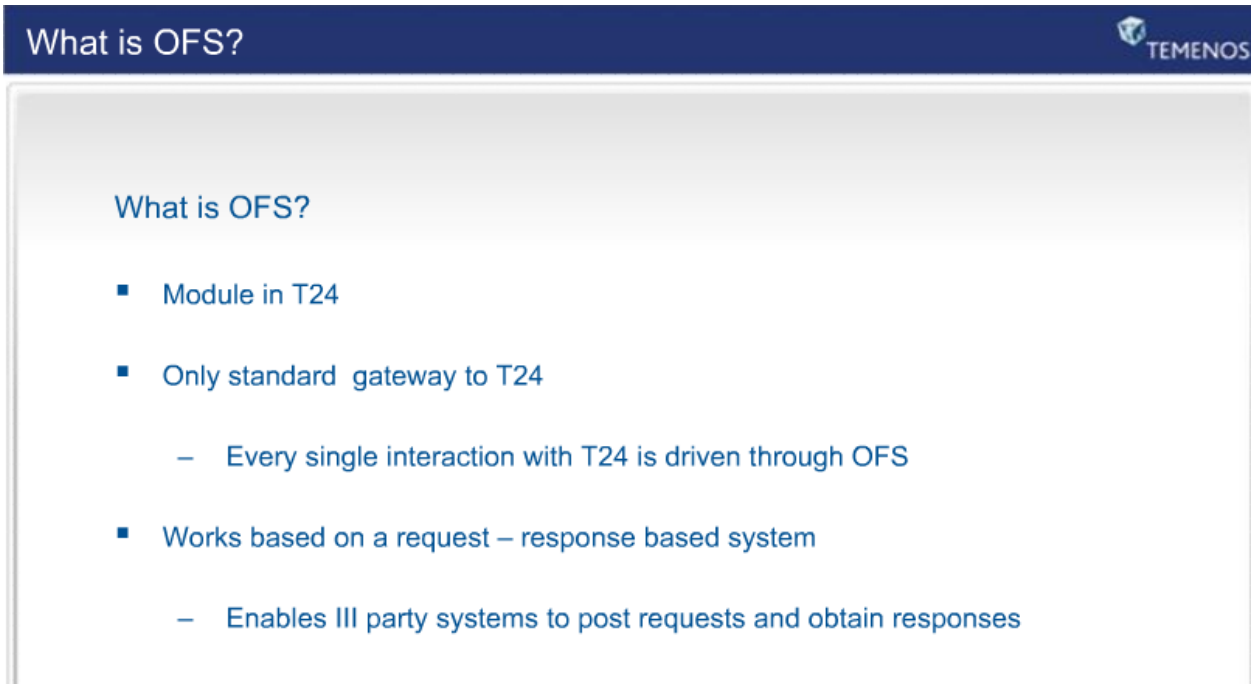
Introduction and Common OFS Messages

What is OFS? Is it something outside of T24? What does it comprise of?

Well, OFS is a standard module within T24, with a module code OF. It is the ONLY standard gateway to TEMENOS T24. Now, you may be thinking what does that mean? Simply put, it means that every single interaction with T24 is driven through OFS. The second piece of OFS are messages. OFS is message driven , i.e. it works on a request-response based system. The OFS syntax, or message structure is proprietary to TEMENOS T24. It is the native way to represent requests to execute T24 transactions, enquiries or routines.

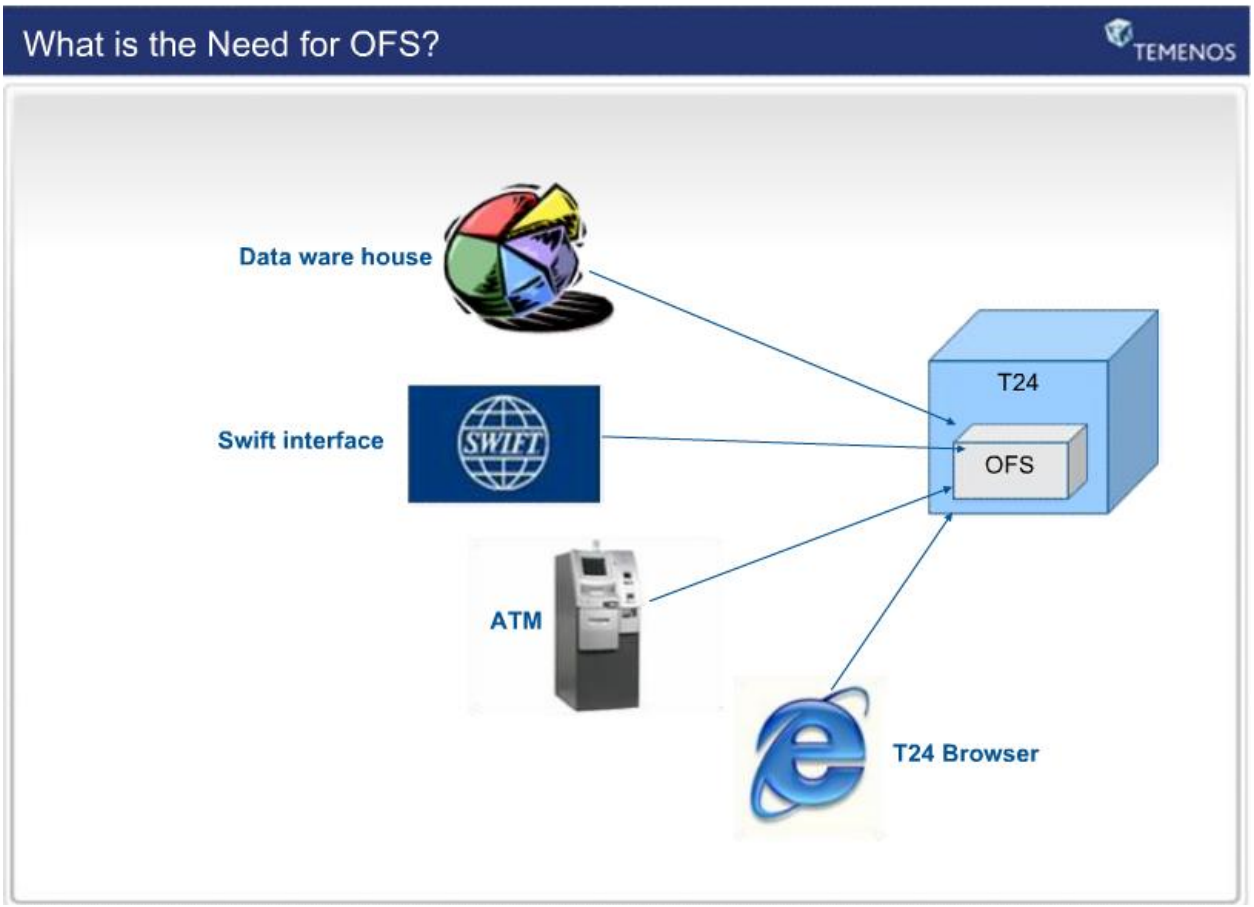
The OFS module provides the infrastructure necessary to process the OFS messages. It has many components which we will see as we go through this course.



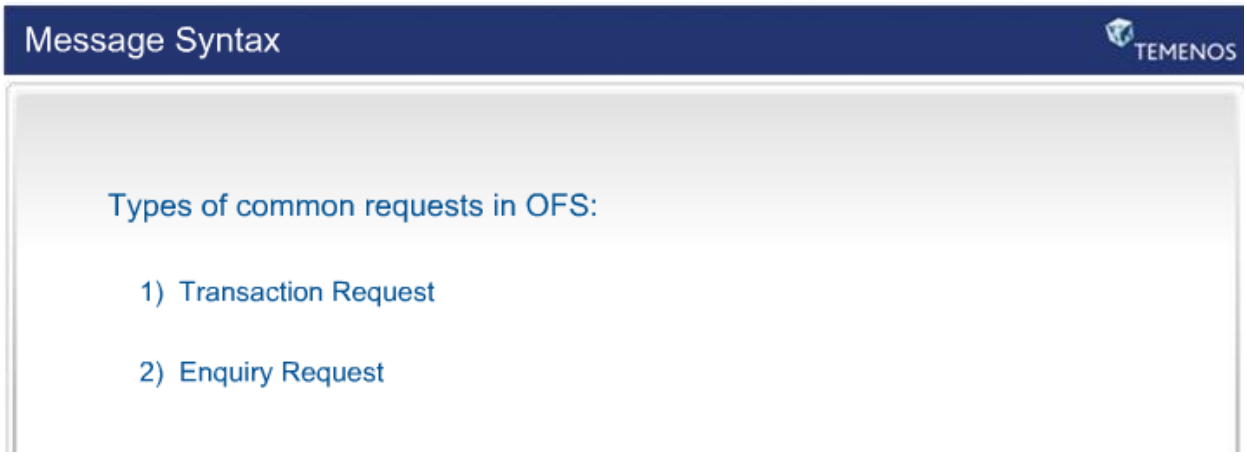
What is the need for OFS?

TEMENOS T24 technology platform products like TEMENOS Connector, TEMENOS T24 Browser and TEMENOS Internet Bank use OFS to interact with TEMENOS T24 Applications. This implies that when we input a transaction, say a Funds Transfer in Browser, T24 actually receives an OFS message for Funds Transfer. This also implies that the OFS processing is not merely an update to the database, but goes through the same level of validations, including SMS, as in a manual effort.

Secondly, with OFS, it becomes possible for T24 to transact with and reply to queries from various external channels (or systems). This is especially handy when we have huge volumes of transaction to be updated, or when there are systems like ATM switches and other banking systems requiring a 24/7 connection to T24.



