by the service as an optimization. It does not have to be perfect. The real endorsement policy is evaluated by the Orderers when creating the block.

The Return parameter determines the contents of the output document in synchronous mode. The output document can be the input document, the chaincode return value, or the TransactionEvent, The chaincode return value is a ByteString, which can be parsed in different ways. The chaincode return value can be:

❑ Interpreted as a UTF8 string.

❑ Encoded as a hexadecimal string with two consecutive hexadecimal characters per byte.

❑ Encoded as a string in Base64.

❑ Parsed as XML or JSON, and returned as a parsed tree.

The TransactionEvent is an object that can be serialized in XML or JSON, and returned as a parsed tree. In asynchronous mode, the Return parameter is ignored because the transaction is not yet complete. In this mode, the output document is always the input document unless there is an error.

Call	
Argument Type	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte.
Argument 0	The value of the first argument.
Argument 1	The value of the second argument.
Argument 2	The value of the third argument.
Argument 3	The value of the fourth argument.
Argument 4	The value of the fifth argument.

In the *Call* group of parameters, the Argument *N* properties are the arguments to the chaincode invocation. By convention, Argument 0 is often an operation name understood by the chaincode. There can be an unlimited number of arguments. This is done by defining user properties with names of the form *argN*. For example, the next argument after Argument 4 is *arg5*. The list of arguments stops at the first empty argument.

The type of the arguments can be string or byte array. All arguments must be the same type. A byte array is encoded as an hexadecimal string with two consecutive hexadecimal characters per byte. For example, 616263 is abc in a UTF8 byte array.

Asynchronous	
Is Asynchronous	When selected, the input document is returned immediately after the proposal request. The TransactionEvent response is sent to an internal queue when it arrives.
Queue Name	Name of the internal queue where the asynchronous TransactionEvent will be sent.
Event Format	Format of the document sent to the internal queue, XML or JSON.
Want User Registers	User registers are processing variables and their values. If you want these registers to be emitted with the message set this to true. In resubmit operations, usually this is set to false.
Priority	The priority is an integer between 0 and 9 inclusive. The lowest priority is 0. The highest most expedited priority is 9.
Put Timeout	Time, in milliseconds, to wait for queue to become available when attempting to put a message on the queue. You can enter 0 for an unlimited wait, but this is not recommended. If no value is supplied, timeout will be set to 3000 milliseconds.

Asynchronous	
Request Context Namespace	For synchronous or asynchronous processing, namespace or list of namespaces containing registers that will be made available to the internal listener. Select *Default* for all registers in default (no prefix) namespace or *None* to send no registers at all. Enter an asterisk character (*) for registers from all namespaces.

Asynchronous mode is enabled by selecting the *Is Asynchronous* parameter. The Queue Name parameter tells the service where the message will be sent when the TransactionEvent arrives. The Event Format chooses the format of the document, either XML or JSON. Want User Registers determines if the User registers are preserved with the message. The priority of the message affects in which order the messages are processed by the internal queue listener. The Put Timeout determines how long the callback will wait for the queue to become available when sending the message. After this timeout, the message is dropped.

The Request Context Namespace selects which registers will be preserved with the message. For example, if the value is *ns*, then all registers with names beginning with *ns.* (such as *ns.reg1*) are preserved. Select *Default* to preserve registers with no prefix. Select *None* to ignore all registers. Enter an asterisk character (*) to select all registers.

Edges:

The following table lists and describes the edges that are returned by the Hyperledger Fabric Transaction service.

Edge	Description
success	The operation was successful.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example 1:

Example 1 shows the parameter values for an invocation of the chaincode in the End2endIT sample of the Hyperledger Fabric Java SDK. It assumes that the *fabric1* provider has the same values as the example shown in *Hyperledger Fabric Channel Provider* on page 15.

Parameter	Value
Fabric Channel Provider	`fabric1`
Chaincode Name	`example_cc_go`
Chaincode Path	`github.com/example_cc`
Chaincode Version	`1`
Argument Type	`String`
Argument 0	`move`
Argument 1	`a`
Argument 2	`b`
Argument 3	`20`
Minimum Successful Proposals	`1`
Maximum Failed Proposals	`0`

Example 2:

Example 2 shows that the same invocation can be achieved with the chaincode arguments encoded as byte arrays. The iFL function *_hex(value,charset)* might be useful.

Parameter	Value
Fabric Channel Provider	`fabric1`
Chaincode Name	`example_cc_go`

Parameter	Value
Chaincode Path	`github.com/example_cc`
Chaincode Version	`1`
Argument Type	`Hex String`
Argument 0	`_hex('move', 'UTF8')`
Argument 1	`61`
Argument 2	`62`
Argument 3	`3230`
Minimum Successful Proposals	`1`
Maximum Failed Proposals	`0`

Example 3:

Example 3 shows the extra properties that can be added to example 1 or 2 to enable asynchronous mode. In this mode, the service will return immediately after sending the transaction to the Orderers. A message will be sent to the internal queue *q1* when the transaction event arrives, confirming the transaction is part of the blockchain.

Parameter	Value
Is Asynchronous	`true`
Queue Name	`q1`
Event Format	`XML`
Want User Registers	`true`
Priority	`4`

Parameter	Value
Put Timeout	`3000`
Request Context Namespace	`*`

The following is a sample output document when the Return parameter is *TransactionEvent as XML* in synchronous mode. A message similar to this is sent to the Internal queue in asynchronous mode when the Event Format is *XML*. This document has been indented for display purposes only.

```
<TransactionEvent>
  <EventHub>
     <Name>peer0</Name>
     <Url>grpc://thor.ibi.com:7053</Url>
  </EventHub>
  <TransactionID>90fdd06ad2752deaed7c9beab00c46bd770667e...</TransactionID>
  <ChannelID>foo</ChannelID>
  <Epoch>0</Epoch>
  <Timestamp>2017-09-26T12:51:32.138-04:00</Timestamp>
  <IsValid>true</IsValid>
  <ValidationCode>0</ValidationCode>
  <TransactionActionInfos>
    <TransactionActionInfo>
      <ResponseStatus>200</ResponseStatus>
      <ResponseMessageBytes/>
      <EndorserInfos>
        <EndorserInfo>
          <Signature>30440220720e51c11b98c17b72ab9a50fbb6...</Signature>
          <Endorser>0a074f7267314d53501280062d2d2d2d2d424...</Endorser>
        </EndorserInfo>
      </EndorserInfos>
      <ChaincodeInputArgs>
        <Arg>696e766f6b65</Arg>
        <Arg>6d6f7665</Arg>
        <Arg>61</Arg>
        <Arg>62</Arg>
        <Arg>3230</Arg>
      </ChaincodeInputArgs>
      <ProposalResponseStatus>200</ProposalResponseStatus>
      <ProposalResponsePayload/>
    </TransactionActionInfo>
  </TransactionActionInfos>
</TransactionEvent>
```

The following is a sample output document when the Return parameter is *TransactionEvent as JSON* in synchronous mode. A message similar to this is sent to the Internal queue in asynchronous mode when the Event Format is *JSON*. The document has been indented for display purposes only.

```
{
  "EventHub":{"Name":"peer0",
  "Url":"grpc://thor.ibi.com:7053"},
  "TransactionID":"49fa1193bc30c17cc011a6b5850f8e57c92...",
  "ChannelID":"foo",
  "Epoch":0,
  "Timestamp":"2017-09-26T12:54:56.689-04:00",
  "IsValid":true,
  "ValidationCode": 0,
  "TransactionActionInfos": [
    {
      "ResponseStatus": 200,
      "ResponseMessageBytes": "",
      "EndorserInfos": [
        {
          "Signature": "3045022100e5181d03460959d751...",
          "Endorser": "0a074f7267314d53501280062d2d2..."
        }
      ],
      "ChaincodeInputArgs": [
        "696e766f6b65",
        "6d6f7665",
        "61",
        "62",
        "3230"
      ],
      "ProposalResponseStatus": 200,
      "ProposalResponsePayload": ""
    }
  ]
}
```

Hyperledger Fabric Query Service

Syntax:

`com.ibi.agents.XDFabricQueryAgent`

Description:

This service invokes chaincode to make a read-only query in the Hyperledger Fabric. It sends a query proposal request to all the configured peers, then it extracts the response payload from the first successful proposal response. The payload is a single ByteString value, but it can also be parsed by the service in XML or JSON, assuming the chaincode produced the result in this format.

Parameters:

The following tables list and describe the parameters for the Hyperledger Fabric Query service.

Main	
Fabric Channel Provider	The name of the configured Hyperledger Fabric Channel provider.
Chaincode Name	The name of the chaincode to call.
Chaincode Path	The path of the chaincode to call.
Chaincode Version	The version of the chaincode to call.
Return Value Format	Determines how the ByteString response should be interpreted.

The service refers to the Fabric Channel Provider by name to gain access to the reconstructed Channel. The chaincode name, path and version are determined when the chaincode is installed. The path is similar to a namespace and is not related to a path in the file system. The chaincode must already be installed before calling the Hyperledger Fabric Query service.

The response is a ByteString value that can be parsed in different ways depending on the *Return Value Format* parameter. It can be:

❑ Interpreted as a UTF8 string.

❑ Encoded as a Hexadecimal string with two consecutive hexadecimal characters per byte.

❑ Encoded as a string in Base64.

❑ Parsed as XML or JSON and returned as a parsed tree.

Call	
Argument Type	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte.
Argument 0	The value of the first argument.
Argument 1	The value of the second argument.
Argument 2	The value of the third argument.

Call	
Argument 3	The value of the fourth argument.
Argument 4	The value of the fifth argument.

In the *Call* group of parameters, the Argument *N* properties are the arguments to the chaincode invocation. By convention, Argument 0 is often an operation name understood by the chaincode. There can be an unlimited number of arguments. This is done by defining user properties with names of the form *argN*. For example, the next argument after Argument 4 is *arg5*. The list of arguments stops at the first empty argument.

The type of the arguments can be string or byte array. All arguments must be the same type. A byte array is encoded as an hexadecimal string with two consecutive hexadecimal characters per byte. For example, 616263 is abc in a UTF8 byte array.

Edges:

The following table lists and describes the edges that are returned by the Hyperledger Fabric Query service.

Edge	Description
success	The operation was successful.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example:

The following example shows the parameter values for an invocation of the chaincode to query the value of *b* in the End2endIT sample of the Hyperledger Fabric Java SDK. It assumes that the *fabric1* provider has the same values as the example shown in *Hyperledger Fabric Channel Provider* on page 15.

