

Block Chain Adapter API Configuration
Oracle FLEXCUBE Investor Servicing

Release 14.0.0.2.5

[January] [2019]



Table of Contents

1.	BLOCKCHAIN ADAPTER CONFIGURATION.....	1
1.1	INTRODUCTION	1
1.2	ADAPTER CONFIGURATION XML [TECHNICAL DETAILS]	1
1.2.1	<i>Configuration in CamelContext.xml</i>	<i>2</i>
1.2.2	<i>P6: MFHYSENDER-IN-TABLE MFHYSENDER-OUT-HTTP.....</i>	<i>3</i>
1.2.3	<i>P5: MFHYRECEIVER-IN-HTTP MFHYRECEIVER-OUT-TABLE</i>	<i>5</i>
1.2.4	<i>P3: MFHYSENDER-IN-HTTP MFHYSENDER-OUT-TABLE.....</i>	<i>6</i>
1.2.5	<i>P4: MFHYRECEIVER-IN-TABLE MFHYRECEIVER-OUT-HTTP</i>	<i>7</i>
1.2.6	<i>P2: MFSENDER-IN-TABLE-UPDATE MFSENDER-OUT-HTTP-UPDATE.....</i>	<i>9</i>
1.2.7	<i>P1: MFRECEIVER-IN-HTTP-UPDATE MFRECEIVER-OUT-TABLE-UPDATE</i>	<i>11</i>
1.3	APIs	12
1.3.1	<i>Posting new transaction.....</i>	<i>12</i>
1.3.2	<i>Update transaction</i>	<i>16</i>

1. Blockchain Adapter Configuration

1.1 Introduction

This chapter explains the steps for updating configuration XML files available in Blockchain Adapter WAR.

1.2 Adapter Configuration XML [Technical Details]

Block Chain Java Adapter for Mutual Fund Transactions (FCIS) Use Case. There are 6 Apache Camel routes as follows:

1. P6: MFHYSENDER-IN-TABLE MFHYSENDER-OUT-HTTP
2. P5: MFHYRECEIVER-IN-HTTP MFHYRECEIVER-OUT-TABLE
3. P3: MFHYSENDER-IN-HTTP MFHYSENDER-OUT-TABLE
4. P4: MFHYRECEIVER-IN-TABLE MFHYRECEIVER-OUT-HTTP
5. P2: MFSENDER-IN-TABLE-UPDATE MFSENDER-OUT-HTTP-UPDATE
6. P1: MFRECEIVER-IN-HTTP-UPDATE MFRECEIVER-OUT-TABLE-UPDATE

1.2.1 Configuration in CamelContext.xml

CamelContext.xml

```
<bean id="sqlComponent"
class="org.apache.camel.component.sql.SqlComponent">
    <property name="dataSource" ref="dataSource" />
</bean>

<bean id="transformerMF"
class="oracle.fsgbu.flexcube.bc.fcis.mf.transformer.MFTransformer" />

<bean id="parserMF"
class="oracle.fsgbu.flexcube.bc.fcis.mf.parser.MFParser" />

<bean id="RestletComponent" class="org.restlet.Component" />

<bean id="RestletComponentService"
class="org.apache.camel.component.restlet.RestletComponent">
    <constructor-arg index="0">
        <ref bean="RestletComponent" />
    </constructor-arg>
</bean>
```

sqlComponent

This bean is used to connect to a database whose credentials are defined in application.properties.

transformMF

MFTransformer class is linked to CamelContext in order to use its methods in the routes.

parserMF

MFParser class is linked to CamelContext in order to use its methods in the routes.

RestletComponent

This bean is used to define REST Endpoints being used in the routes.

RestletComponentService

This bean is used to define REST Endpoints being used in the routes.

1.2.2 P6: MFHYSENDER-IN-TABLE MFHYSENDER-OUT-HTTP

CamelContext.xml

```
<!-- P6: MFHYSENDER-IN-TABLE MFHYSENDER-OUT-HTTP -->
<route id="P6:MFSender_DBtoREST">
    <from uri="sqlComponent:{{sql.newTransactionProcessing}}" />
    <transform>
        <method ref="parserMF" method="parse" />
    </transform>
    <log message="\${body}" />
    <to uri="freemarker:templateMF3.ftl" />
    <log message="\${body}" />
    <setHeader headerName="Content-Type">
        <constant>application/json</constant>
    </setHeader>
    <to uri="http://localhost:8080/transaction" />
    <convertBodyTo type="String" />
    <choice>
        <when>
            <simple>\${body} contains 'ok'</simple>
            <log message="\${body}" />
            <transform>
                <method ref="parserMF" method="getRowID" />
            </transform>
            <log message="\${body}" />
            <to
uri="sqlComponent:{{sql.deleteRecord_Sender_DBtoREST_Update}}"
/>
                <to uri="sqlComponent:{{sql.commitSql}}" />
            </when>
        </choice>
    </route>
```