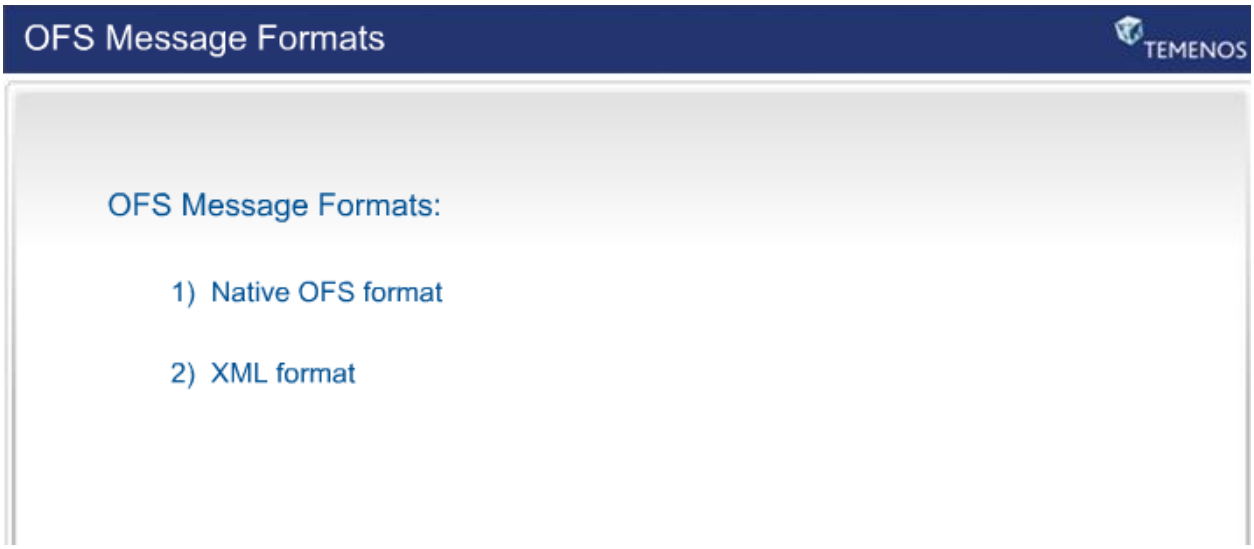
We have seen that OFS messages are requests to T24 . There are two types of common requests in OFS. Each request follows a particular syntax which you will learn as you go through this session. One is a transaction request that will create, modify or delete a record in any application in T24. The other is an enquiry request, which queries data from T24.



OFS messages follow two formats:

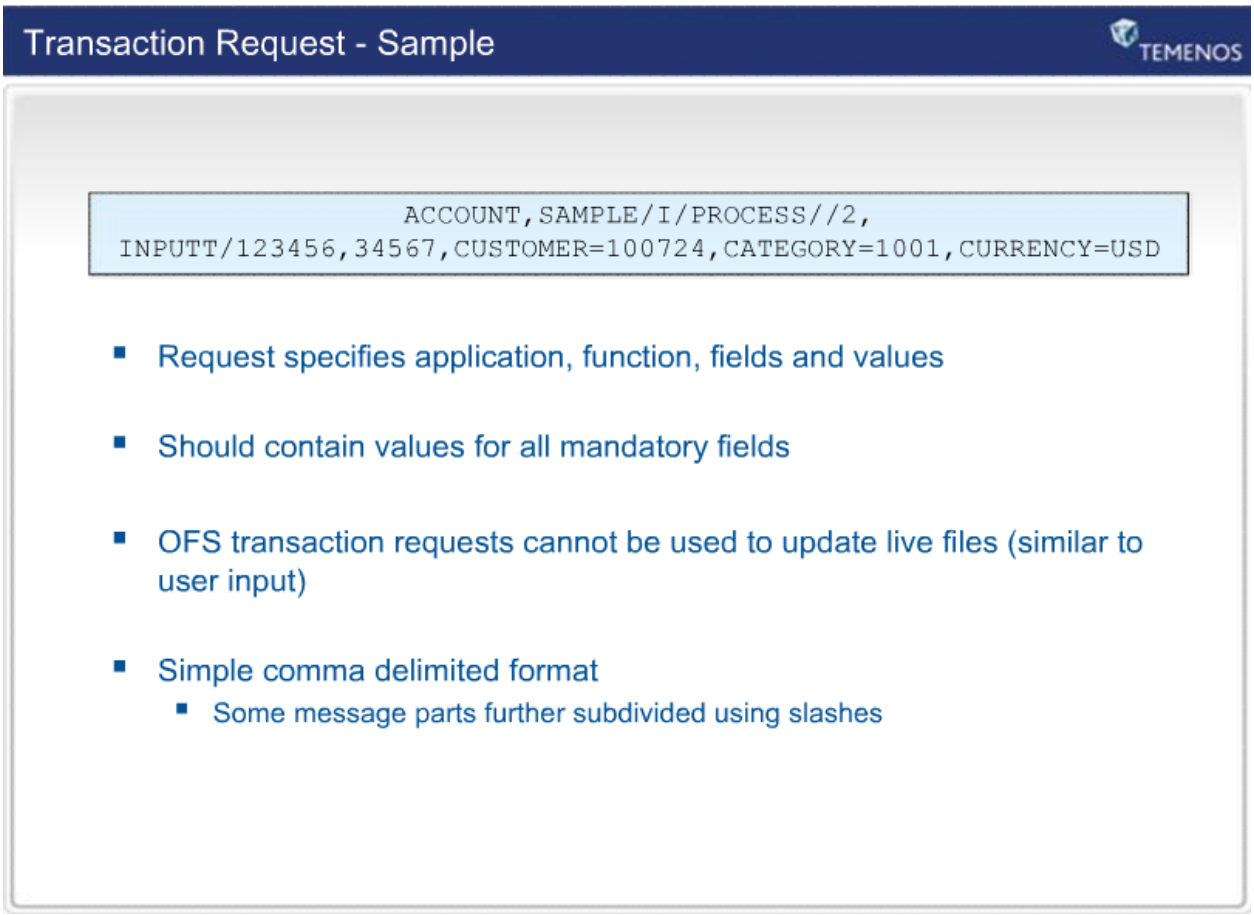
1) Native format – this is a simple string format, following a comma delimited pattern

2) XML format – OFS also understands an XML format which is described as the Browser XML format since it is used only by Temenos Browser.



Look at the sample OFS request shown. This is a request to update an account (which is not a live application). It contains all the mandatory fields for the account such as customer number, category code and currency. It also contains other details essential for a transaction such as the T24 user id, password and the id of the account to be updated.

You can also see that the OFS messages follow a simple comma delimited format. Some parts of the message are divided further using a forward slash character. We will look at the syntax in detail as we move on.

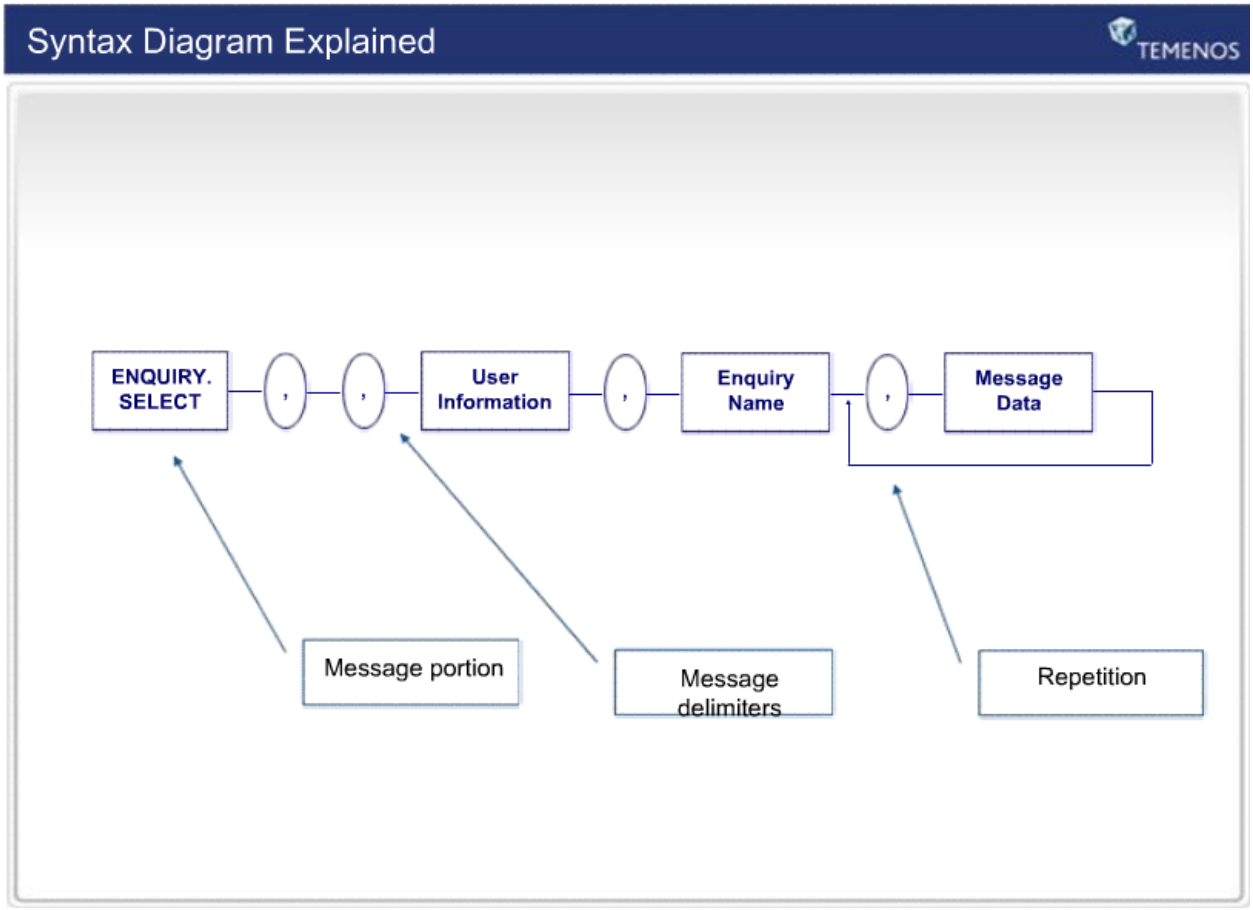


We have used syntax diagrams in this course to describe OFS messages. You will now learn the purpose behind the symbols used within a syntax diagram. The sample shown is a syntax diagram for an ENQUIRY.