Parameter	Value
Fabric Channel Provider	`fabric1`
Chaincode Name	`example_cc_go`

Parameter	Value
Chaincode Path	`github.com/example_cc`
Chaincode Version	`1`
Return Value Format	`UTF8 String`
Argument Type	`String`
Argument 0	`query`
Argument 1	`b`

The same invocation can be achieved with the chaincode arguments encoded as byte arrays. The iFL function *_hex(value,charset)* might be useful.

Parameter	Value
Argument Type	`Hex String`
Argument 0	`_hex('query', 'UTF8')`
Argument 1	`62`

Hyperledger Fabric Query Block Service

Syntax:

```
com.ibi.agents.XDFabricQueryBlockAgent
```

Description:

This service queries Hyperledger Fabric for a specific block and returns it. The query will be sent to one of the peers chosen randomly. The search criteria can be a block number, a block hash, or a Transaction ID. The result is a BlockInfo encoded in an XML or JSON parsed tree.

In batch mode, the blocks previous to the queried block can also be returned. The number of blocks can be limited by the Batch Size or a date limit.

Parameters:

The following table lists and describes the parameters for the Hyperledger Fabric Query Block service.

Main	
Fabric Channel Provider	The name of the configured Hyperledger Fabric Channel provider.
Query Type	Determines the search criteria for the block query.
Query Argument	Querying by block number: Supply a non-negative integer block number, or a negative integer block number relative to the current block height (-1 is the latest block). Querying by block hash: This is a hex encoded string with two hex characters per byte. When querying by transaction ID, this is the transaction ID.
Output Format	Format of the result.

The service refers to the Fabric Channel Provider by name to gain access to the reconstructed Channel.

The Query Type determines what kind of search criteria is provided in the Query Argument:

- It can be the exact non-negative block number.
- It can also be a negative integer relative to the block height. Here the value -1 refers to the latest block, -2 refers to the block before the last, and so on. Notice the blockchain can continue to grow after the relative block number is computed before the query is executed.
- The third Query Type is a block hash. This is a byte array encoded as a hexadecimal string with two hexadecimal characters per byte. This value can be obtained from the PreviousHash of the next block for example. Be aware the DataHash is not the same as a block hash. The block hash is computed from the block number, the data hash and the previous hash and does not appear in the BlockInfo.
- Finally, the Query Type can a Transaction ID. This is a regular String. The Hyperledger Fabric Transaction Service stores the Transaction ID in the fabric.TransactionID special register when successful. The value of that register can be used to query the corresponding block.

The Output Format can be XML or JSON. The output document stores the returned BlockInfo or the batch of blocks encoded in an XML or JSON parsed tree.

Batch Mode	
Early Limit	Format is `yyyy-MM-dd'T'HH:mm:ss.SSS`*XXX*, where *XXX* is an ISO 8601 time zone. Specifying an Early Limit turns on batch mode, which will always return the queried block and also includes the previous blocks up to, but not including, the first encountered block with all transactions timestamped earlier than the early limit. Timestamps are created by clients without a central clock. It is therefore recommended to specify a date earlier than necessary by some margin. The batch is also limited by the Batch Size when defined.
Batch Size	Specifying a batch size turns on batch mode, which will return up to that many blocks, starting from the queried block and its previous blocks, within the limits of the Early Limit if defined, or the origin of the chain.

Batch mode is enabled by specifying any of the Batch Mode properties. In batch mode, the queried block is always returned plus zero or more previous blocks. The Batch Size parameter is the maximum number of blocks that can be returned. The Early Limit parameter selects the previous blocks that have at least one transaction timestamped on or after the specified date. If both parameters are specified, then the batch stops with the first constraint satisfied.

Edges:

The following table lists and describes the edges that are returned by the Hyperledger Fabric Block service.

Edge	Description
success	The operation was successful.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Examples:

The following example shows the parameter values to query the latest block in the End2endIT sample of the Hyperledger Fabric Java SDK. It assumes that the *fabric1* provider has the same values as the example shown in *Hyperledger Fabric Channel Provider* on page 15.

Parameter	Value
Fabric Channel Provider	`fabric1`
Query Type	`Query by Block Number`
Query Argument	`-1`
Query Format	`XML`

The following example shows how to query the previous block assuming the input document is the XML output of a previous call to the Hyperledger Fabric Query Block service.

Parameter	Value
Fabric Channel Provider	`fabric1`
Query Type	`Query by Block Hash`
Query Argument	`_xpath(/BlockInfo/PreviousHash)`
Query Format	`XML`

The following example shows how to query the block corresponding to the transaction committed by the latest invocation of the Hyperledger Fabric Transaction Service

Parameter	Value
Fabric Channel Provider	`fabric1`
Query Type	`Query by Transaction ID`
Query Argument	`_sreg('fabric.TransactionID')`

Parameter	Value
Query Format	XML

The following is a sample output in XML format, which has been indented for display purposes only.

```
<BlockInfo>
  <BlockNumber>46</BlockNumber>
<DataHash>26a1b666a83679a3b8fbaf16149c1b0b01e424956fd1ba6d3...</DataHash>
<PreviousHash>ce738d39292304e98289e43515c4ae91cd649421f...</PreviousHash>
  <EnvelopeInfos>
    <EnvelopeInfo>
<TransactionID>4f8d8728ffbf27adf21b32f7125de6e5929323d...</TransactionID>
      <ChannelID>foo</ChannelID>
      <Epoch>0</Epoch>
      <Timestamp>2017-07-28T17:15:52-04:00</Timestamp>
      <IsValid>true</IsValid>
      <ValidationCode>0</ValidationCode>
      <TransactionActionInfos>
        <TransactionActionInfo>
          <ResponseStatus>200</ResponseStatus>
          <ResponseMessageBytes/>
          <EndorserInfos>
            <EndorserInfo>
<Signature>3044022022dcc5424b0d0b50f3ad3cdec8ad7cbda7524fg...</Signature>
<Endorser>0a074f7267314d53501280062d2d2d2d2d42494e202d2d2d...</Endorser>
            </EndorserInfo>
```

```
</EndorserInfos>
<ChaincodeInputArgs>
  <Arg>696e766f6b65</Arg>
  <Arg>6d6f7665</Arg>
  <Arg>61</Arg>
  <Arg>62</Arg>
  <Arg>31</Arg>
</ChaincodeInputArgs>
<ProposalResponseStatus>200</ProposalResponseStatus>
<ProposalResponsePayload/>
<TxReadWriteSetInfo>
  <NsRwsetInfos>
    <NsRwsetInfo>
      <Namespace>example_cc_go</Namespace>
      <KVReads>
        <KVRead><Key>a</Key>
          <Version>
            <BlockNum>45</BlockNum>
            <TxNum>0</TxNum>
          </Version>
        </KVRead>
        <KVRead>
          <Key>b</Key>
          <Version>
            <BlockNum>45</BlockNum>
            <TxNum>0</TxNum>
          </Version>
        </KVRead>
      </KVReads>
      <KVWrites>
        <KVWrite>
          <Key>a</Key>
          <Value>313931</Value>
        </KVWrite>
        <KVWrite>
          <Key>b</Key>
          <Value>353039</Value>
        </KVWrite>
      </KVWrites>
    </NsRwsetInfo>
    <NsRwsetInfo>
      <Namespace>lscc</Namespace>
      <KVReads>
        <KVRead>
          <Key>example_cc_go</Key>
          <Version>
            <BlockNum>1</BlockNum>
```

```
                      <TxNum>0</TxNum>
                    </Version>
                  </KVRead>
                </KVReads>
              </NsRwsetInfo>
            </NsRwsetInfos>
          </TxReadWriteSetInfo>
        </TransactionActionInfo>
      </TransactionActionInfos>
    </EnvelopeInfo>
  </EnvelopeInfos>
</BlockInfo>
```

The following is a sample output in JSON format, which has been indented for display purposes only. Notice that arrays are more natural in JSON.

```
{
  "BlockNumber": 46,
  "DataHash": "26a1b666a83679a3b8fbaf16149c1b0b01e424956fd1ba6d3ec5sf...",
  "PreviousHash": "ce738d39292304e98289e43515c4ae91cd649421f96813e8af...",
  "EnvelopeInfos": [
    {
      "TransactionID": "4f8d8728ffbf27adf21b32f7125de6e5929323d3a6a37c...",
      "ChannelID": "foo",
      "Epoch": 0,
      "Timestamp": "2017-07-28T17:15:52-04:00",
      "IsValid": true,
      "ValidationCode": 0,
      "TransactionActionInfos": [
        {
          "ResponseStatus": 200,
          "ResponseMessageBytes": "",
          "EndorserInfos": [
            {
              "Signature":
"3044022022dcc5424b0d0b50f3ad3cdec8ad7cbda7gs...",
              "Endorser":
"0a074f7267314d53501280062d2d2d2d2d40128006505d..."
```

```
  }
],
"ChaincodeInputArgs": [
  "696e766f6b65",
  "6d6f7665",
  "61",
  "62",
  "31"
],
"ProposalResponseStatus": 200,
"ProposalResponsePayload": "",
"TxReadWriteSetInfo": {
  "NsRwsetInfos": [
    {
      "Namespace": "example_cc_go",
      "KVReads": [
        {
          "Key": "a",
          "Version": {
            "BlockNum": 45,
            "TxNum": 0
          }
        },
        {
          "Key": "b",
          "Version": {
            "BlockNum": 45,
            "TxNum": 0
          }
        }
      ],
      "KVWrites": [
        {
          "Key": "a",
          "Value": "313931"
        },
        {
          "Key": "b",
          "Value": "353039"
        }
      ]
    },
              {
                "Namespace": "lscc",
                "KVReads": [
                  {
                    "Key": "example_cc_go",
                    "Version": {
                      "BlockNum": 1,
                      "TxNum": 0
                    }
                  }
                ]
              }
            ]
          }
        }
      ]
    }
  ]
}
```

The following is an abbreviated sample output for batch mode containing two blocks in XML format. The document has been indented for display purposes only. See the sample above for the contents of a single BlockInfo element.

```
<BlockInfos>
      <BlockInfo>
          <BlockNumber>46</BlockNumber>
          ...
      </BlockInfo>
      <BlockInfo>
          <BlockNumber>45</BlockNumber>
    ...
      </BlockInfo>
</BlockInfos>
```

The following is an abbreviated sample output for batch mode containing two blocks in JSON format. The document has been indented for display purposes only. In JSON, the batch of blocks is an array of BlockInfo objects. See the sample above for the contents of a single BlockInfo object.

```
[
  {
      "BlockNumber": 46,
      ...
        },
  {
      "BlockNumber": 45,
      ...
        }
]
```

Hyperledger Fabric Key History Service

Syntax:

```
com.ibi.agents.XDFabricKeyHistoryAgent
```

Description:

This service queries the Hyperledger Fabric for the history of a single key showing the assignments going back in time. The report starts from a given key version and continues with previous versions, provided the transaction that wrote the key also read the previous value. Technically, this means the key must also appear in the KVReads when it appears in the KVWrites of the transaction.