A new record added through Oracle FLEXCUBE Frontend is automatically picked up for processing from the sender database. After being processed by the java adapter, the data is prepared as a freemarker template and then sent to Hyperledger. If a successful response is received then the record is deleted from the ofba table.

MF_SQLConfig.properties

```
sql.newTransactionProcessing = SELECT ROWID AS  
ROW_ID, TRANSACTIONNUMBER, SOURCEENTITY, NVL (DESTINATIONENTITY, ' ')  
DESTINATIONENTITY, NVL (STATUS, ' ') STATUS,  
TO_CHAR (TRANSACTIONDATE, 'DD-MON-YY') AS  
TransactionDate, NVL (FUNDID, ' ') FUNDID, NVL (UNITHOLDERIDTYPE, '  
' ) UNITHOLDERIDTYPE, NVL (UNITHOLDERIDNUMBER, ' ' )  
UNITHOLDERIDNUMBER, NVL (TOFUNDID, ' ' )  
TOFUNDID, NVL (TOUNITHOLDERIDTYPE, ' ' )  
TOUNITHOLDERIDTYPE, NVL (TOUNITHOLDERIDNUMBER, ' ' )  
TOUNITHOLDERIDNUMBER, NVL (TRANSACTIONCCY, ' ' )  
TRANSACTIONCCY, NVL (TRANSACTIONMODE, ' ' )  
TRANSACTIONMODE, NVL (TRANSACTIONVALUE, 0)  
TRANSACTIONVALUE, NVL (GROSSORNET, ' ' )  
GROSSORNET, NVL (UNITSALLOTTED, 0) UnitsAlloted, NVL (NAV, 0)  
NAV, NVL (MESSAGEDIRECTION, ' ' ) MESSAGEDIRECTION FROM  
DLTB_TRANSACTION_DETAIL_OFBA WHERE TYPE_OF_TRIGGER ='INSERT' AND  
ROWNUM <=1 AND MessageDirection='I' ORDER BY SERIAL_NO ASC  
  
sql.deleteRecord_Sender_DBToREST_Update=delete from  
DLTB_TRANSACTION_DETAIL_OFBA where ROWID=CHARTOROWID(:#rowid)
```

Using SQL query, newly inserted data in the sender database is retrieved and sent for processing to Parser where the row id is extracted and set into a HashMap which is used later in the route. The data from the database is used to prepare a Freemarker template. The output of the template is a JSON request which is sent to the Hyperledger network. Subsequently, the response is checked for error, and on receiving successful response with OK Status, the picked record is removed from the table 'DLTB_TRANSACTION_DETAIL_OFBA' so that the same record is not repeatedly picked up by Camel.

1.2.3 **P5: MFHYRECEIVER-IN-HTTP MFHYRECEIVER-OUT-TABLE**

CamelContext.xml

```
<!-- P5: MFHYRECEIVER-IN-HTTP MFHYRECEIVER-OUT-TABLE -->
<route id="P5:MFRcvr_RESTtoDB">
    <from uri="direct:MFTTransact" />
    <log message="P5 is called" />
    <log message="${body}" />
    <to uri="bean:transformerMF?method=transform" />
    <log message="${body}" />
    <to uri="sqlComponent:{{sql.insertQueryMF}}" />
    <to uri="sqlComponent:{{sql.commitSql}}" />
    <setBody><simple>{"result":"ok"}</simple></setBody>
    <transform>
        <method ref="transformerMF" method="stringToMap"
    />
    </transform>
</route>
```

Data posted to REST by Hyperledger is picked up by Camel route and sent to the Transformer. After processing, the record is inserted into 'DLTB_TRANSACTION_DETAIL' on the receiver side.