1. The rectangular boxes indicate message portions. For example, ENQUIRY.SELECT is one part, user information is another.

2. The ovals indicate message delimiters i.e. the comma is used as delimiter in this example

3. The arrow indicates repetition or multiple instances i.e. In the above OFS message, message data may occur many times



This diagram describes the syntax of a Transaction request.

Operation

We need to tell T24 the name of the application. Operation part of the message contains the name of the Application. Eg: ACCOUNT.

Options

Options is optional since many of the parameters here may be defaulted. This contains many sub-parts with each sub-part separated from the other using a forward slash. The sub-parts are Version name/Function/Process type/ GTS.Control value/No.of.authorisers. E.g. TRG/I/VALIDATE//2

Version name – this must a valid version name without the usual preceding application name and comma. For eg to use a version name ACCOUNT,CLIENT you will only specify CLIENT. Therefore comma versions are not valid.

Function – a T24 function such as I, R, A,H,V. Note that C and P is not supported.

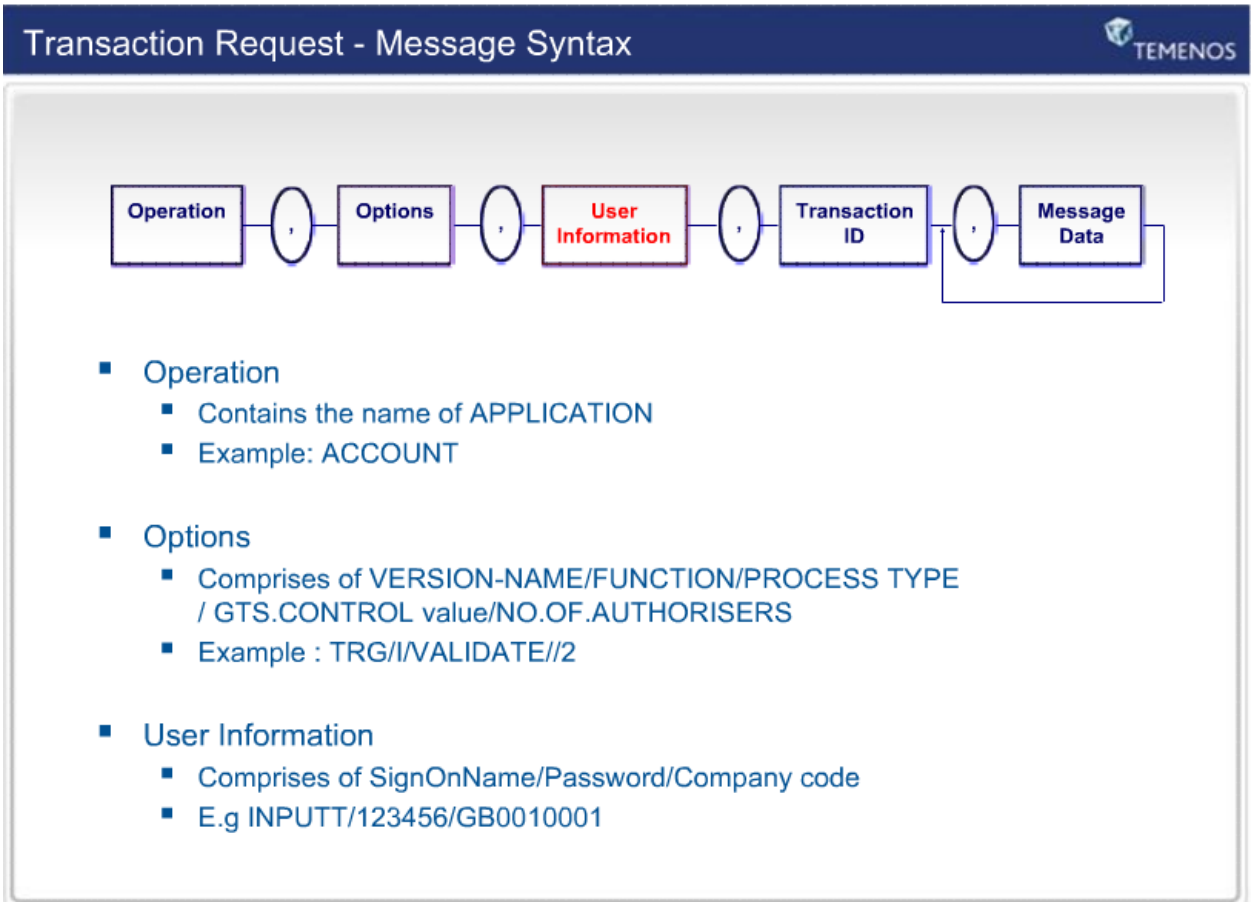
Process Type – this may be VALIDATE or PROCESS and controls whether the transaction is to be validated or processed.

Gts control will be discussed later. No of authorisers may be used to specify the no of authorisers . I.e. zero, one or two. GTS control and no of authorisers if supplied will override the corresponding settings in the Version.

User Information

This consists of the Sign On Name/Password/Company code

E.g INPUTT/123456/GB0010001



Transaction ID:

The Transaction Id part of the message contains the transaction id of the record used in the transaction. This is optional for new transactions if the underlying application is designed to allow this. In such a case where it is allowed, the ID may be automatically generated by the application. The transaction ID is mandatory for See, Authorize and Delete.

The transaction ID can also contain an optional Message ID for the message.

For example, 20548/20081010001. In this case, the 20548 is the record id and 20081010001 is the message id. This message id could be used by external systems to uniquely identify each OFS message

Message Data:

The message data portion of the message structure contains the data required to create or update the transaction.

Message portion follows the format:

Fieldname=data

For example, CUSTOMER=100297

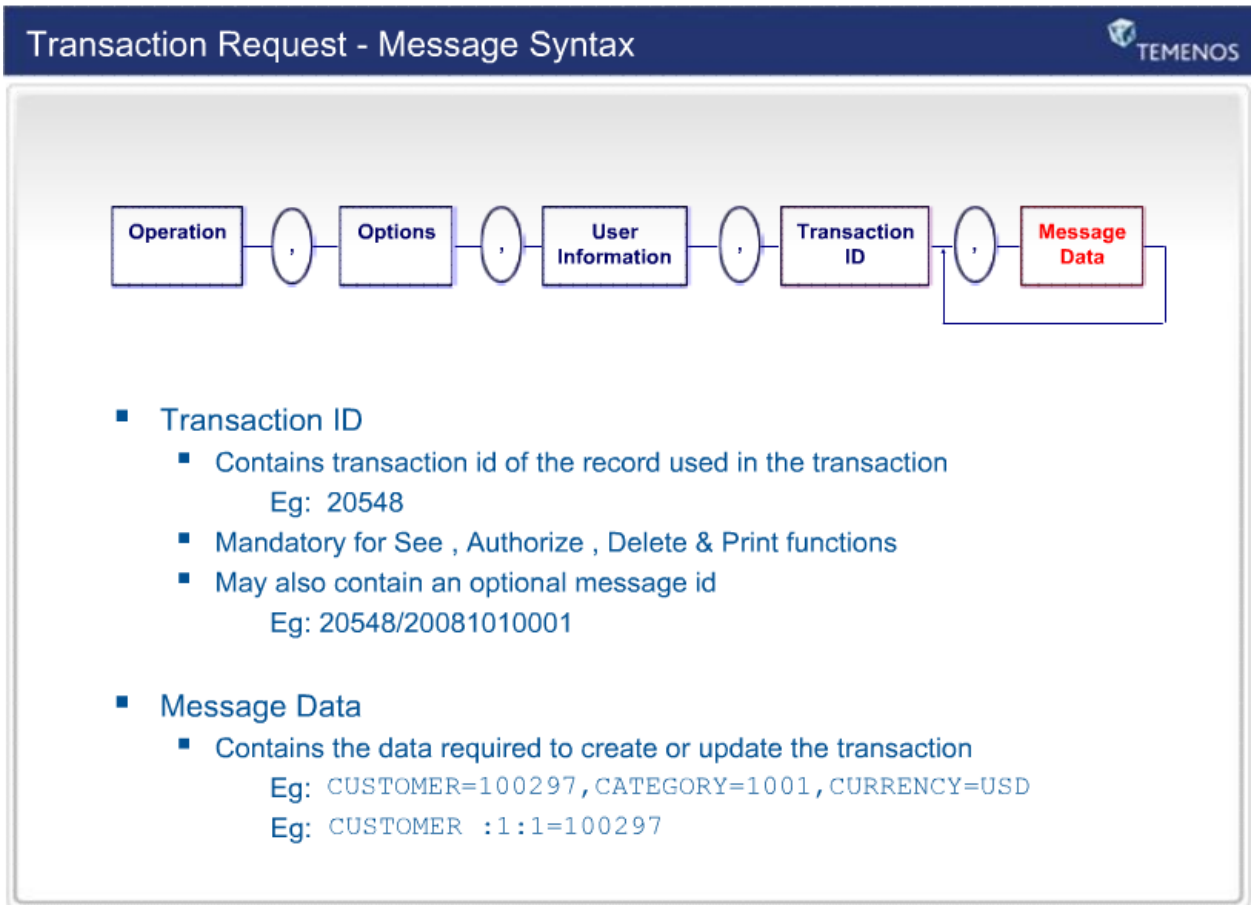
In cases where you need to assign values to multi values or sub values, you may follow the format:

Fieldname : multi value number : sub value number = data

For example, CUSTOMER:1:1=100297

This implies the first sub value belonging to the first multi value of the field CUSTOMER is assigned a value of 100297. The first multi value or sub value is taken as the default in case the multi value or sub value positions are not mentioned in the message.