Parameters:

The following table describes the parameters of the Hyperledger Fabric Key History service.

Main	
Fabric Channel Provider	The name of the configured Hyperledger Fabric Channel provider.
Namespace	Namespace of the key, this is usually the chaincode name.
Key	Name of the key.
Block Number	The number of a block containing a transaction with a write of the key. This is a non negative integer block number, or a negative integer block number relative to the current block height (-1 is the latest block).
Transaction Number	The index of the transaction within the selected block.
Encoding	Determines how the value will be converted to string. Choose UTF-8 or type a Charset name to convert the bytes to string, or choose Hex to convert the ByteString to a hex encoded string with two hex characters per byte.

Main	
Early Limit	Format is `yyyy-MM-dd'T'HH:mm:ss.SSS`*`XXX`* where *`XXX`* is an ISO 8601 time zone. Limit the key history depth to key writes timestamped earlier than the early limit. Timestamps are created by clients without a central clock. It is therefore recommended to specify a date earlier than necessary by some margin.
Maximum Depth	The maximum number of versions of the key to report.
History Format	Format of the history in the output document.

The service refers to the Fabric Channel Provider by name to gain access to the reconstructed Channel.

A key is private to a chaincode instance. The Namespace specifies which chaincode owns the key. The key version to start the trace is given by the Block Number and the Transaction Number within that block. The Block Number can be a non-negative integer as usual, or or a negative integer block number relative to the current block height (-1 is the latest block).

The value of a key is a ByteString. The Encoding parameter specifies how it will be converted to a string in the history. For example, the bytes can be converted to string according to the rules of a specified Charset, like the suggested UTF-8 or any other Charset name typed in the field. The other option is to convert to a Hex String with two hex characters per byte. This is the default.

The Maximum Depth limits the size of the history if the origin block is not reached first. The Early Limit also limits the size of the history to the transactions that are timestamped on or after the specified date.

The History Format can be XML or JSON. The output document stores the key history encoded in an XML or JSON parsed tree.

Edges:

The following table lists and describes the edges that are returned by the Hyperledger Fabric Key History service.

Edge	Description
success	The operation was successful.

Edge	Description
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example:

This example shows how to query the history of the a key maintained by the chaincode *example_cc_go*.

Parameter	Value
Fabric Channel Provider	`fabric1`
Namespace	`example_cc_go`
Key	`a`
Block Number	`-1`
Transaction Number	`0`
Encoding	`UTF-8`
Maximum Depth	`5`
History Format	`XML`

The following is a sample output in XML format, which has been indented for display purposes only.

```
<KVHistory>
  <Namespace>example_cc_go</Namespace>
  <Key>a</Key>
 <KVPuts>
   <KVPut>
     <Value>300</Value>
     <BlockNum>116</BlockNum>
     <TxNum>0</TxNum>
   </KVPut>
   <KVPut>
     <Value>325</Value>
     <BlockNum>115</BlockNum>
     <TxNum>0</TxNum>
   </KVPut>
   <KVPut>
     <Value>350</Value>
     <BlockNum>114</BlockNum>
     <TxNum>0</TxNum>
   </KVPut>
   <KVPut>
     <Value>375</Value>
     <BlockNum>113</BlockNum>
     <TxNum>0</TxNum>
   </KVPut>
   <KVPut>
     <Value>400</Value>
     <BlockNum>112</BlockNum>
     <TxNum>0</TxNum>
   </KVPut>
 </KVPuts>
</KVHistory>
```

The following is a sample of the same history in JSON format, which has been indented for display purposes only.

```
{
"Namespace":"example_cc_go",
"Key":"a",
"KVPuts":
 [
   {
     "Value":"300",
     "BlockNum":116,
     "TxNum":0},
   {
     "Value":"325",
     "BlockNum":115,
     "TxNum":0},
   {
     "Value":"350",
     "BlockNum":114,
     "TxNum":0},
   {
     "Value":"375",
     "BlockNum":113,
     "TxNum":0},
   {
     "Value":"400",
    "BlockNum":112,
    "TxNum":0}
 ]
}
```

Hyperledger Fabric Listener

Syntax:

```
com.ibi.edaqm.XDFabricMaster
```

Description:

The Hyperledger Fabric listener connects to the event hubs and listens for block events. The listener returns the BlockEvent encoded in an XML or JSON parsed tree. In Hyperledger Fabric, a BlockEvent is an abbreviated view of a BlockInfo with the TxReadWriteSetInfo missing but the event hub origin added. If necessary, the complete BlockInfo can be retrieved by BlockNumber later with the help of the Hyperledger Fabric Query Block Service.

Parameters:

The following table lists and describe the parameter for the Hyperledger Fabric listener.

Parameter	Description
Fabric Channel Provider	The name of the configured Hyperledger Fabric Channel provider.

The listener refers to the Fabric Channel Provider by name to gain access to the reconstructed channel.

Examples:

The following is a sample Bloc kEvent in XML format, which has been indented for display purposes only:

```
<BlockEvent>
  <EventHub>
    <Name>peer0</Name>
    <Url>grpc://thor.ibi.com:7053</Url>
  </EventHub>
  <BlockNumber>105</BlockNumber>
<DataHash>bfd22913e1d50e98ada53a9b8ee28936a661bc5bdc25bb23a...</DataHash>
<PreviousHash>b12c299d2bc1ac7866c6f17d90c8b6ec6fd47c415...</PreviousHash>
  <EnvelopeInfos>
    <EnvelopeInfo>
<TransactionID>866054c0ef3cafafd22d4f3acf9bb3dc1054c32...</TransactionID>
      <ChannelID>foo</ChannelID>
      <Epoch>0</Epoch>
      <Timestamp>2017-08-02T16:52:15-04:00</Timestamp>
      <IsValid>true</IsValid>
      <ValidationCode>0</ValidationCode>
      <TransactionActionInfos>
        <TransactionActionInfo>
          <ResponseStatus>200</ResponseStatus>
          <ResponseMessageBytes/>
          <EndorserInfos>
            <EndorserInfo>
<Signature>304502210097bc7cae110c25ed7743e99043970271aa59e...</Signature>

<Endorser>642874198347247568317641736457125316523417265383 4...</Endorser>
            </EndorserInfo>
          </EndorserInfos>
          <ChaincodeInputArgs>
            <Arg>696e766f6b65</Arg>
            <Arg>6d6f7665</Arg>
            <Arg>61</Arg>
            <Arg>62</Arg>
            <Arg>31</Arg>
          </ChaincodeInputArgs>
          <ProposalResponseStatus>200</ProposalResponseStatus>
          <ProposalResponsePayload/>
        </TransactionActionInfo>
      </TransactionActionInfos>
    </EnvelopeInfo>
  </EnvelopeInfos>
</BlockEvent>
```

The following is the same BlockEvent in JSON format, which has been indented for display purposes only. Notice that arrays are more natural in JSON.

```
{
  "EventHub": {
    "Name": "peer0",
    "Url": "grpc:\/\/thor.ibi.com:7053"
  },
  "BlockNumber": 105,
  "DataHash": "bfd22913e1d50e98ada53a9b8ee28936a661bc5bdc25bb23a5a76...",
  "PreviousHash": "b12c299d2bc1ac7866c6f17d90c8b6ec6fd47c415b7161e8f...",
  "EnvelopeInfos": [
    {
      "TransactionID": "866054c0ef3cafafd22d4f3acf9bb3dc1054c32874892a...",
      "ChannelID": "foo",
      "Epoch": 0,
      "Timestamp": "2017-08-02T16:52:15-04:00",
      "IsValid": true,
      "ValidationCode": 0,
      "TransactionActionInfos": [
        {
          "ResponseStatus": 200,
          "ResponseMessageBytes": "",
          "EndorserInfos": [
            {
              "Signature":
"304502210097bc7cae110c25ed7743e99043970271aa59e7b94c83d7bef56532abd...",
              "Endorser":
"0a074f7267314d53501280062d2d2d2d424547494e202d2d2d2d2d0a4d4949434d..."
            }
          ],


"ChaincodeInputArgs": [
            "696e766f6b65",
            "6d6f7665",
            "61",
            "62",
            "31"
          ],"ProposalResponseStatus": 200,
          "ProposalResponsePayload": ""
        }
      ]
    }
  ]
}
```

Chapter 3

Using the Fabric Wizard to Create a Fabric Agent Skin

This chapter describes how to use the Fabric Wizard to create a Fabric agent skin.

In this chapter:

Fabric Wizard Overview

The agents (services) that interact with the chaincode are general purpose, as seen by the user of the *arg0* type parameters.

The Fabric Wizard (also referred to as the *Fabric Agent Generator* or *Fabric Skinner*) in iWay Service Manager (iSM) enables a chaincode designer to *skin* the agent with more meaningful chaincode-specific prompts and metadata. This simplifies their use and reduces casual errors that can enter a process flow (pflow) due to an incorrect configuration.

This chapter demonstrates how to apply an application skin to the Fabric posting agent appropriate to a funds transfer transaction. The chaincode will be assumed to execute the chaincode transaction Post, with the following parameters:

Actual Agent Parameter	Meaningful Name	Purpose
arg1	To Account	Account number to be credited
arg2	From Account	Source of the funds
arg3	Amount	Value
arg4	Correlation	Application identifier for this transaction. Default will be the iSM Transaction ID (TID).

Actual Agent Parameter	Meaningful Name	Purpose
arg5	Unique ID	Key to an ancillary table holding additional information.