MF_SQLConfig.properties

```
sql.insertQueryMF = INSERT INTO DLTB_TRANSACTION_DETAIL
(SMART_CONTRACT_ID,TransactionNumber,SourceEntity,DESTINATIONENTITY,STATUS,TransactionDate,FundId,UnitHolderIdType,UnitHolderIdNumber,ToFundId,ToUnitHolderIdType,ToUnitHolderIdNumber,TransactionCCY,TransactionMode,TransactionValue,GrossorNet,MessageDirection,UnitsAlloted,NAV)
VALUES(:#SmartContractID, :#TransactionNumber, :#SourceEntity, trim(:#DestinationEntity), trim(:#Status), TO_DATE(:#TransactionDate, 'DD-MON-YY'), trim(:#FundId), trim(:#UnitHolderIdType), trim(:#UnitHolderIdNumber), trim(:#ToFundId), trim(:#ToUnitHolderIdType), trim(:#ToUnitHolderIdNumber), trim(:#TransactionCCY), trim(:#TransactionMode), :#TransactionValue, trim(:#GrossorNet), 'O', :#UnitsAlloted, :#NAV)
```

On the receiver side, data (payload-MFTransact) comes from the Hyperledger (REST endpoint), and is sent to the direct component of Apache Camel. From there, data is sent to transform() method of the MFTransformer.java class where the data along with the private data is extracted and converted to a HashMap. This data is inserted in the receiver side 'DLTB_TRANSACTION_DETAIL' table using the SQL query and the transaction is committed. On successful insertion of the record in database, a JSON string which is first converted to a Map is set as a response.

1.2.4 P3: MFHYSENDER-IN-HTTP MFHYSENDER-OUT-TABLE

CamelContext.xml

```
<!-- P3: MFHYSENDER-IN-HTTP MFHYSENDER-OUT-TABLE -->
<route id="P3:MFSENDER_RESTtoDB">
    <from uri="direct:MFNotifySender" />
    <transform>
        <method ref="transformerMF" method="transform" />
    </transform>
    <log message="P3 is called" />
    <log message="{body}" />
    <to
uri="sqlComponent:{{sql.updateQuery_Sender_RESTtoDB_Notification}}"/>
    <to uri="sqlComponent:{{sql.commitSql}}"/>
    <setBody><simple>{"result":"ok"}</simple></setBody>
    <transform>
        <method ref="transformerMF" method="stringToMap" />
    </transform>
</route>
```


Data posted to REST by Hyperledger is picked up for processing and sent to the Transformer. The returned HashMap output is used to update the record into the sender side database in the table 'DLTB_TRANSACTION_DETAIL' and the transaction is committed.

MF_SQLConfig.properties

```
sql.updateQuery_Sender_RESTToDB_Notification=UPDATE
DLTB_TRANSACTION_DETAIL SET
SMART_CONTRACT_ID=:#SmartContractID,STATUS=trim(:#Status),Transact
ionDate=TO_DATE(:#TransactionDate,'DD-MON-
YY'),FundId=trim(:#FundId),UnitHolderIdType=trim(:#UnitHolderIdTyp
e),UnitHolderIdNumber=trim(:#UnitHolderIdNumber),ToFundId=trim(:#T
oFundId),ToUnitHolderIdType=trim(:#ToUnitHolderIdType),ToUnitHolde
rIdNumber=trim(:#ToUnitHolderIdNumber),TransactionCCY=trim(:#Trans
actionCCY),TransactionMode=trim(:#TransactionMode),TransactionValu
e=:#TransactionValue,GrossorNet=trim(:#GrossorNet),UnitsAlloted=:#
UnitsAlloted,NAV=:#NAV WHERE TransactionNumber=:#TransactionNumber
AND MessageDirection='I'
```

On the sender side, data is picked up from the Hyperledger(payload-MFNotifySender) and sent to the Transformer.java where the relevant data is extracted and converted to a HashMap. This data is updated in the sender side 'DLTB_TRANSACTION_DETAIL' table using the sql query and the transaction is committed.

On successful updating of the record in the sender side database, a JSON string which is first converted to a Map is set as a response.

1.2.5 P4: MFHYRECEIVER-IN-TABLE MFHYRECEIVER-OUT-HTTP

Any update in existing record in receiver side database is picked up by below route. This data is sent to parse method of MFParser.java where Row ID is set for later use. The HashMap result of query is then sent to a Freemarker Template which produces the update request for Hyperledger. The template content is appended with a Header and then sent to Hyperledger (URL - <http://localhost:8080/transaction>). The string result of Hyperledger is checked for successful result. Once that condition is passed, Row ID which was set earlier is used to delete the record from table so that route is executed only once. Changes are committed to database.