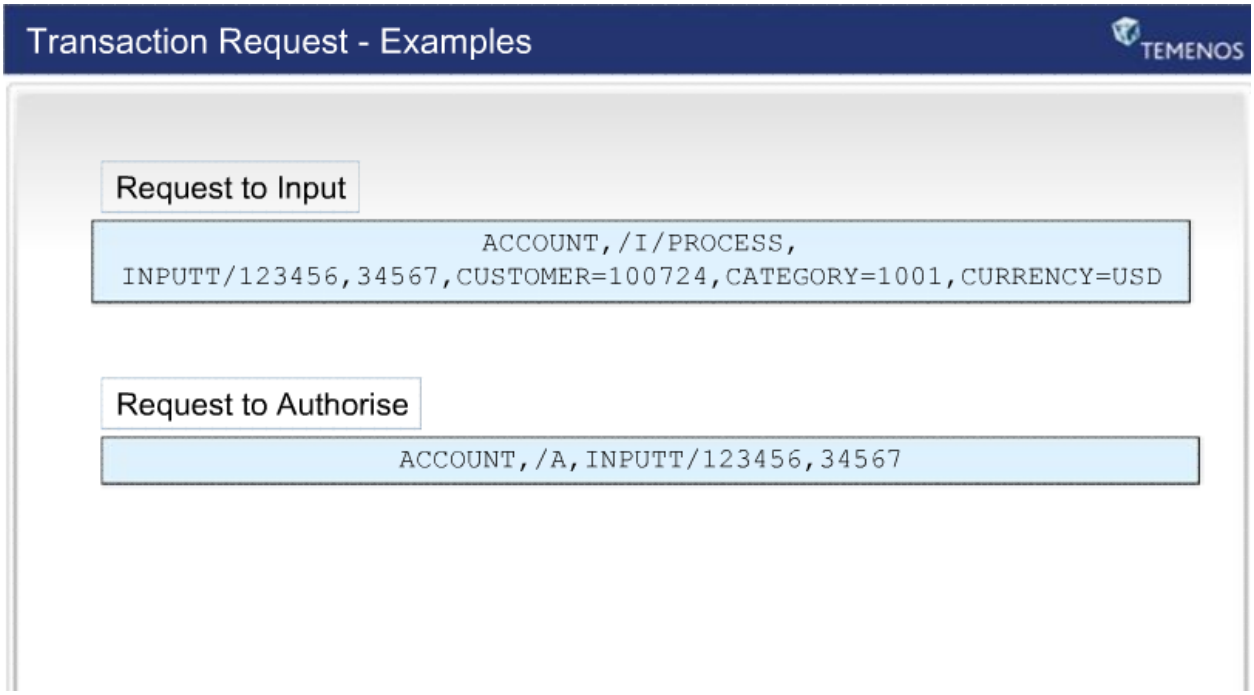
If 'NULL' is specified as field data, OFS will blank the field of all data. The message data portion of the message can be repeated for each field separated by a comma (,).



Look at the sample requests shown.

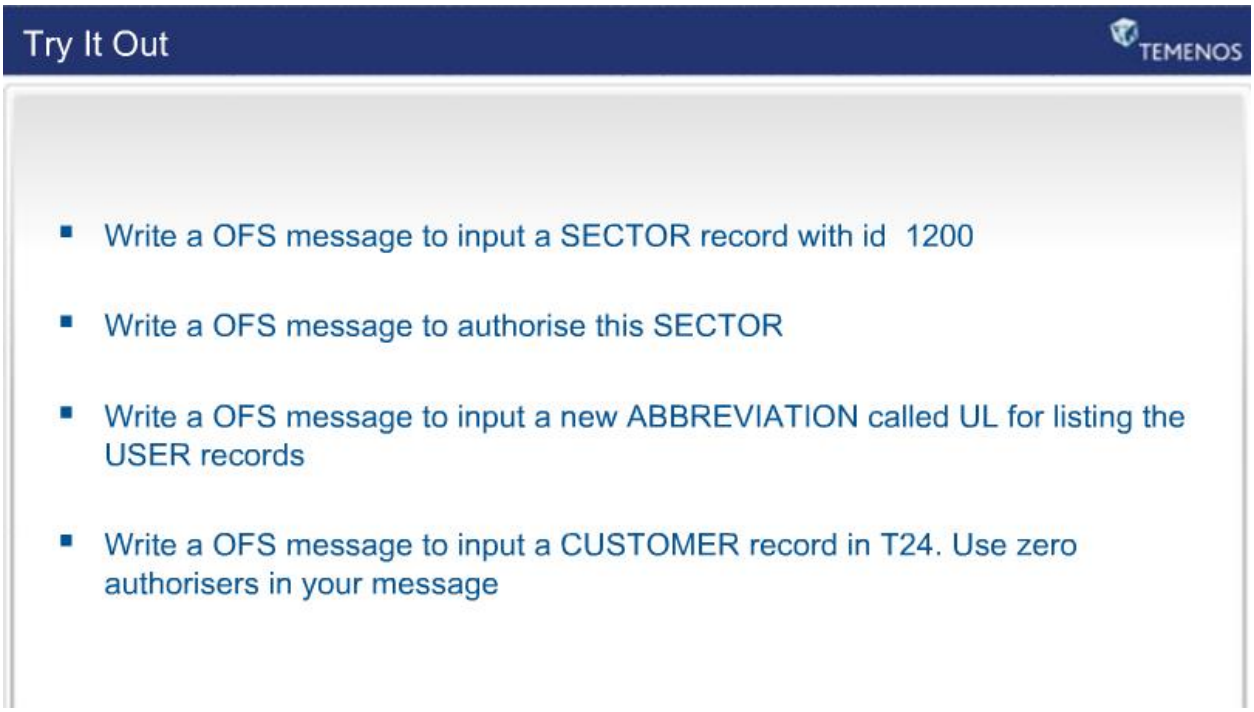
The first one is a sample request to input an account record. This has not used a version. The mandatory field values have been specified.

The second request is one to authorise the record previously input. This does not contain values in the data portion. PROCESS has not been specified either since that is the default process type.



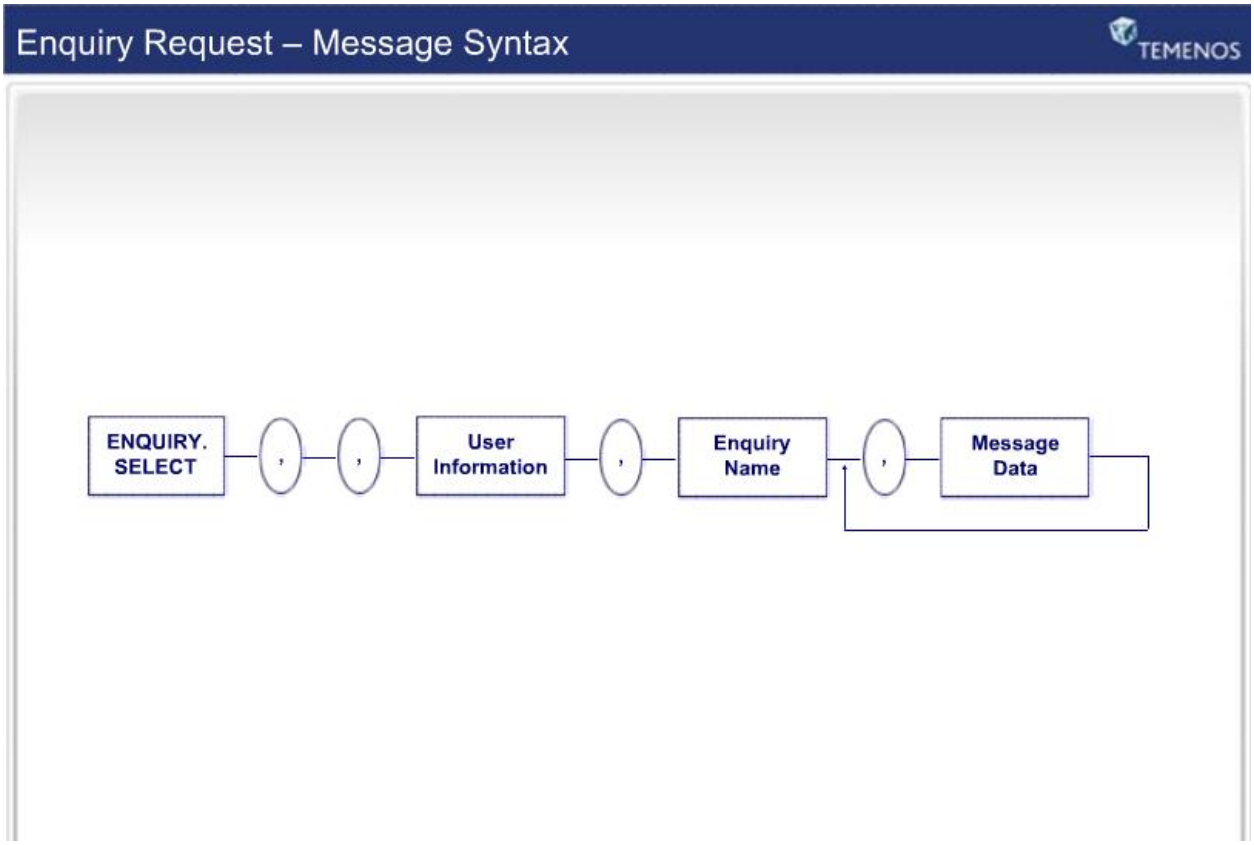
Now you try to write an OFS message to input a SECTOR record with id 1200; write a OFS message to authorise this SECTOR; write a OFS message to input a new ABBREVIATION called UL for listing the USER records; write a OFS message to input a CUSTOMER record in T24. Use zero authorisers in your message.

Hint : Open the applications from browser or classic. Identify the mandatory fields. Then write the messages.



The diagram shows the syntax of OFS Enquiry type request messages. All the main portions of the message for Enquiry type requests are shown in it.