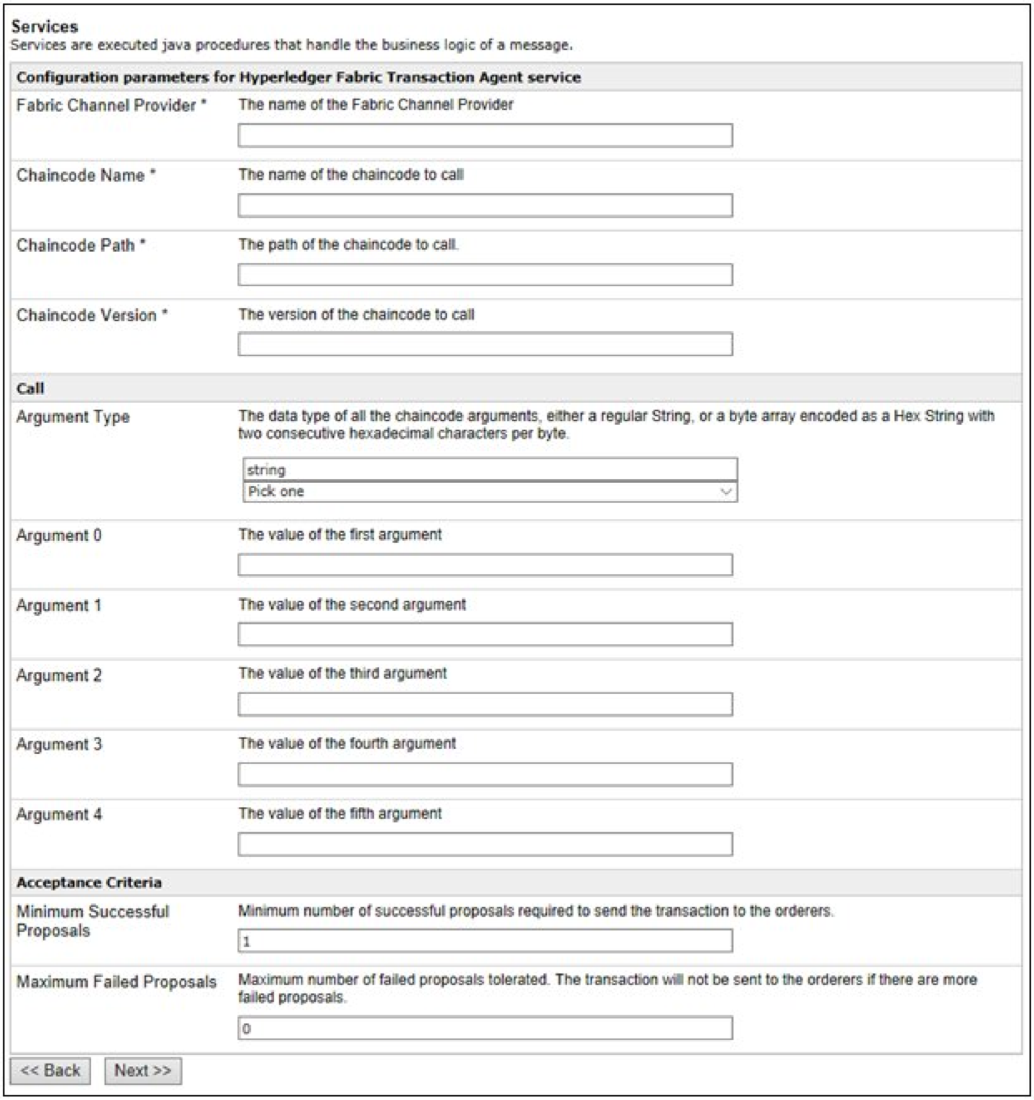
Additional parameters would be required for a true transfer. However for demonstration purposes, only five parameters are being configured.

The agent generator enables the application designer to assign a meaningful label and description to the agent, along with defaults for each configuration parameter. The *arg* parameters, limited to five in the basic agent as delivered, can be extended to the needed number for the application and each assigned an application-meaningful name.

Original Agent Configuration

A process flow could use the Fabric Agent to invoke chaincode to execute a transaction on a blockchain. For example, setting *Argument 0* to *Post*, *Argument 1* to the target account number (possibly obtained by an *_xpath()* or *_jsonpath()* iFL function) and so forth. *Argument 5*, the iSM TID, would be set to *_sreg(tid)*.

As presented to the pflow builder, the form has no values, as shown in the following image.



Services

Services are executed java procedures that handle the business logic of a message.

Configuration parameters for Hyperledger Fabric Transaction Agent service

Fabric Channel Provider *	The name of the Fabric Channel Provider	
Chaincode Name *	The name of the chaincode to call	
Chaincode Path *	The path of the chaincode to call.	
Chaincode Version *	The version of the chaincode to call	

Call

Argument Type	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte.	string Pick one
Argument 0	The value of the first argument	
Argument 1	The value of the second argument	
Argument 2	The value of the third argument	
Argument 3	The value of the fourth argument	
Argument 4	The value of the fifth argument	

Acceptance Criteria

Minimum Successful Proposals	Minimum number of successful proposals required to send the transaction to the orderers.	1
Maximum Failed Proposals	Maximum number of failed proposals tolerated. The transaction will not be sent to the orderers if there are more failed proposals.	0

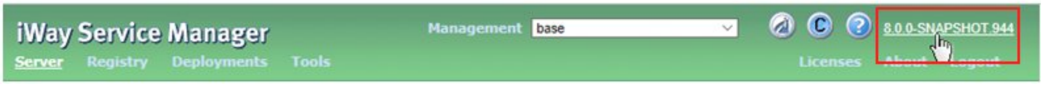
The Template (“Fabric Skinner”)

The template exists in the *blue* version of the iWay Service Manager (iSM) Administration Console, reached as a development tool. The template produces Java code that subclasses the base agents in the *iwxfabric* extension. For more information on compiling and packaging the code to execute in iSM, see the *iSM Programmer's Guide*. Essentially you will create your own extension that has a dependency on the *iwxfabric* extension.

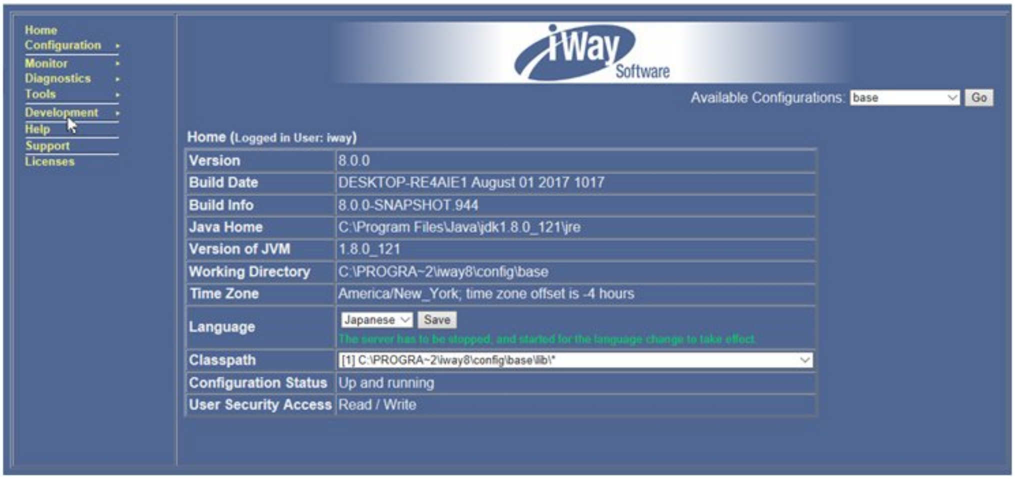
Accesing the Fabric Wizard

To access the Fabric Wizard:

1. Logon to the iSM Administration Console and click the build version number in the upper-right corner, as shown in the following image.



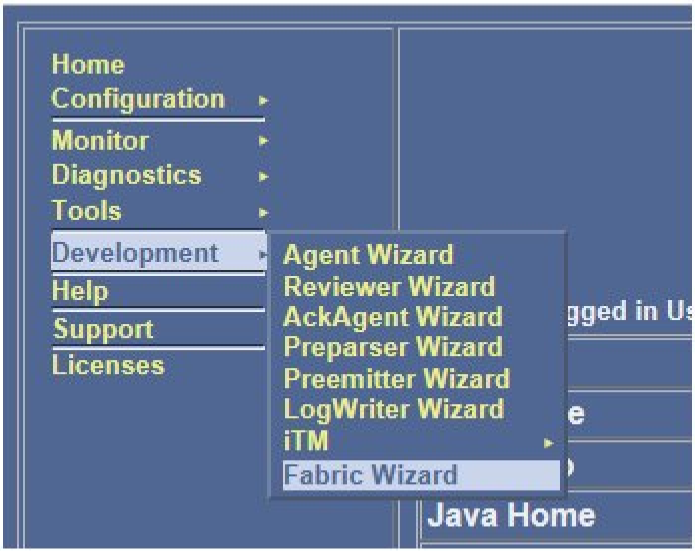
The *blue* version of the iSM Administration Console opens, as shown in the following image.



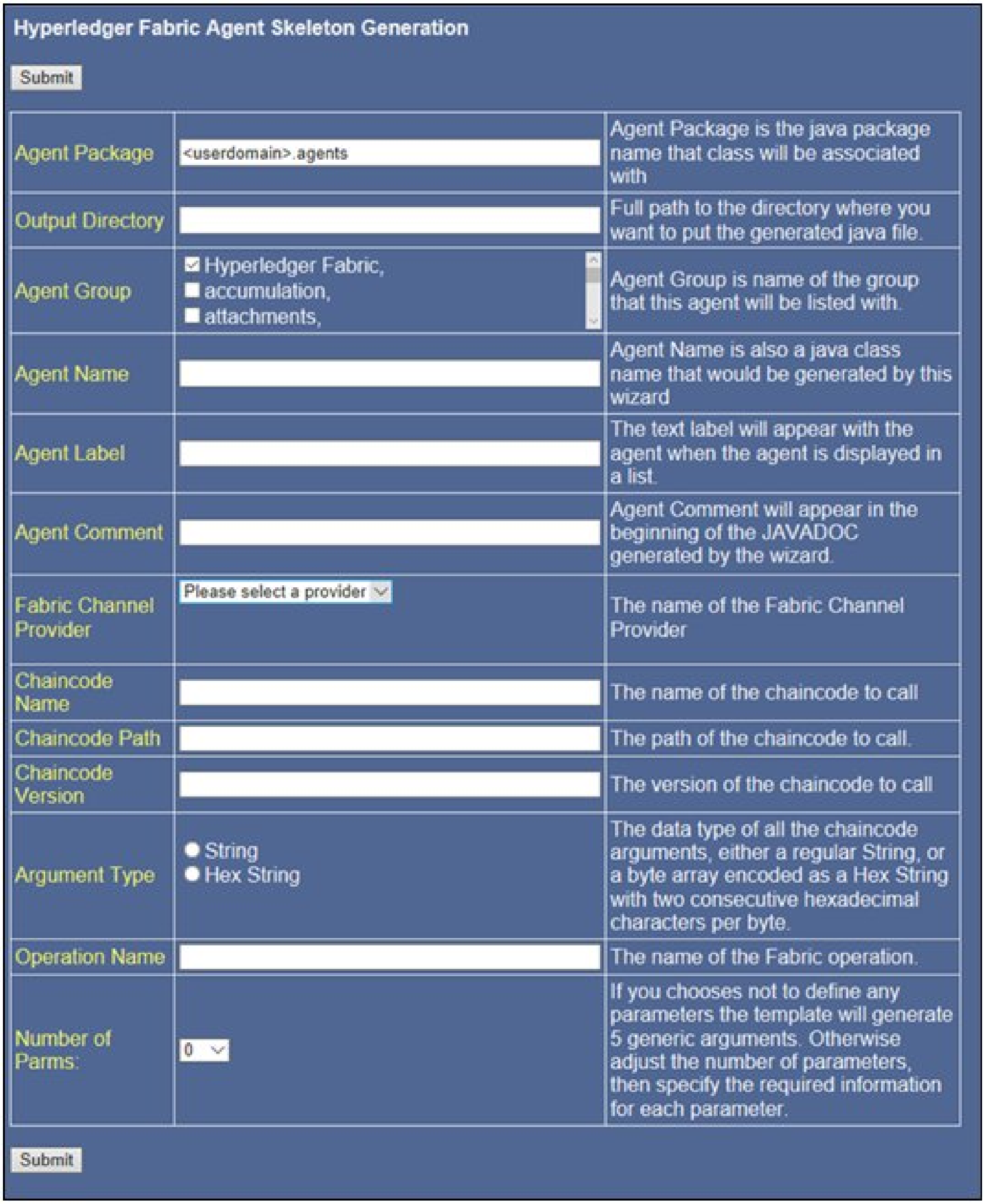
Home (Logged in User: iway)