CamelContext.xml

```
<!-- P4: MFHYRECEIVER-IN-TABLE MFHYRECEIVER-OUT-HTTP -->
<route id="P4:MFRcvr_DBtoREST">
    <from uri="sqlComponent:{{sql.clearingQueryMF1}}" />
    <transform>
        <method ref="parserMF" method="parse" />
    </transform>
    <log message="\${body}" />
    <to uri="freemarker:templateMF2.ftl" />
    <setHeader headerName="Content-Type">
        <constant>application/json</constant>
    </setHeader>
    <log message="\${body}" />
    <to uri="http://localhost:8080/transaction" />
    <convertBodyTo type="String" />
    <choice>
        <when>
            <simple>\${body} contains 'ok'</simple>
            <transform>
                <method ref="parserMF" method="getRowID" />
            </transform>
            <log message="\${body}" />
            <to
uri="sqlComponent:{{sql.deleteRecord_Sender_DBToREST_Update}}" />
                <to uri="sqlComponent:{{sql.commitSql}}" />
            </when>
        </choice>
    </route>
```

Database record is picked up from DLTB_TRANSACTION_DETAIL_OFBA table. Same record is deleted from the table after processing.

MF_SQLConfig.properties

```
sql.clearingQueryMF1 = SELECT ROWID AS  
ROW_ID, SMART_CONTRACT_ID, TRANSACTIONNUMBER, SOURCEENTITY, NVL (DESTIN  
ATIONENTITY, ' ') DESTINATIONENTITY, NVL (STATUS, ' ')  
STATUS, TO_CHAR (TRANSACTIONDATE, 'DD-MON-YY') AS  
TransactionDate, NVL (FUNDID, ' ') FUNDID, NVL (UNITHOLDERIDTYPE, '  
' ) UNITHOLDERIDTYPE, NVL (UNITHOLDERIDNUMBER, ' ')  
UNITHOLDERIDNUMBER, NVL (TOFUNDID, ' ')  
TOFUNDID, NVL (TOUNITHOLDERIDTYPE, ' ')  
TOUNITHOLDERIDTYPE, NVL (TOUNITHOLDERIDNUMBER, ' ')  
TOUNITHOLDERIDNUMBER, NVL (TRANSACTIONCCY, ' ')  
TRANSACTIONCCY, NVL (TRANSACTIONMODE, ' ')  
TRANSACTIONMODE, NVL (TRANSACTIONVALUE, 0)  
TRANSACTIONVALUE, NVL (GROSSORNET, ' ')  
GROSSORNET, NVL (UNITSALLOTTED, 0) UnitsAlloted, NVL (NAV, 0)  
NAV, NVL (MESSAGEDIRECTION, ' ') MESSAGEDIRECTION FROM  
DLTB_TRANSACTION_DETAIL_OFBA WHERE TYPE_OF_TRIGGER ='UPDATE' AND  
ROWNUM <=1 AND STATUS IN ('2','3','4') AND MessageDirection='O'  
ORDER BY SERIAL_NO  
  
sql.deleteRecord_Sender_DBtoREST_Update=delete from  
DLTB_TRANSACTION_DETAIL_OFBA where ROWID=CHARTOROWID(:#rowid)
```

1.2.6 P2: MFSENDER-IN-TABLE-UPDATE MFSENDER-OUT-HTTP-UPDATE

Any update in existing record in sender side database is picked up by below route. This data is sent to parse method of MFParser.java where Row ID is set for later use. The HashMap result of query is then sent to a Freemarker Template which produces the update request for Hyperledger. The template content is appended with a Header and then sent to Hyperledger (URL - <http://localhost:8080/transaction>). The string result of Hyperledger is checked for successful result. Once that condition is passed, Row ID which was set earlier is used to delete the record from table so that route is executed only once. Changes are committed to database.