The structure is very much similar to a Transaction Request. You may have noticed that the options part has been omitted, since it is not relevant to an enquiry. However commas are given to indicate the placeholder and retain the general structure of an OFS message.



The portions of an Enquiry Request are:

ENQUIRY.SELECT:

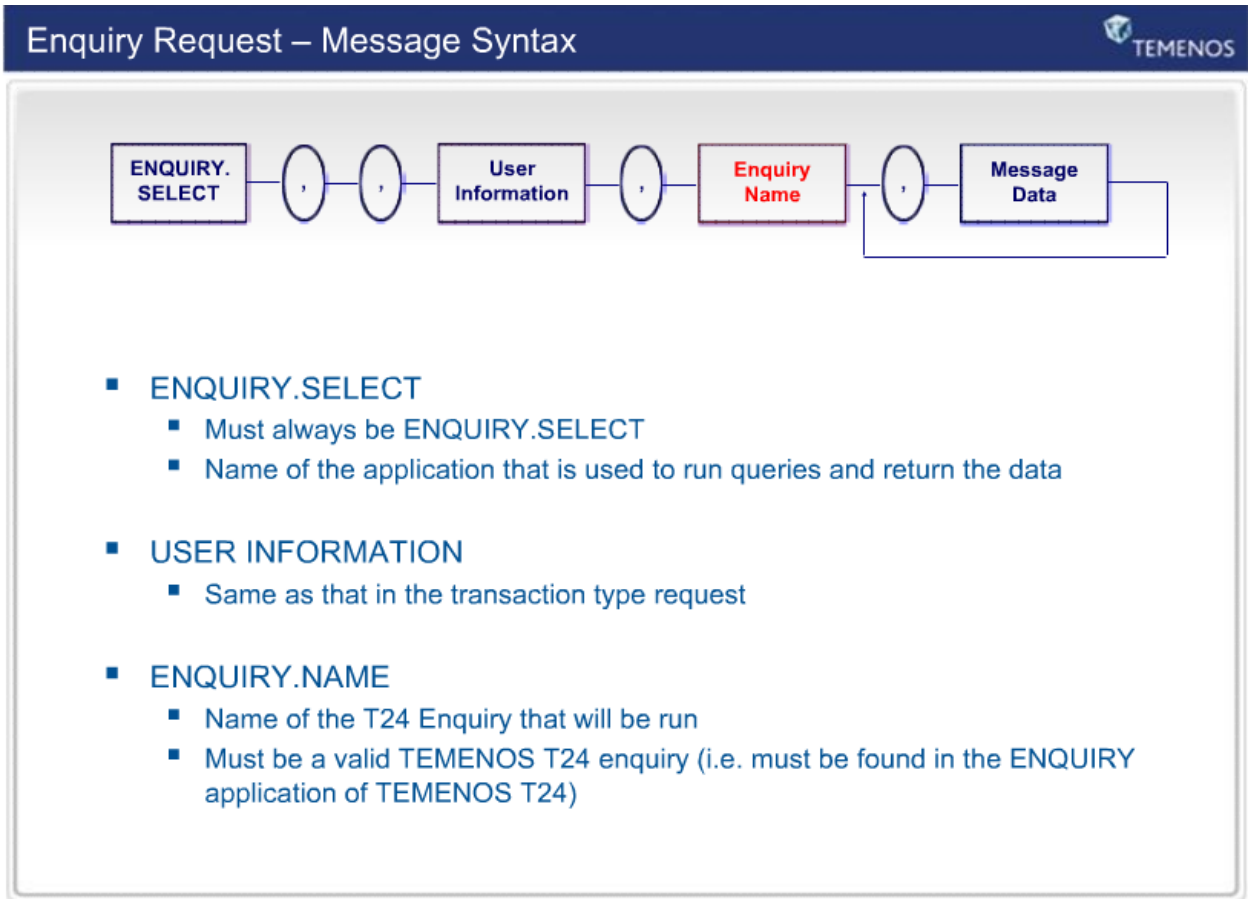
The first portion of an enquiry type request must always be ENQUIRY.SELECT. This is the name of the routine that is used to run queries and return the data.

User information:

The user information portion of the message structure is the same as that of the transaction type request.

Enquiry Name:

You will specify the name of the T24 Enquiry to run in this part. The Enquiry name supplied here must be a valid T24 enquiry (i.e. it must be found in the ENQUIRY application of T24).



What does the message data portion of the enquiry request contain?

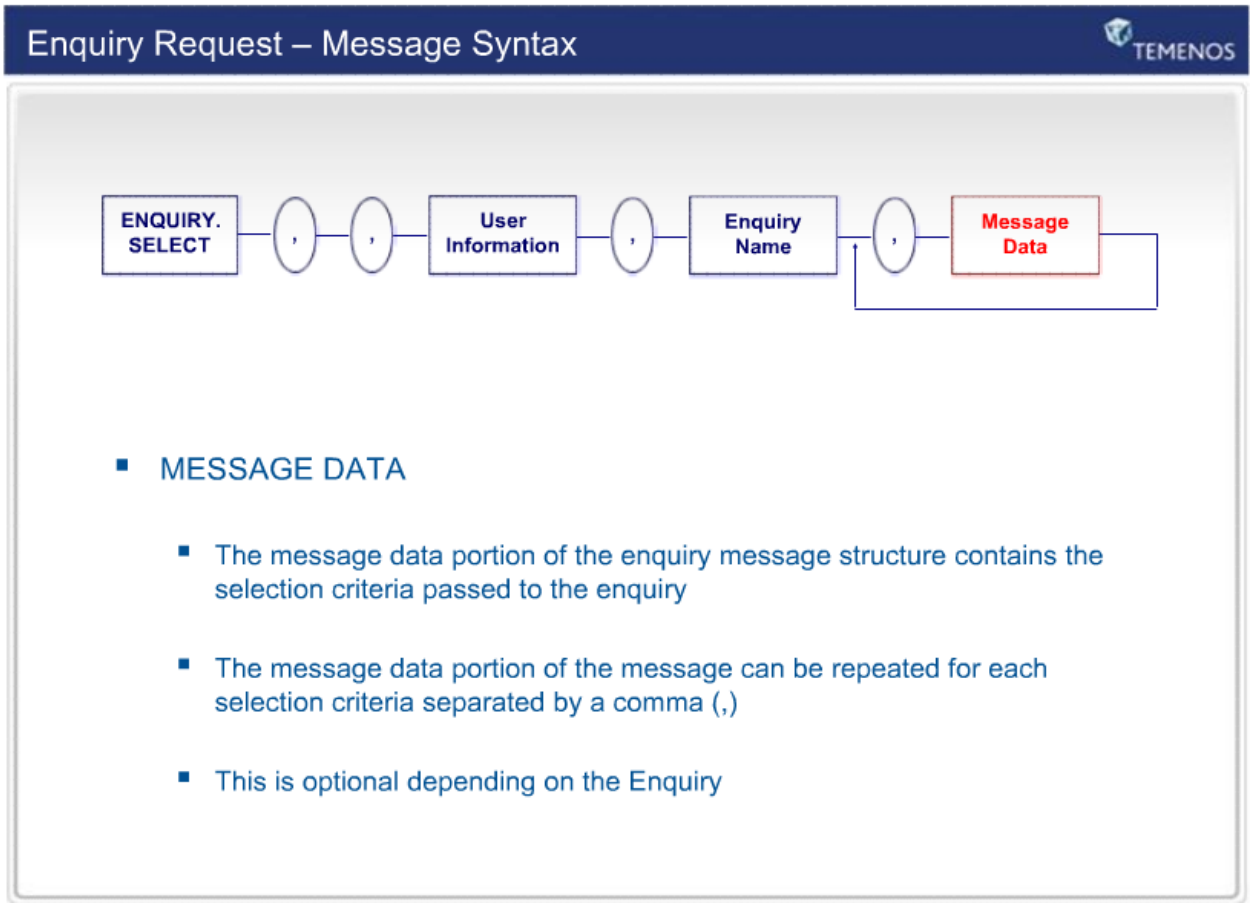
The message data portion of the enquiry message structure contains the selection criteria passed to the enquiry. The message data portion of the message can be repeated for each selection criteria separated by a comma (,). This is optional depending on the Enquiry.

The three different parts of a Selection Criteria part of the MESSAGE DATA are:

1) Selection Field – denotes the name of the field of that must be either a field in the STANDARD.SELECTION for the file on which the enquiry is based (i.e. the application that is specified in FILE.NAME field of the ENQUIRY application)or a value set in SELECTION.FLDS field of the ENQUIRY application.

2) Operand – denotes the operand that must be used for selection for the value specified in the SELECTION.FLDS field of the ENQUIRY application. The operands can be EQ, NE, GE, GT, LE, LT, UL, LK and NR. The operand must be separated from the selection fields using a colon (:).

3) Field Value – denotes the data value for the values specified in the SELECTION.FLDS (field of the ENQUIRY application) and the operand for selection. This must be separated from the operand using a equal sign (=). For example, ACCOUNT.NUMBER:EQ=11107.



This is an enquiry request to call the default account enquiry

ENQUIRY.SELECT,,INPUTT/654321,%ACCOUNT

This is the same enquiry request with a criteria:

ENQUIRY.SELECT,,INPUTT/654321,%ACCOUNT, ACCOUNT.NUMBER:EQ=11109

Now you try to write an enquiry type request to find the list of INDUSTRY records found in yourT24 area. Also you write an enquiry type request to find the current day's balance summary of an ACCOUNT (say for example 29987).

TIP: Use the enquiry ACCT.BAL.TODAY.