Version	8.0.0
Build Date	DESKTOP-RE4AIE1 August 01 2017 1017
Build Info	8.0.0-SNAPSHOT.944
Java Home	C:\Program Files\Java\jdk1.8.0_121\jre
Version of JVM	1.8.0_121
Working Directory	C:\PROGRA~2\iway8\config\base
Time Zone	America/New_York; time zone offset is -4 hours
Language	Japanese Save The server has to be stopped, and started for the language change to take effect.
Classpath	[1] C:\PROGRA~2\iway8\config\base\lib\*
Configuration Status	Up and running
User Security Access	Read / Write

2. On the left pane, click *Development* and then *Fabric Wizard* from the context menu, as shown in the following image.



The Hyperledger Fabric Agent Skeleton Generation pane opens, as shown in the following image.



Hyperledger Fabric Agent Skeleton Generation

Agent Package	<userdomain>.agents	Agent Package is the java package name that class will be associated with
Output Directory		Full path to the directory where you want to put the generated java file.
Agent Group	☑ Hyperledger Fabric, □ accumulation, □ attachments,	Agent Group is name of the group that this agent will be listed with.
Agent Name		Agent Name is also a java class name that would be generated by this wizard
Agent Label		The text label will appear with the agent when the agent is displayed in a list.
Agent Comment		Agent Comment will appear in the beginning of the JAVADOC generated by the wizard.
Fabric Channel Provider	Please select a provider	The name of the Fabric Channel Provider
Chaincode Name		The name of the chaincode to call
Chaincode Path		The path of the chaincode to call.
Chaincode Version		The version of the chaincode to call
Argument Type	○ String ○ Hex String	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte.
Operation Name		The name of the Fabric operation.
Number of Parms:	0	If you chooses not to define any parameters the template will generate 5 generic arguments. Otherwise adjust the number of parameters, then specify the required information for each parameter.

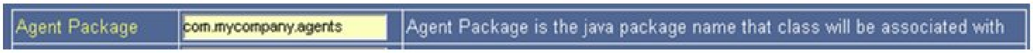
This *Fabric Skinner* form allows you to create an iSM agent that can be used in a process flow. The values for these parameters can be changed during design time. iFL functions can also be used to provide specific values for these parameters during run time.

Understanding the Basics

When you begin to skin a Fabric agent, you need to make a few decisions regarding the agent package, output directory, agent group, agent name, label, and comments. You must use the available parameters on the *Fabric Skinner* form to set these values, which are described in this section.

Agent Package

The Agent Package parameter tells the Java compiler where the agent resides.

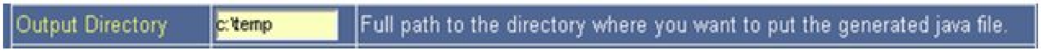


Agent Package	com.mycompany.agents	Agent Package is the java package name that class will be associated with

Similar Java classes are usually placed within the same package. The package *com.ibi.agents* is reserved for iWay Software agents and should not be used. One suggestion would be to use an abbreviation for your company name. For example, if your company name is *My Company*, then the package name could be *com.mycompany.agents*. Note that Java guidelines discourage the use of capital letters in package names.

Output Directory

The Output Directory parameter tells the Skinner where on your system the generated agent and the accompanying translation files will be stored.



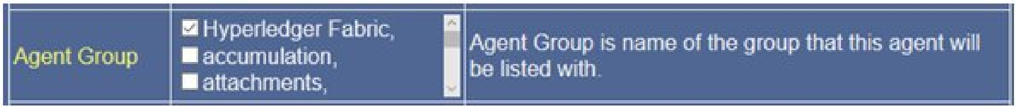
Output Directory	c:\temp	Full path to the directory where you want to put the generated java file.

You must enter a full directory path that is accessible to this Service Manager instance. Pick a directory outside of the Service Manager's directory structure. Using a directory within the Service Manager's area may cause operational problems or prevent updates from occurring.

Agent Group

The Agent Group parameter allows you to logically group this agent with similar agents. This will help you find your new agent when you create a process flow using iWay Integration Tools (iIT).

By default, the Hyperledger Fabric group is already selected for you, as shown in the following image.

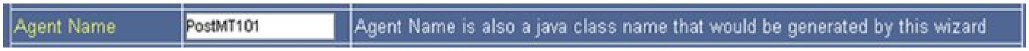


Agent Group	☑ Hyperledger Fabric, □ accumulation, □ attachments,	Agent Group is name of the group that this agent will be listed with.

Browse through the available groups by using the scroll bar and select the corresponding group that is most appropriate for your agent. If none of the listed groups are appropriate or you cannot decide, then leave the default Hyperledger Fabric group selected. You can always modify this at a later time.

Agent Name

The Agent Name parameter is the actual Java class name that will be used when the Fabric Wizard generates the Java source code.

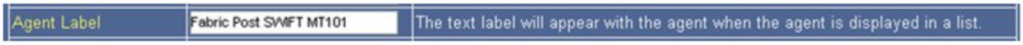


Agent Name	PostMT101	Agent Name is also a java class name that would be generated by this wizard

When determining an agent name, ensure that the name is descriptive of the class. For example, *PostMT101* when the class is used for posting a SWIFT MT101 message (Request for Transfer). Java classes typically have the first character in the name capitalized. In this example, the P in post is capitalized. As a best practice, the first letter of each word that would follow a space in the name of the class is capitalized. In this example, *post MT101* becomes *PostMT101*.

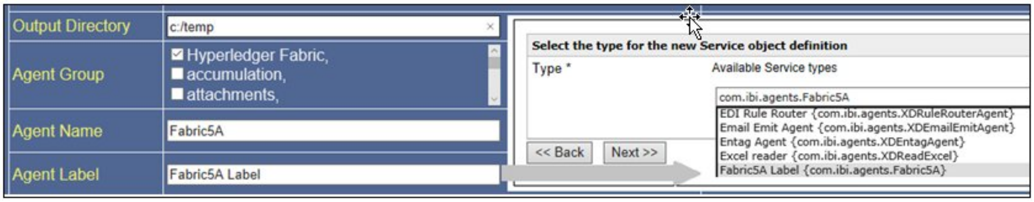
Agent Label

The Agent Label parameter contains a short descriptive sentence.



Agent Label	Fabric Post SWIFT MT101	The text label will appear with the agent when the agent is displayed in a list.

This sentence should describe precisely what the Fabric agent does. The Agent Label is what is displayed in the selection list when you look for the agent to configure, as shown in the following image.



Output Directory	c:/temp
Agent Group	☑ Hyperledger Fabric, □ accumulation, □ attachments,
Agent Name	Fabric5A
Agent Label	Fabric5A Label

Select the type for the new Service object definition

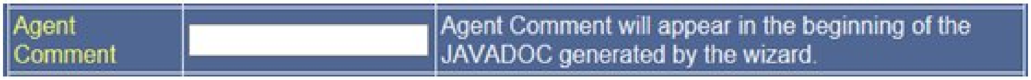
Type * Available Service types

com.ibi.agents.Fabric5A

EDI Rule Router {com.ibi.agents.XDRuleRouterAgent}
Email Emit Agent {com.ibi.agents.XDEmailEmitAgent}
Entag Agent {com.ibi.agents.XDEntagAgent}
Excel reader {com.ibi.agents.XDReadExcel}
Fabric5A Label {com.ibi.agents.Fabric5A}

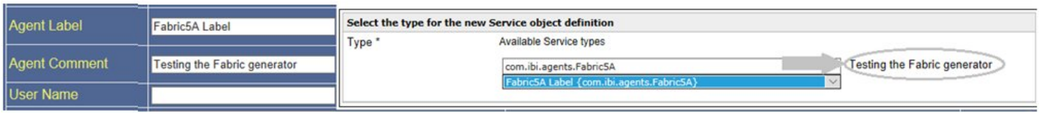
Agent Comment

The Agent Comment parameter is used to enter more descriptive text about the agent's function.



Agent Comment		Agent Comment will appear in the beginning of the JAVADOC generated by the wizard.

This comment is used within the Java's JAVADOC and displayed to the user when the agent is selected for configuration, as shown in the following image.



Agent Label	FabricSA Label
Agent Comment	Testing the Fabric generator
User Name	

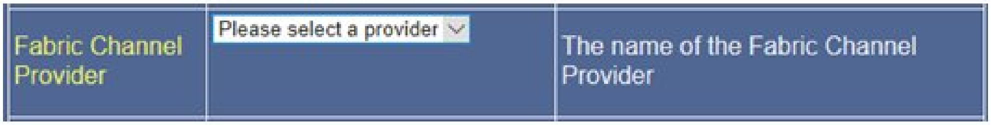
Select the type for the new Service object definition

Type * — Available Service types: com.ibi.agents.FabricSA; FabricSA Label {com.ibi.agents.FabricSA}

Testing the Fabric generator

Fabric Channel Provider

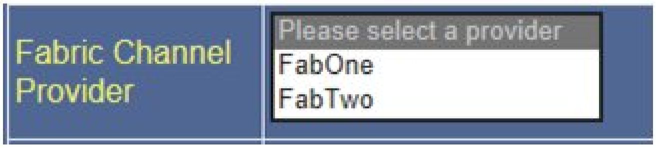
The Fabric Channel Provider parameter allows you to select an available Fabric Provider from a drop-down list, as shown in the following image.



Fabric Channel Provider	Please select a provider	The name of the Fabric Channel Provider

Each configured provider will contain the parameters that are required to execute the Fabric transaction (for example, User Name, MSPID, Enrollment Certificate, and so on). Since these parameters are shared between all Fabric chaincode calls and should not change, a provider is used where a centralized repository of Fabric values is available. This topic does not go into detail describing each of those parameters. You can set default values for your Fabric Provider that are appropriate to your application design.