CamelContext.xml

```
<!-- P2: MFSENDER-IN-TABLE-UPDATE MFSENDER-OUT-HTTP-UPDATE -->  
<route id="P2:MFSENDER_DBtoRESTUpdate">  
    <from  
uri="sqlComponent:{{sql.clearingQuery_Sender_DBtoREST_Update}}" />  
    <transform>  
        <method ref="parserMF" method="parse" />  
    </transform>  
    <to uri="freemarker:templateMF2.ftl" />  
    <log message="${body}" />  
    <setHeader headerName="Content-Type">  
        <constant>application/json</constant>  
    </setHeader>
```

```

<to uri="http://localhost:8080/transaction" />
<convertBodyTo type="String" />
<choice>
    <when>
        <simple>${body} contains 'ok'</simple>
        <transform>
            <method ref="parserMF" method="getRowID" />
        </transform>
        <log message="${body}" />
        <to
uri="sqlComponent:{{sql.deleteRecord_Sender_DBToREST_Update}}" />
        <to uri="sqlComponent:{{sql.commitSql}}" />
    </when>
</choice>
</route>

```

MF_SQLConfig.properties

Database record is picked up from DLTB_TRANSACTION_DETAIL_OFBA table. Same record is deleted from the table after processing.

```

sql.clearingQuery_Sender_DBToREST_Update=SELECT ROWIDTOCHAR(ROWID)
AS
ROW_ID,SMART_CONTRACT_ID,SOURCEENTITY,DESTINATIONENTITY,STATUS,TRA
NSACTIONNUMBER,TRANSACTIONDATE,FUNDID,UNITHOLDERIDTYPE,UNITHOLDERI
DNUMBER,TOFUNDID,TOUNITHOLDERIDTYPE,TOUNITHOLDERIDNUMBER,TRANSACTIONCCY,TRANSACTIONMODE,TRANSACTIONVALUE,GROSSORNET,UNITSALLOTTED,NAV
,NVL(MESSAGEIDIRECTION, ' ') MESSAGEIDIRECTION FROM
DLTB_TRANSACTION_DETAIL_OFBA WHERE TYPE_OF_TRIGGER ='UPDATE' AND
ROWNUM <=1 AND MessageDirection='I' ORDER BY SERIAL_NO ASC

sql.deleteRecord_Sender_DBToREST_Update=delete from
DLTB_TRANSACTION_DETAIL_OFBA where ROWID=CHARTOROWID(:#rowid)

```

1.2.7 P1: MFRECEIVER-IN-HTTP-UPDATE MFRECEIVER-OUT-TABLE-UPDATE

This route notifies the receiver about successful posting of a transaction from receiver to Hyperledger. Rest Endpoint related configurations are required to access result of that web service in route. Updated record details sent through web service are sent to transform method of MFTransformer.java. This method extracts data from HashMap input and constructs a new HashMap containing required fields only. Receiver side database is then updated using SQL query and changes are committed. A successful response message is then sent back after converting it to HashMap from String datatype.

CamelContext.xml

```
<restConfiguration bindingMode="json" component="servlet" />
<rest path="/rest">
    <post uri="/MFNotifyReceiver" consumes="application/json"
        produces="application/json">
        <to uri="direct:MFNotifyReceiver" />
    </post>
</rest>
<!-- P1: MFRECEIVER-IN-HTTP-UPDATE MFRECEIVER-OUT-TABLE-UPDATE -->
<route id="P1:MFRcvr_RESTtoDBUpdate">
    <from uri="direct:MFNotifyReceiver" />
    <transform>
        <method ref="transformerMF" method="transform" />
    </transform>
    <log message="${body}" />
    <to
        uri="sqlComponent:{{sql.updateQuery_Recvr_RESTtoDB_Update}}" />
    <to uri="sqlComponent:{{sql.commitSql}}" />
    <setBody><simple>{"result":"ok"}</simple></setBody>
    <transform>
        <method ref="transformerMF" method="stringToMap" />
    </transform>
</route>
```

MF_SQLConfig.properties

Above query updates DLTB_TRANSACTION_DETAIL table on receiver side.

```
sql.updateQuery_Recvr_RESTtoDB_Update=UPDATE
DLTB_TRANSACTION_DETAIL SET
SMART_CONTRACT_ID=:#SmartContractID,STATUS=trim(:#Status),Transac
tionDate=TO_DATE(:#TransactionDate,'DD-MON-
YY'),FundId=trim(:#FundId),UnitHolderIdType=trim(:#UnitHolderIdTy
pe),UnitHolderIdNumber=trim(:#UnitHolderIdNumber),ToFundId=trim(:
#ToFundId),ToUnitHolderIdType=trim(:#ToUnitHolderIdType),ToUnitHo
lderIdNumber=trim(:#ToUnitHolderIdNumber),TransactionCCY=trim(:#T
ransactionCCY),TransactionMode=trim(:#TransactionMode),Transactio
nValue=:#TransactionValue,GrossorNet=trim(:#GrossorNet),UnitsAllo
ted=:#UnitsAlloted,NAV=:#NAV WHERE
TransactionNumber=:#TransactionNumber AND MessageDirection='O'
```