Try It Out

- Write an Enquiry type request to find the list of INDUSTRY records found in yourT24 area
- Write an Enquiry type request to find the current day's balance summary of an ACCOUNT (say for example 29987)

Important Points

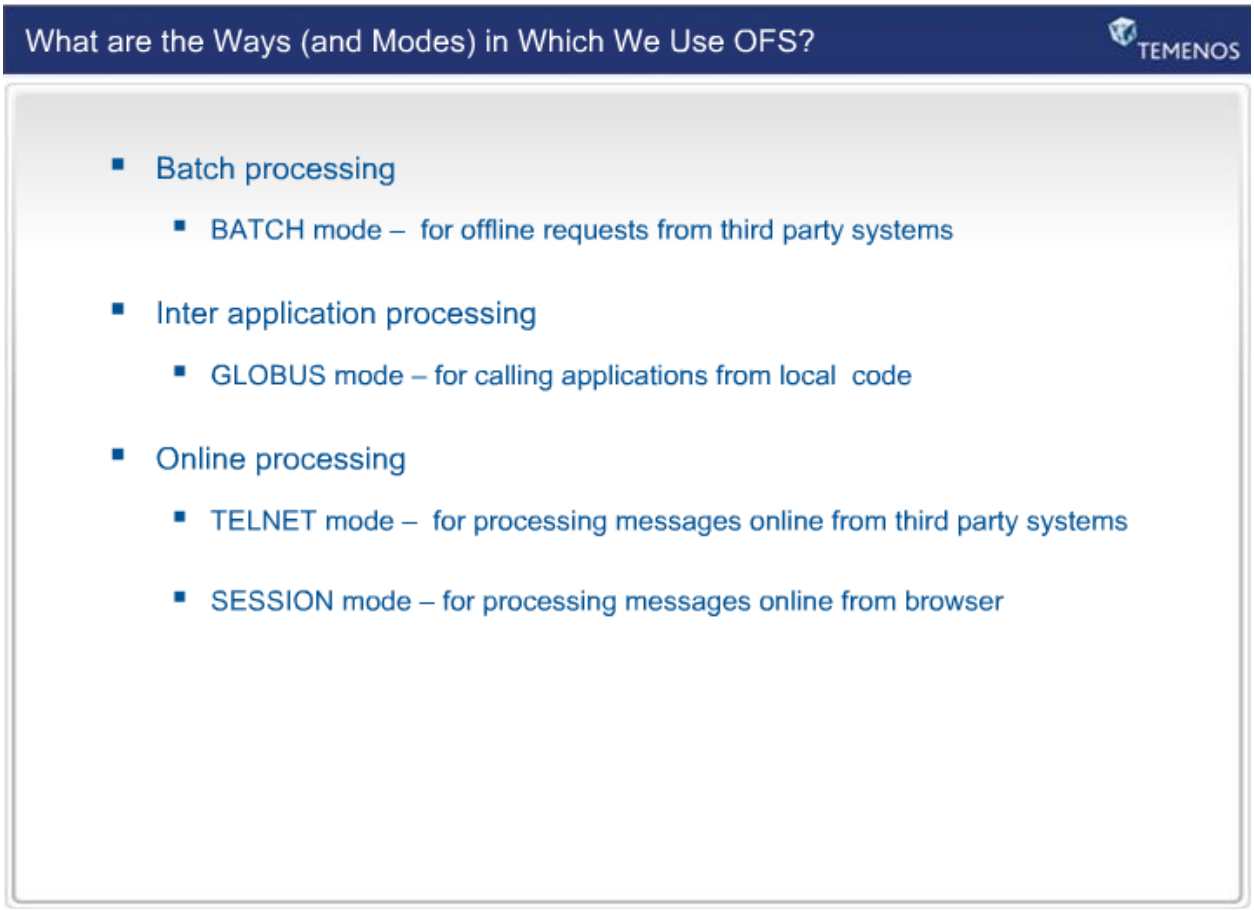
- Common OFS message types
  - Transaction message
  - Enquiry message
- Transaction message parts
  - Operation
  - Options
  - User information
  - Transaction Id
  - Message Data
- You cannot use comma versions in a OFS request
- Enquiry request
  - Always starts with ENQUIRY.SELECT
  - Does not contain options
  - May pass criteria also

OFS modes – basics

OFS can be used in three different ways with four different modes.

1) The first way we can use OFS is in Batch processing. Batch processing, as the name implies, is used for handling multiple messages which may be processed offline. This is useful for handling offline requests from third party systems. The mode used is obviously BATCH 2) The second way that we use OFS is Inter application processing. Inter application processing is used for calling one application from another. For example, this is used in user definable routines such as version routines. The mode used is GLOBUS 3) The last way in which we can use OFS is online. Online processing signifies that is a request is sent, processed immediately and the response sent back to the requestor There are two types or modes of online processing:

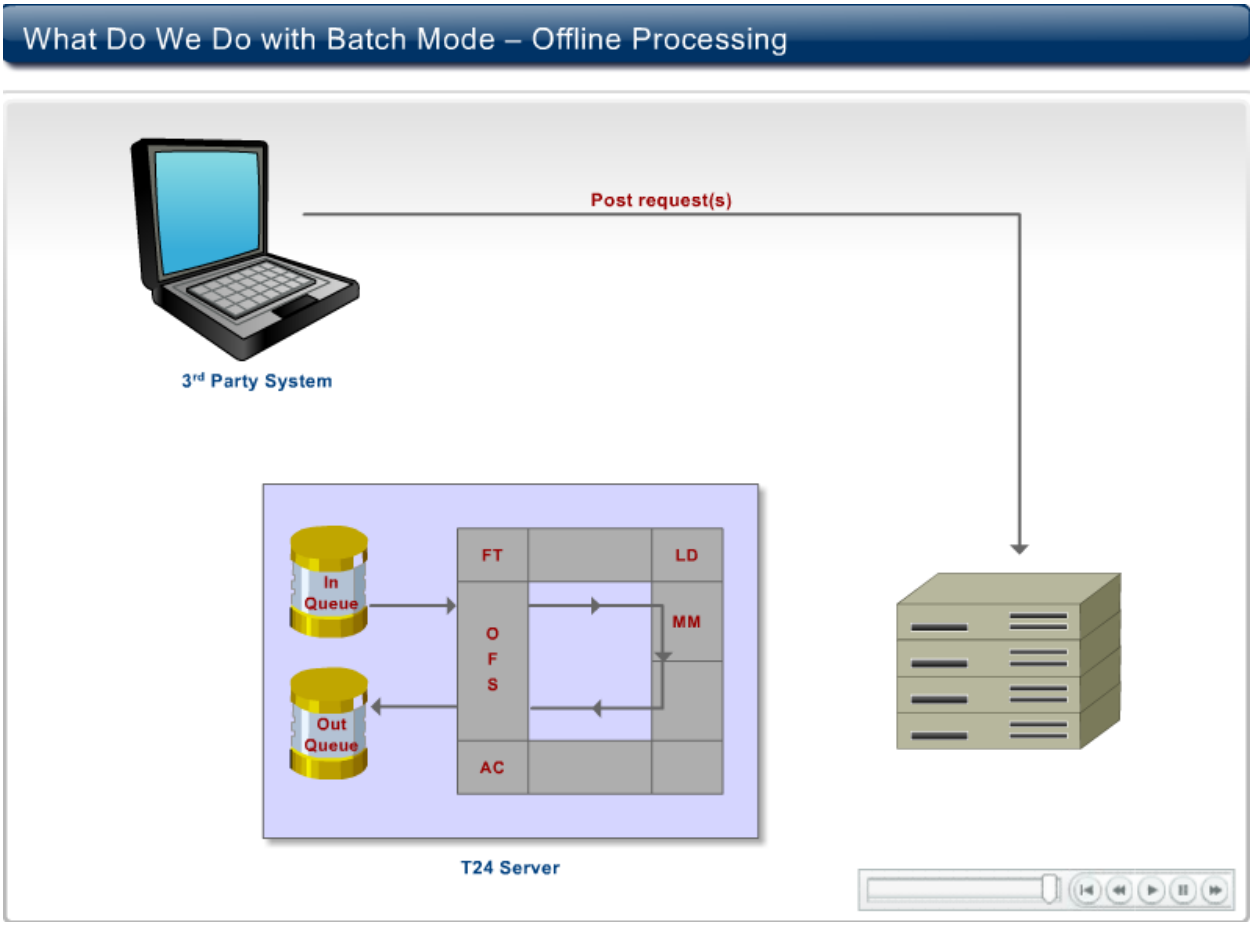
   a) An external system or user connects directly to OFS or to OFS through TCS. The mode used in this case is TELNET and b) A user connects to the T24 system using Browser. In this case a special mode called SESSION is used