Click the drop-down list to display a list of available Fabric Providers, as shown in the following image.



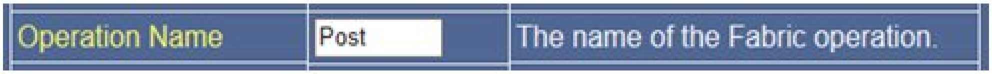
Fabric Channel Provider

- Please select a provider
- FabOne
- FabTwo

Select a Fabric Provider from this list. Note that your list will vary. The names and the number of entries are based on the names and number of Fabric Providers that you have configured.

Operation Name

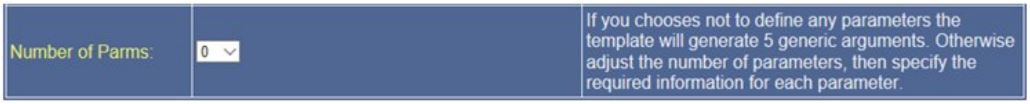
The Operation Name parameter, by convention normally the first argument (Argument 0) to follow Fabric chaincode specific parameters is the Fabric's chaincode operation. In this example, we are going to enter Post, as shown in the following image.



Operation Name	Post	The name of the Fabric operation.

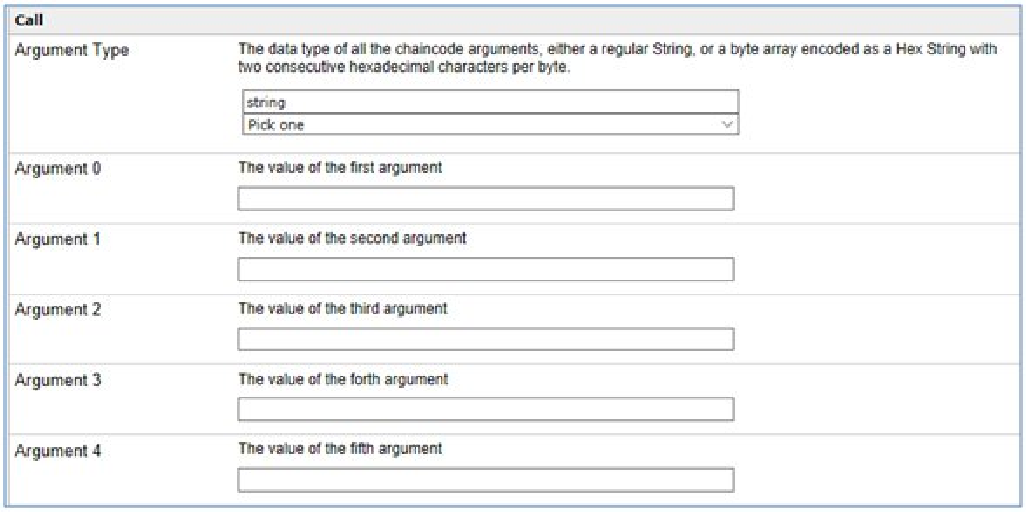
Number of Parms

The Number of Parms parameter allows you to customize the parameters that will be displayed when your Skinned Fabric Agent is presented for configuration.



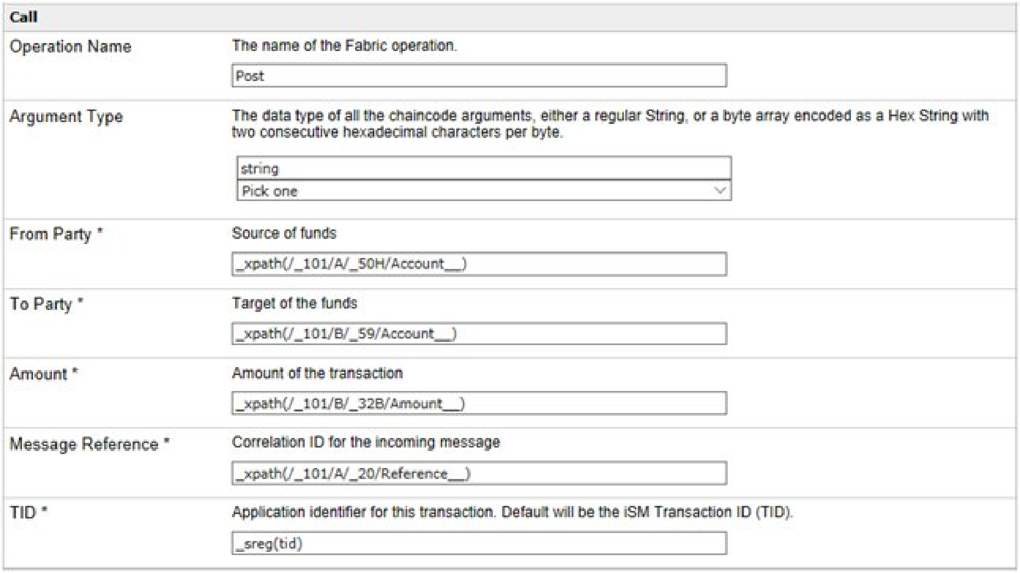
Number of Parms:	0	If you chooses not to define any parameters the template will generate 5 generic arguments. Otherwise adjust the number of parameters, then specify the required information for each parameter.

By default the Fabric Agent displays 5 generic parameters labeled Argument 0 through Argument 4, as shown in the following image.



Call	
Argument Type	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte. string Pick one
Argument 0	The value of the first argument
Argument 1	The value of the second argument
Argument 2	The value of the third argument
Argument 3	The value of the forth argument
Argument 4	The value of the fifth argument

Each of the above argument (Argument 0 through Argument 4) values would have to be configured at design time. But when you set the Number of Parms to a value other than 0, and define those parameters, the Fabric call parameters become much more relevant and easier for a user to understand and configure, as shown in the following image.

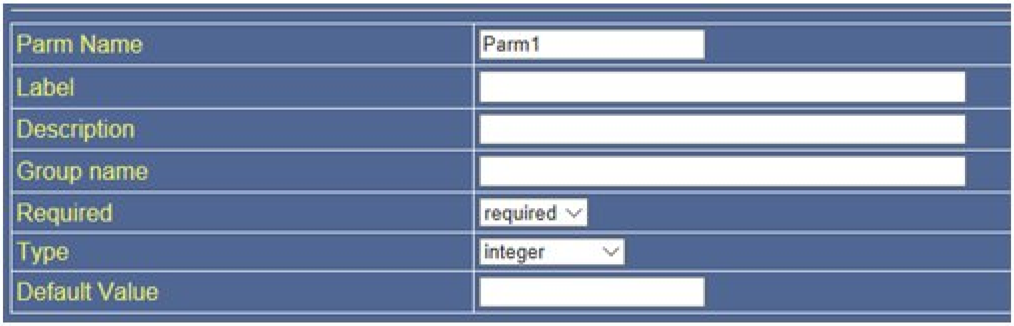


Call

Operation Name	The name of the Fabric operation. Post
Argument Type	The data type of all the chaincode arguments, either a regular String, or a byte array encoded as a Hex String with two consecutive hexadecimal characters per byte. string Pick one
From Party *	Source of funds _xpath(/_101/A/_50H/Account__)
To Party *	Target of the funds _xpath(/_101/B/_59/Account__)
Amount *	Amount of the transaction _xpath(/_101/B/_32B/Amount__)
Message Reference *	Correlation ID for the incoming message _xpath(/_101/A/_20/Reference__)
TID *	Application identifier for this transaction. Default will be the iSM Transaction ID (TID). _sreg(tid)

Defining Additional Parameters

When you select a number of parameters from the list (up to 10) the skinner displays the parameter configuration block for each parameter, as shown in the following image.



Parm Name	Parm1
Label	
Description	
Group name	
Required	required
Type	integer
Default Value	

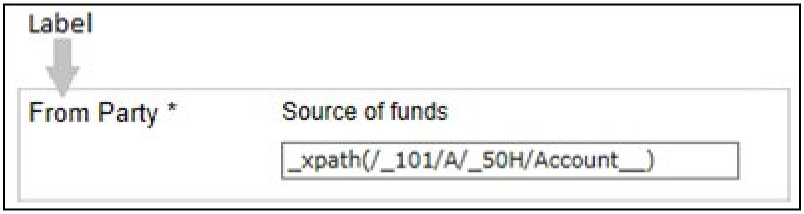
This block shows the default for the first parameter, and is the same for each parameter configuration block that follows.

Parm (Parameter) Name

This is the name that will be used by the Java class to identify the parameter. Based on usage guidelines, the first character in the name is **not** capitalized. For example, a good parameter name that can be used to identify the source of funds in a financial transaction could be *fromAccount*. Notice that the first letter in *Account* was capitalized, which improves readability. Ensure that parameter names are descriptive, but not too long.

Label

The Label parameter is used to identify the parameter to the person who is configuring this agent.



From Party *

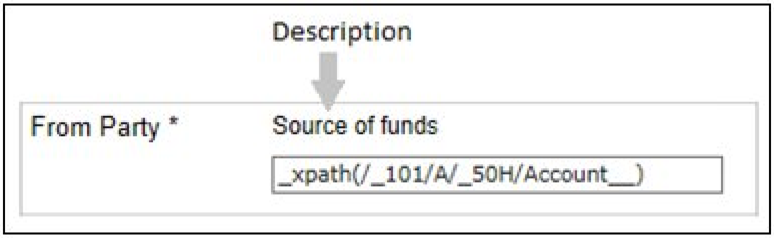
Source of funds

_xpath(/_101/A/_50H/Account__)

The value (description) entered here should be concise (one to three words). The Description parameter that follows allows you to provide more descriptive content as required.

Description

The Description parameter allows you to enter a detailed description of the field in question.



From Party *

Source of funds

_xpath(/_101/A/_50H/Account__)

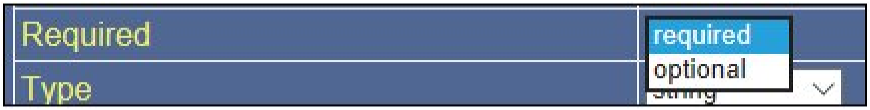
The description can also include suggestions on possible entries in this field or any other information that would help the user make a valid entry into this field.

Group Name

This parameter is not used at this time and can be left blank.

Required

Clicking on the required field displays a drop-down listing of two choices; required or optional.