1.3 APIs

1.3.1 Posting new transaction

1.3.1.1 Sender's end

Sender adds a new transaction in hyperledger. First an entry is added to the table "DLTB_TRANSACTION_DETAIL" and "DLTB_TRANSACTION_DETAIL_OFBA" and this record is processed by the routes which are mentioned above. From the sender's end a request is sent to hyperledger in following format:

Request from sender's adapter to hyperledger

```
{
  "chainCodeName": "OracleTrnsctnChainCode",
  "functionName": "addTransaction",
  "arguments": [
    "{\\"
    SMRT_CONTRACT_TXNTBL_PK\\": [\\"
    transactionNumber123\\",\\"
    DIST1\\",\\"
    AMC\\"]\\"]\\",
    "{\\"
    SMRT_CONTRACT_TXNTBL\\": [\\"
    1\\",\\"
    07 - JAN 18\\",\\"
    f_id12\\",\\"
    UHidType\\",\\"
```

```

UHidNum\","\"
to_fId\","\"
to_UHidType\","\"
to_UHidNum\","\"
EUR\","\"
M\","\"
G\","\"
I\","\"
\","\"
\","\"
\","\"
\","\"
\"]]}",
"{\"
privateAllocation\":[[\"
\","\"
100\","\"
150\","\"
120\","\"
UDF_1\","\"
UDF_2\","\"
UDF_3\"]]}"
]
}

```

If transaction is posted successfully then response from hyperledger is received in following format:

Response from hyperledger to Sender's adapter

```
{
  "payload": {
    "SmartContractID":
    "13347d7276ccfb220efa52ffb5ed78b26fa23c9f0698ecfc185d3a84dec313dc
    "
  },
  "status": 200
  "result": "ok"
}
```

Smart contract ID is updated in "DLTB_TRANSACTION_DETAIL" table and the same record is deleted from "DLTB_TRANSACTION_DETAIL_OFBA" table after processing.

1.3.1.2 Receiver's end

Receiver receives the following message from hyperledger which is processed and an entry is made to "DLTB_TRANSACTION_DETAIL" table on the receiver's end.

Request from hyperledger to receiver's adapter

```
{
  "message": {
    "SMRT_CONTRACT_TXNTBL": [{
      "FundId": "F",
      "GrossNet": "",
      "MessageDirection": "I",
      "Status": "S",
      "ToFundId": "",
      "ToUnitHolderIdNumber": "",
      "ToUnitHolderIdType": "",
      "TransactionCCY": "T_ccy",
      "TransactionDate": "07-JAN-18",
      "TransactionMode": "",
      "UDF_1": "",
      "UDF_2": "",
      "UDF_3": "",
      "UDF_4": "",
      "UDF_5": "",
    }
  ]
}
```



```

        "UnitHolderIdNumber": "",
        "UnitHolderIdType": ""
    }],
    "SMRT_CONTRACT_TXNTBL_PK": [{
        "FromBankCode": "AMC",
        "ToBankCode": "DIST2",
        "TransactionNumber": "txnnum100"
    }],
    "privateData": {
        "privateAllocation": {
            "NAV": "",
            "TransactionValue": "",
            "UDF_1": "",
            "UDF_2": "",
            "UDF_3": "",
            "UnitsAlloted": ""
        }
    },
    "functionName": "addTransaction",
    "SmartContractID":
    "482c0bd1d2e6c5430afa372c73b54163a8c4db6963adbfa772a6ae19bab053ff
    "
    }
}

```

Following is the response sent from the receiver's adapter to hyperledger:

Response from receiver's adapter to hyperledger

```

{"result": "ok"}

```

At this point add transaction flow gets completed.