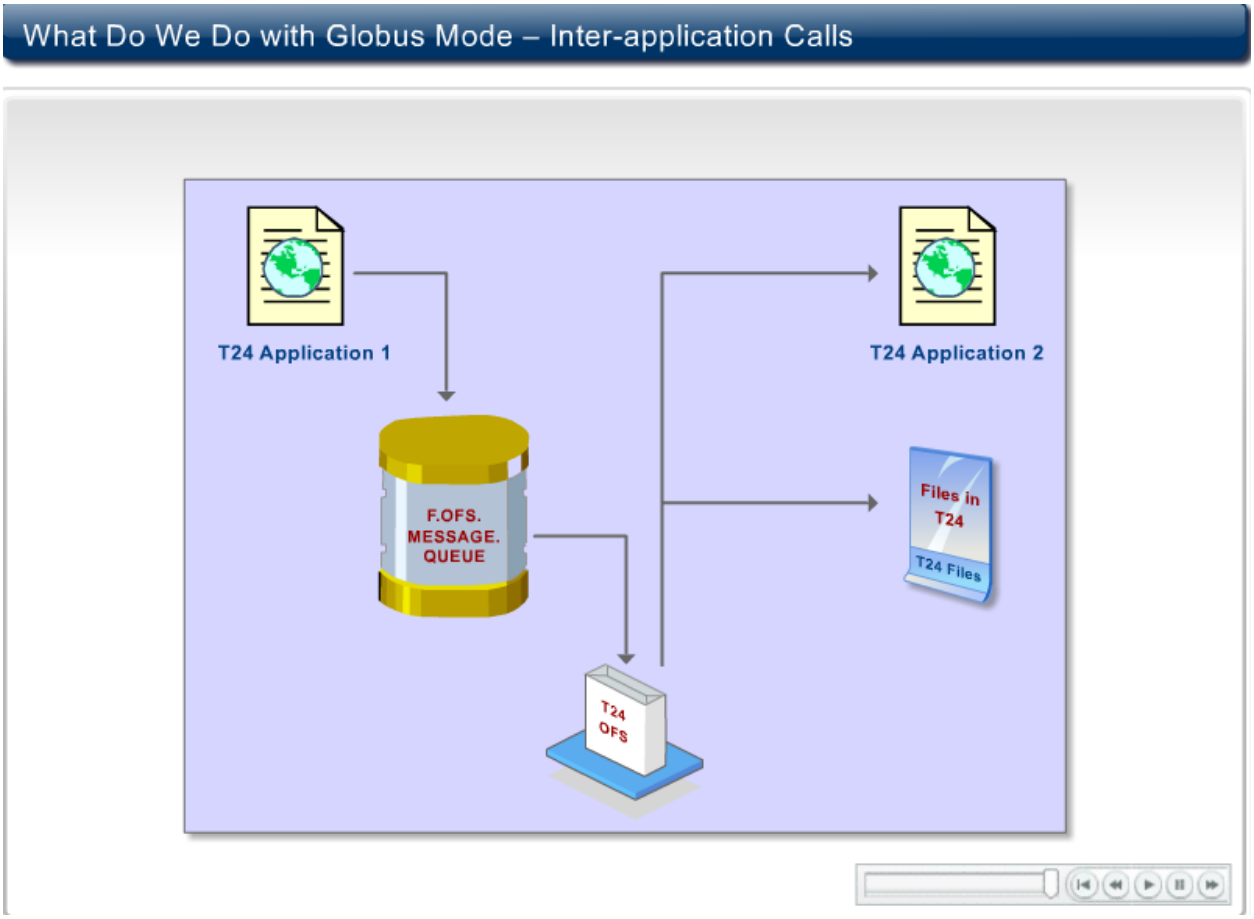
OFS batch, as the name suggests, deals with batches of messages. These messages are processed offline.

Third Party Systems post messages into a designated directory on the T24 server termed as the IN queue. Take a scenario, where a bank T24 as its main banking software but uses another software application for processing its FX transactions. These transactions are posted to T24 in the form of OFS messages before running COB. Messages can be sent in batches where a single file contains multiple messages or each message can be sent in a separate file. These files are placed in the IN Queue.

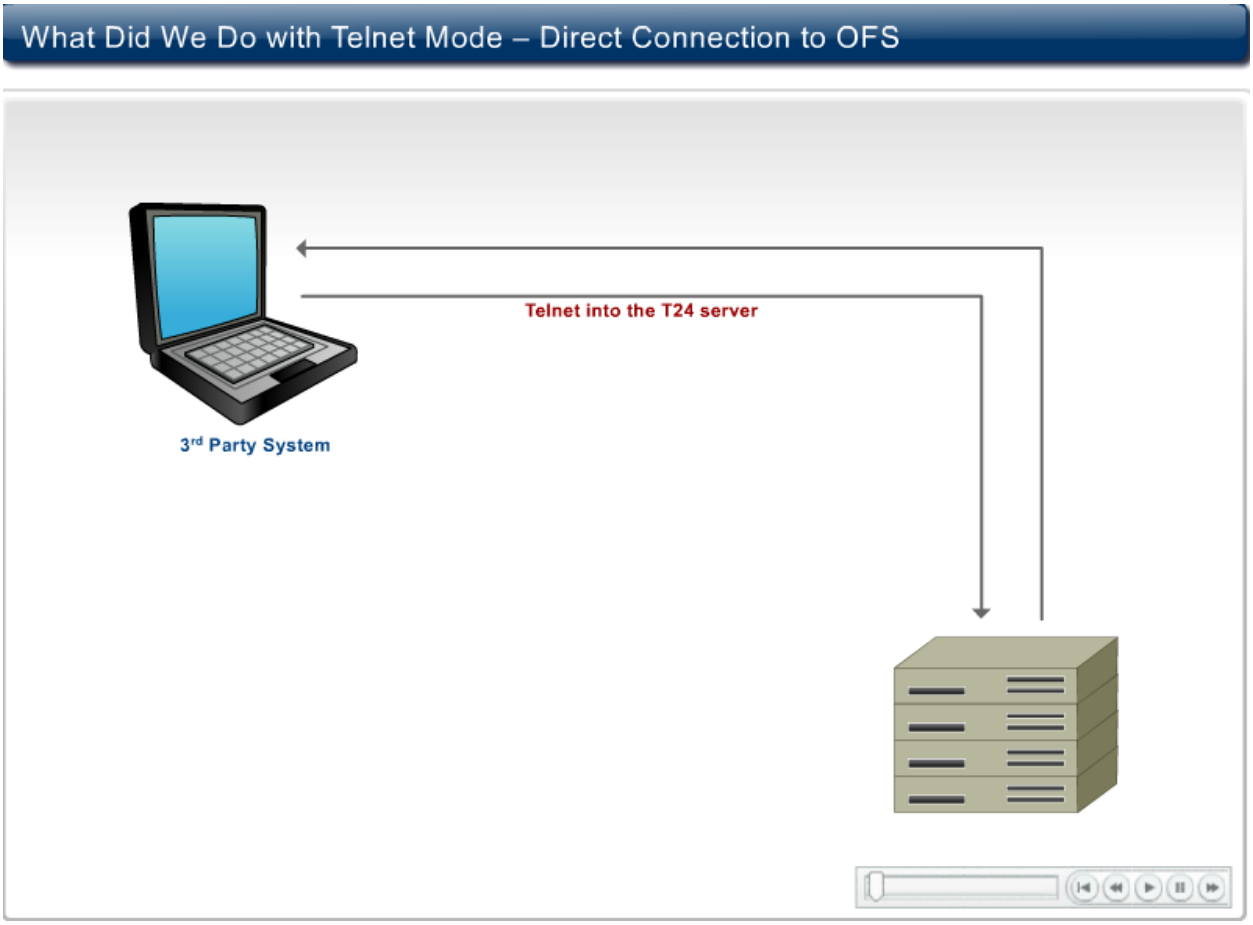
OFS picks up the request from the IN queue and processes it. OFS then places the response into a designated directory on the T24 server, termed as the OUT queue. It is then the responsibility of the third party system to pick up the response from the OUT queue.



Imagine a situation where in, when a record in the CUSTOMER application/version is committed, you wish to update some other application in T24 (such as a locally developed application) or a T24 file. For this purpose, when a record is committed, an appropriate OFS request to update the relevant T24 applications and files is formed and placed in a queue. OFS picks up the request from the queue, processes it and updates the relevant applications and files. Note that the update that you are making through OFS is totally independent of the update that is happening to the main application.

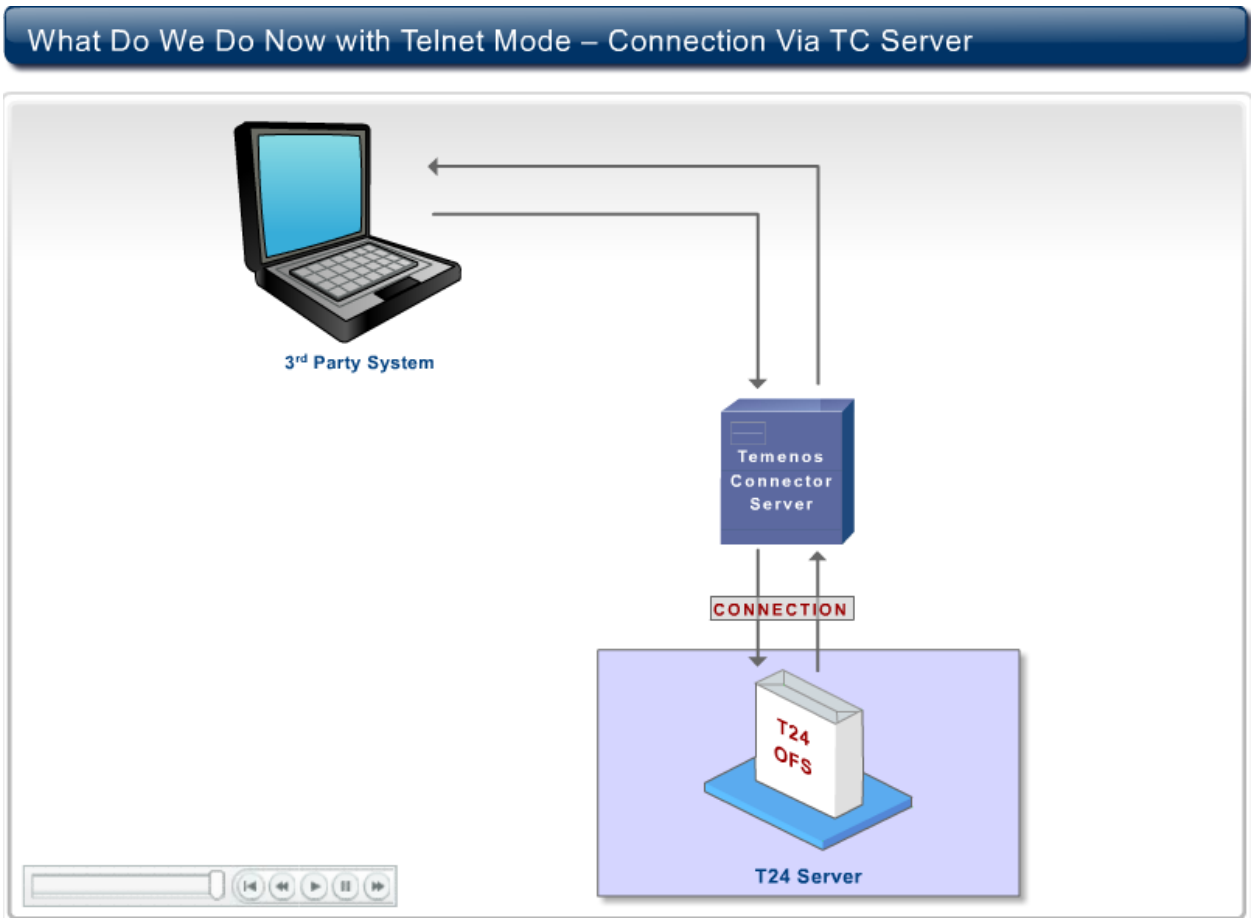


The Telnet mode was used for setting up a direct connection to T24, usually from a third party system, via a live TELNET link / connection. It used an Operating System level user id and password to gain Telnet access to the T24 server. Once the connection was established, the third party system would send an OFS request via the open connection and wait for the response. This way of using the Telnet mode has become obsolete with the advent of TC Server.