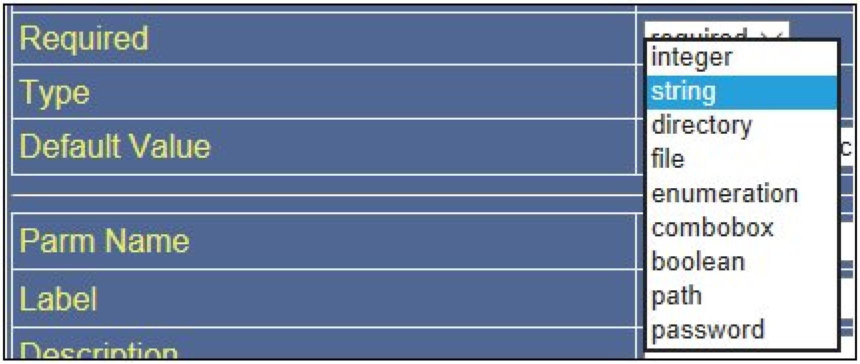


This drop-down list allows you to select between the two options.

- **required.** Indicates that the person configuring the parameter must make a valid entry in the field. If a field is required a visual queue of "*" follows the field label on the screen.
- **optional.** An entry in this field is not necessary and may be left blank by the person configuring the agent.

Type

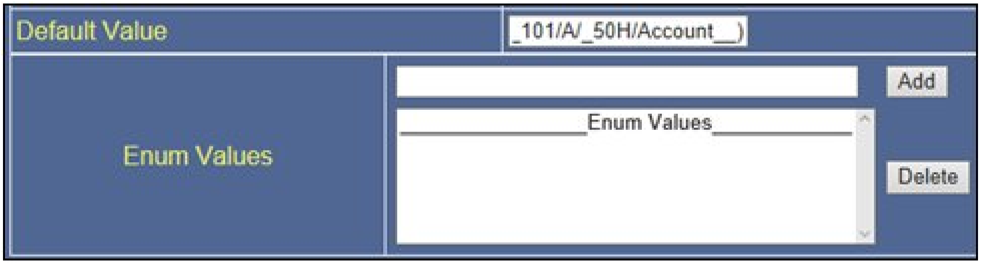
The type parameter tells the Java class what type of data to expect in this field.



Clicking on this field displays a drop-down list. The Type selected is used by the Service Manager to validate the parameter's data field at design time.

- **integer.** The integer type will contain a number that can be written without a fractional component. For example, 21, 4, 0, and ?2048 are integers, while 9.75, 4.8 are not.
- **string.** The string type will contain any combination of letters numbers and white space (except tabs).
- **directory.** The directory type will contain a fully qualified path to a directory accessible to iSM.
- **file.** The file type will contain a fully qualified path to a file accessible to the Service Manager.

- **enumeration.** The enumeration type allows you to enter a fixed list of values that the person configuring the agent can choose from during design time. To build the list, enter the value in the field to the left of the Add button. Once a value is specified, click *Add* to add the entry to the list.

  The entry will then be moved to the list that is forming under the field. To remove an entry from the list. Highlight the entry in the list and click the Delete button.

- **combobox.** The combobox type is similar to the enumeration. The combobox allows you to enter a selection of values that the person configuring the agent can choose from but also allow the person to enter a value into the field that is not on the list of values.

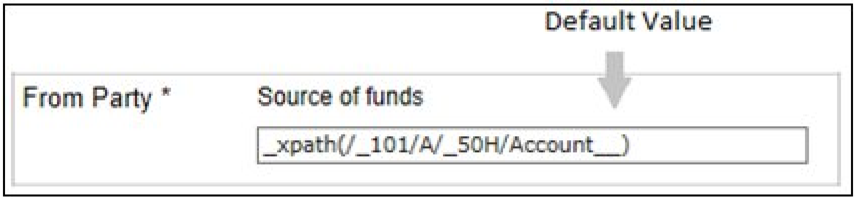
- **boolean.** The boolean type will display a selection list with only two items in the list true or false.

- **path.** Similar to the directory or file types field will contain a fully qualified path to a directory or file that is accessible to iSM.

- **password.** Similar to the string type the field will contain any combination of letters numbers and white space (except tabs). The difference is that the value entered will not be shown, in its place each character is replaced by the "*" character.

Default Value

The Default Value is what the Service manager will display to the user at design time.



When you have completed configuring all of the parameters click the Submit button. The generator produces two files:

- Text properties
- Agent executable code:

```
#Text Strings associated with PostMT101.java
#Mon Aug 07 11:20:21 EDT 2017
PostMT101.desc=Posts a SWFT DB/CR from MT101 Message
PostMT101.amtParm.desc=Amount to transfer in USD
…
```

The new agent as generated should be compiled and loaded into an iSM extension. For more information, see the *iSM Programmer's Guide*. This agent can now be used in a process flow.

The values entered into the fields will be the defaults for the configuration. They can be overridden at the time the agent is configured.

Chapter 4

Blockchain Example

This chapter provides an example that demonstrates how an application can be used to accept a SWIFT MT101 message (Request for Transfer) and post the transfer to a blockchain (synchronously and asynchronously).

In this chapter:

Understanding the Process Flow

The application in this example contains a channel that uses an arbitrary listener to read the MT101 message found in *Original SWIFT MT101 Message (Request for Transfer)* on page 66. The e-Business Information Exchange (Ebix) file for SWIFT converts this message to XML as shown in *Transformed SWIFT MT101 Message (XML Format)* on page 67, before passing the message to a process flow. For simplicity and demonstration purposes, most of the MT101 message, after it has been transformed into XML, is omitted.

The process flow performs the following steps:

1. Updates a local database with the ancillary data required for the actual financial records. These additional data fields are not carried on the blockchain. No commit is done.

2. Sends the transaction fields to the blockchain.

3. If successful, then the process flow completes and a commit is automatically made to the database.

4. If the transaction is rejected by a timeout, then the database is rolled back and the message is sent as rejected with an iSM retry. This retry causes the message to be represented to the process flow at a later time (as configured). For more information on retry processing, see the *iWay Service Manager User's Guide*.

5. If the transaction is rejected as an error, then the database is rolled back and an email is sent to a responsible party.

The following table lists the parameter values that are used by this example for the original (raw) Hyperledger Fabric Transaction service.

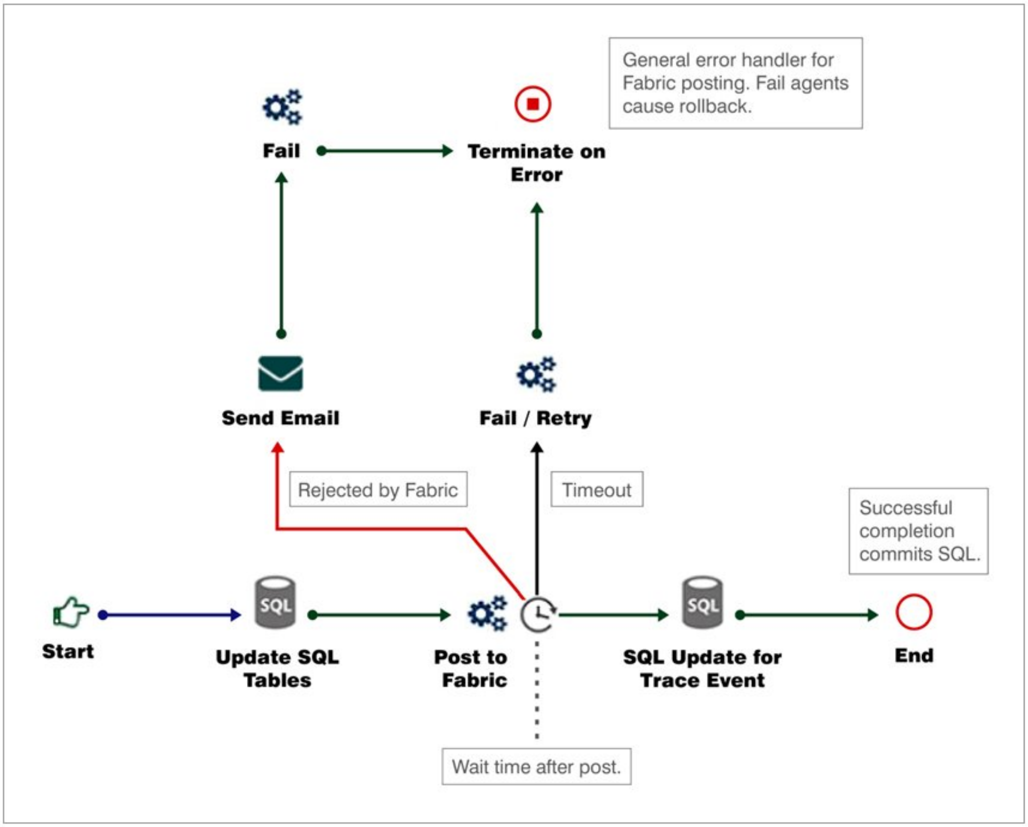
Parameter	Value
arg0	`Post`
arg1	`_xpath(/_101/A/_50H/Account__)`
arg2	`_xpath(/_101/B/_59/Account__)`
arg3	`_xpath(/_101/B/_32B/Amount__)`
arg4	`_xpath(/_101/A/_20/Reference__)`
arg5	`_sreg(tid)`

The following table lists the corresponding parameter values that are used by this example for the *skinned* version of the Hyperledger Fabric Transaction service. For more information, see *Using the Fabric Wizard to Create a Fabric Agent Skin* on page 45.

Parameter	Value
Function	`Post`
From Party	`_xpath(/_101/A/_50H/Account__)`
To Party	`_xpath(/_101/B/_59/Account__)`
Amount	`_xpath(/_101/B/_32B/Amount__)`
Message Reference	`_xpath(/_101/A/_20/Reference__)`
Correlation	`_sreg(tid)`

Synchronous Posting Example

The following diagram illustrates the general process flow used for synchronous posting.