1.3.2 Update transaction

1.3.2.1 Receiver's end

When receiver updates few entries in a record which exists in "DLTB_TRANSACTION_DETAIL" table. First an entry is added to the table "DLTB_TRANSACTION_DETAIL_OFBA" and this record is processed by the routes which are mentioned above. From the receiver's end a request is sent to hyperledger in following format:

Request from receiver's adapter to hyperledger

```
{
  "chainCodeName": "OracleTrnsctnChainCode",
  "functionName": "mainUpdate",
  "arguments": [
    "Smrt_Contract_Id_452af57d700baa8e887cc4343947b64b36f6f913700086455c5191738a2c7aa8",
    "{\n
      SMRT_CONTRACT_TXNTBL\ ": [[\n
        3 \",\n
        08 - JAN - 18 \", \n
        f_id12 \", \n
        UHidType \", \n
        UHidNum \", \n
        to_fId \", \n
        to_UHidType \", \n
        to_UHidNum \", \n
        EUR \",\n
        A \",\n
        B \",\n
        O \",\n
        \",\n
        \",\n
        \",\n
        \",\n
        \"]]]",
    "{\n
      privateAllocation \": [[\n
        78 \",\n
        1500 \",\n
```

```

        1200 "\",\"
        \",\"
        \",\"
        \"]]]\"
    ]
}

```

If transaction is posted successfully then response from hyperledger is received in following format:

Response from hyperledger to receiver's adapter

```

{"result": "ok"}

```

1.3.2.2 Sender's end

Sender receives the following message from hyperledger which is processed and entry is updated in "DLTB_TRANSACTION_DETAIL" table.

Response from hyperledger to the sender's adapter

```

{
    "message": {
        "SMRT_CONTRACT_TXNTBL": [{
            "FundId": "f_id12",
            "GrossNet": "G",
            "MessageDirection": "I",
            "Status": "S",
            "ToFundId": "to_fId",
            "ToUnitHolderIdNumber": "to_UHidNum",
            "ToUnitHolderIdType": "to_UHidType",
            "TransactionCCY": "EUR",
            "TransactionDate": "07-JAN-18",
            "TransactionMode": "M",
            "UDF_1": "",
            "UDF_2": "",
            "UDF_3": "",
            "UDF_4": "",
            "UDF_5": "",
            "UnitHolderIdNumber": "UHidNum",
            "UnitHolderIdType": "UHidType"
        }
    ]
}

```

```

    }],
    "SMRT_CONTRACT_TXNTBL_PK": [{
        "FromBankCode": "AMC",
        "ToBankCode": "DIST2",
        "TransactionNumber": "txnnum100"
    }],
    "privateData": {
        "privateAllocation": {
            "NAV": "150",
            "TransactionValue": "50",
            "UDF_1": "",
            "UDF_2": "",
            "UDF_3": "",
            "UnitsAlloted": "3"
        }
    },
    "functionName": "addTransaction",
    "SmartContractID":
    "482c0bd1d2e6c5430afa372c73b54163a8c4db6963adbfa772a6ae19bab053ff
    "
    }
}

```

Following is the response sent from the sender's adapter to hyperledger:

Response from sender's adapter to hyperledger

```

{"result": "ok"}

```

At this point add transaction flow gets completed.

1.3.3 **Fund and Entity Configuration**

There are two different maintenance required.

1. Transfer Agency (Source)

An Entry is required in DLTM_TXN_OFFUS_FUNDS_DTL at the Transfer Agency side.

The Table should have a valid source entity, destination entity and fund ID.

Based on the above-maintained combination, the transaction will be picked.

1. SOURCE ENTITY – Bank ID given During Distributed Ledger property file
2. DESTINATION ENTITY – Bank ID given During Distributed Ledger property file
3. FUNDID – Fund ID for which Smart contract transaction needs to generate

Note: Distributed Ledger configuration Not in Scope

2. AMC (Destination)

An Entry is required in DLTM_ONUS_FUNDS_MAINT at the AMC side.

The above table should have an entry for the transaction to be processed.

- FUND ID – Same as the Transfer Agency Fund ID
- SOURCE ENTITY – Same source entity given at the Transfer Agency side
- TXN USERID – FCIS application User ID. This User will go as maker id for the transaction generated at the AMC side
- MODULE ID- Valid module for the transaction
- CASHNOMINEE ID- At the Transfer Agency side, if the Unitholder type is maintained as 'C' then this ID will be sent as Unitholder ID for the Transaction
- UNITHOLDER TYPE- This should match with the Unitholder type maintained at the Transfer Agency side



Blockchain Adapter for Mutual Fund Transactions
[January] [2019]
Version 14.0.0.2.5

Oracle Financial Services Software Limited
Oracle Park
Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India

Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax: +91 22 6718 3001
www.oracle.com/financialservices/

Copyright © [2007], [2019], Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