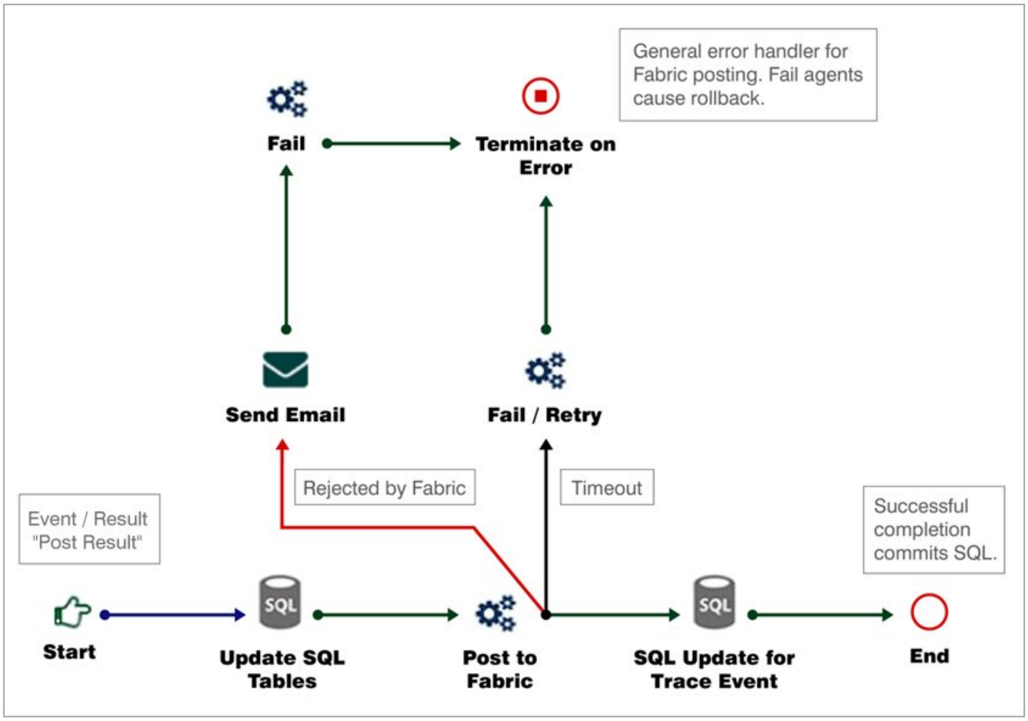


Complete posting to the Hyperledger Fabric is handled by this single process flow.

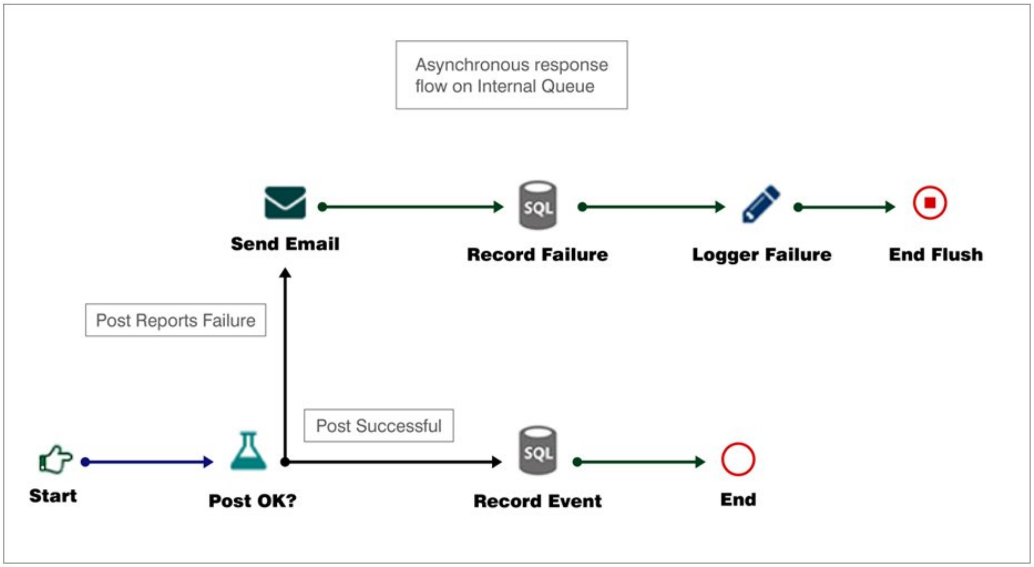
Asynchronous Posting Example

Asynchronous posting divides the work between two process flows. The first process flow performs the actual post and the second process flow checks the post result. The wait time for posting to the Hyperledger Fabric is reduced by not waiting for a response from the Orderer. The following pair of diagrams illustrate the general approach used for asynchronous posting.

The first process flow posts and checks for problems (only from the peers), as shown in the following diagram.


General error handler for Fabric posting. Fail agents cause rollback.
Fail
Terminate on Error
Send Email
Fail / Retry
Rejected by Fabric
Timeout
Event / Result "Post Result"
Successful completion commits SQL.
SQL
SQL
Start
Update SQL Tables
Post to Fabric
SQL Update for Trace Event
End

The actual posting result from the Orderer is handled in the response process flow, as shown in the following diagram.



Application Design Consideration

iWay Correlation Manager (not shown in this example) can be used for tracking expected events in longer running process flows (for example, the delay between the post and the response). Consider a scenario where the posting side might open the correlation using the transaction ID for correlation, and the response side might close the correlation. For more information on iWay Correlation Manager, see the *iWay Business Activity Monitor User's Guide*.

Sample SWIFT MT101 Documents

For reference purposes, this section provides the original and XML transformed version of the SWIFT MT101 document.

Standard iSM services are used to generate an XML representation of the SWIFT MT101 message, which is suitable for use in a process flow. For this example, portions of the transformed SWIFT MT101 message in XML format that are not required have been removed.

For more information on integrating with SWIFT using iSM, see the *iWay Integration Solution for SWIFT 2017 User's Guide*.

Original SWIFT MT101 Message (Request for Transfer)

```
{1:F01FNBCUS44A1230000000000}{2:I101BOFAUS6SX123N2}{4:
:20:091117-DSNY0001
:28D:1/1
:50H:/12345-67891
WALT DISNEY COMPANY
MOUSE STREET 1
LOS ANGELES CA
:30:091117
:21:WDC091117RPCUS
:32B:USD377250,
:50L:WALT DISNEY COMPANY LOS ANGELES CA
:57A:WFBIUS6S
:59:/26351-38947
RIVERS PAPER COMPANY
37498 STONE ROAD
SAN RAMON CA
:71A:OUR
-}
```

Transformed SWIFT MT101 Message (XML Format)

```
<?xml version="1.0" encoding="ISO-8859-1" ?><Output><SWIFTMT101>
    <Basic_Header>
     …
    </Basic_Header>
    <Application_Header>
        <I>…
            <Message_Type__>101</Message_Type__>
        </I>
    </Application_Header>
    <_101>
        <A>
            <_20>
                <Reference__>091117-DSNY0001</Reference__>
            </_20>
…
            <_50H>
                <Account__>12345-67891</Account__>
                <Name_Address__>…</Name_Address__>
            </_50H>
            <_30>
                <Date__>091117</Date__>
            </_30>
        </A>
        <B>
            <_21>
                <Transaction_Reference__>WDC091117RPCUS</
Transaction_Reference__>
            </_21>
            <_32B>
                <Currency__>USD</Currency__>
                <Amount__>377250,</Amount__>
            </_32B>
            <_50L>
                <Party_Identifier__>…</Party_Identifier__>
            </_50L>
            <_57a>
                <Identifier_Code__>WFBIUS6S</Identifier_Code__>
            </_57a>
            <_59>
                <Account__>26351-38947</Account__>
                <Name_Address…</Name_Address__>
            </_59>
            <_71A>
                <Code__>OUR</Code__>



</_71A>
        </B>
    </_101>
</SWIFTMT101></Output>
```

Glossary

This appendix provides a reference for common terms that are used in blockchain discussions with iWay Service Manager (iSM).

Block:

A block is a collection of data that contains zero or more transactions, the hash of the previous block (*parent*), and other data. The genesis block is the first block. The total set of blocks is called the blockchain and contains the entire transaction history of a network.

Note that some blockchain-based cryptocurrencies such as Bitcoin use the word ledger instead of blockchain. The two are roughly equivalent.

Blockchain:

An ever-extending series of data blocks that grows as new transactions are confirmed as part of a new block.

Chaincode:

Chaincode is a program that implements a prescribed interface. It initializes and manages ledger state through transactions submitted by applications.

Decentralization:

The elimination of a single *owner* of validity of the data (and the data itself) to a joint ownership by the network peers.

Hash:

A cryptographic function which takes an input (or 'message') and returns a fixed-size value, which is called the hash. A hash function (or hash algorithm) is a process by which a document (for example, a piece of data or file) is processed into a small piece of data (usually 32-bytes) which looks completely random, and from which no meaningful data can be recovered about the document, but which has the important property that the result of hashing one particular document is always the same.

Additionally, it is crucially important that it is computationally infeasible to find two documents that have the same hash. Generally, changing even one letter in a document will completely randomize the hash; for example, the SHA256 hash of *iWay Software* (lower-case “i”) is:

```
561ce1ea04f6297138c0416c31cb9a31a851b46cbd9253a8efa702c585ef50cf
```

The SHA256 hash of *IWay Software* (upper-case “I”) is:

```
d3336e44c81730e05021588fd424651068daebe0ec385fe730edf12ae5cfcbde
```

Hashes are usually used as a way of creating a globally agreed-upon identifier for a particular document that cannot be forged.

A common term for hash is checksum.

Peer:

Other computers on the network also running in Fabric with an exact copy of the blockchain that you have.

Signing:

Producing a piece of data from the data to be signed using your private key, to prove that the data originates from you. Usually a hash of the data is signed rather than the whole message.

Public, Private, and Consortium Blockchains:

Most Ethereum projects today rely on Ethereum as a public blockchain, which grants access to a larger audience of users, network nodes, currency, and markets. However, there are often reasons to prefer a private blockchain or consortium blockchain (among a group of trusted participants). For example, a number of companies in verticals, like banking, are looking to Ethereum as a platform for their own private blockchains.

Below is an excerpt from a blog post On Public and Private Blockchains that explains the difference between the three types of blockchains based on permissioning:

- ❑ **Public Blockchains.** A public blockchain is a blockchain that anyone in the world can read, anyone in the world can send transactions to and expect to see them included if they are valid, and anyone in the world can participate in the consensus process - the process for determining what blocks get added to the chain and what the current state is. As a substitute for centralized or quasi-centralized trust, public blockchains are secured by cryptoeconomics - the combination of economic incentives and cryptographic verification using mechanisms such as proof of work or proof of stake, following a general principle that the degree to which someone can have an influence in the consensus process is proportional to the quantity of economic resources that they can bring to bear. These blockchains are generally considered to be *fully decentralized*.
- ❑ **Consortium Blockchains.** A consortium blockchain is a blockchain where the consensus process is controlled by a pre-selected set of nodes; for example, one might imagine a consortium of 15 financial institutions, each of which operates a node and of which 10 must sign every block in order for the block to be valid. The right to read the blockchain may be public, or restricted to the participants, and there are also hybrid routes such as the root hashes of the blocks being public together with an API that allows members of the public to make a limited number of queries and get back cryptographic proofs of some parts of the blockchain state. These blockchains may be considered *partially decentralized*.

❑ **Private Blockchains.** A fully private blockchain is a blockchain where write permissions are kept centralized to one organization. Read permissions may be public or restricted to an arbitrary extent. Likely applications include database management, auditing, etc internal to a single company, and so public readability may not be necessary in many cases at all, though in other cases public auditability is desired.

Legal and Third-Party Notices

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.