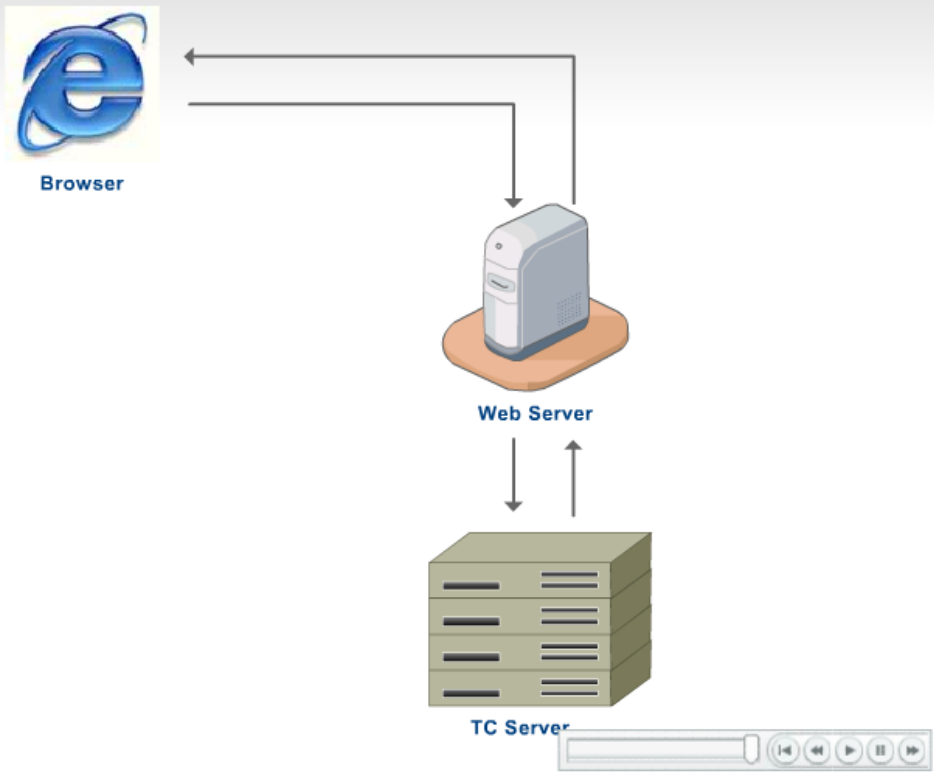
Message from third party systems could be sent to T24 via the Temenos Connector Server (TCS). These messages are received by TCS components called listeners which listen at designated ports. For instance, the bank could have developed its own internet banking software. All requests from this software would need to posted back to be the T24 application server in the form of an OFS message. A second common scenario could be where the bank needs an ATM interface. These systems would post a message to a listener running in TCS. TCS opens a connection into T24 when a message is received and passes the message to the OFS module. The OFS module processes the message and sends back the response to TCS. There are a couple of differences if you compare this with a direct Telnet connection:

a) TCS can use different protocols such as tcp, ssl and ftp. Note that TCS supports other types of listeners such as file, batch and ftp listeners also. b) Secondly, the connections that are opened into T24 are managed by TCS. That is, TCS does activities like spawning additional connections, killing unused connections etc.



The session mode of communication is used, when T24 requests come from the Temenos Browser client. A User sends a request to the webserver through a recommended browser (i.e. – Internet Explorer or Firefox). The webserver (or actually the browser servlet) forwards the request to the Temenos Connector Server. The TC Server in turn passes the request to T24 Server, where it is processed by the OFS module. The response is then sent back to the user via OFS and the Web Server.

What Do We Do with Session Mode – Browser Connection



Important Points

Let us summarise what you have seen so far:

- OFS is a module within T24
- It is the only gateway into T24
- It is message based
- OFS syntax is native to T24
- There are four OFS modes

Batch mode

What is OFS.SOURCE used for?

OFS.SOURCE contains the parameters for setting up an OFS connection into T24.

Usually different OFS source records are used for each connection.

You can see a screen shot of an OFS.SOURCE record with some suggested sample data. Let us now see some of the important fields.

ID: This field contains the name or Id of the OFS Source record. The user can set an appropriate name that can contain a maximum of 20 alphanumeric characters.

SOURCE.TYPE: This field defines the type of communication to be used. This is a mandatory field.

Remember that OFS allows the following modes of communication:

- A BATCH mode where one or more messages may be sent for offline processing through a queue
- Single messages may be passed to T24 for online processing through a TELNET connection
- T24 Applications can internally use OFS to perform updates across application/files using the GLOBUS mode
- Browser uses a specific mode called SESSION to 'talk' to OFS

IN.QUEUE.DIR:

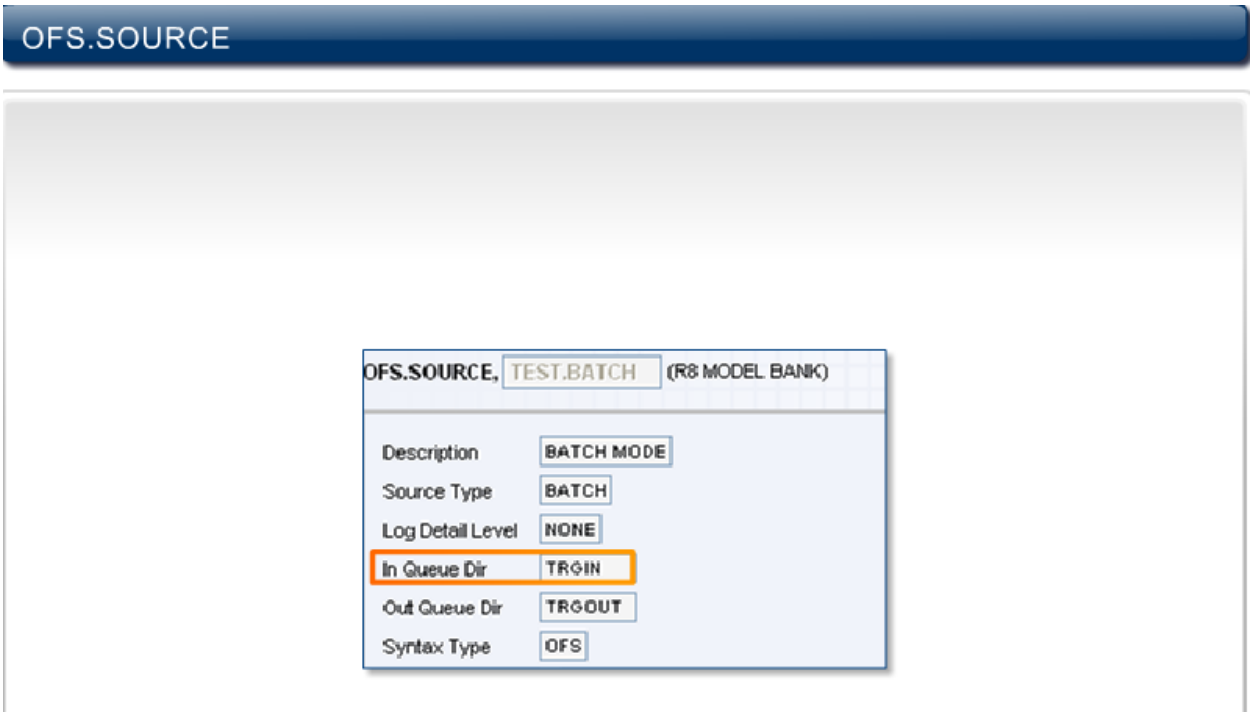
A queue may be a directory itself with all the files treated as holding message for processing. In case where messages are arriving from multiple sources, each source can use a distinct file to store its messages. In this case, the file itself becomes the queue.

How do we specify the queue?

IN.QUEUE.DIR defines the name of the directory used to hold incoming batches of messages when operating in a BATCH mode. A Batch OFS service can operate using either a specific record within the directory specified here, defined in IN.QUEUE.NAME (see next field), or can process any file within the directory if nothing is specified. Each file may contain one or more records in specified OFS message format. Processed records are written to the directory specified in the OUT.QUEUE.DIR either to the specified OUT.QUEUE.NAME or the same name as the inward record. In Queue Dir may refer to an existing directory or if not, a new directory will be created (under the .run directory) on authorization of the record. This field is mandatory, if SOURCE.TYPE is BATCH.

IN.QUEUE.NAME:

This field holds the name of a specific input file present in the input directory (specified by IN.QUEUE.DIR) to be processed when operating in a BATCH mode. If not specified, OFS will pick up any messages falling into the directory defined in the IN.QUEUE.DIR.



OUT.QUEUE.DIR

This defines the directory used to store output files, which contain details of completed transactions together with any validation errors or overrides. This either refers to an existing directory or a new directory will be created (under the .run directory) at authorization of the record. This field is mandatory, if SOURCE.TYPE is BATCH.

OUT.QUEUE.NAME

This field contains the name of a specific output file resident in the output directory that is used to hold outgoing batches of messages when operating in a BATCH mode. If not specified, OFS will write output messages into the directory defined in the OUT.QUEUE.DIR.

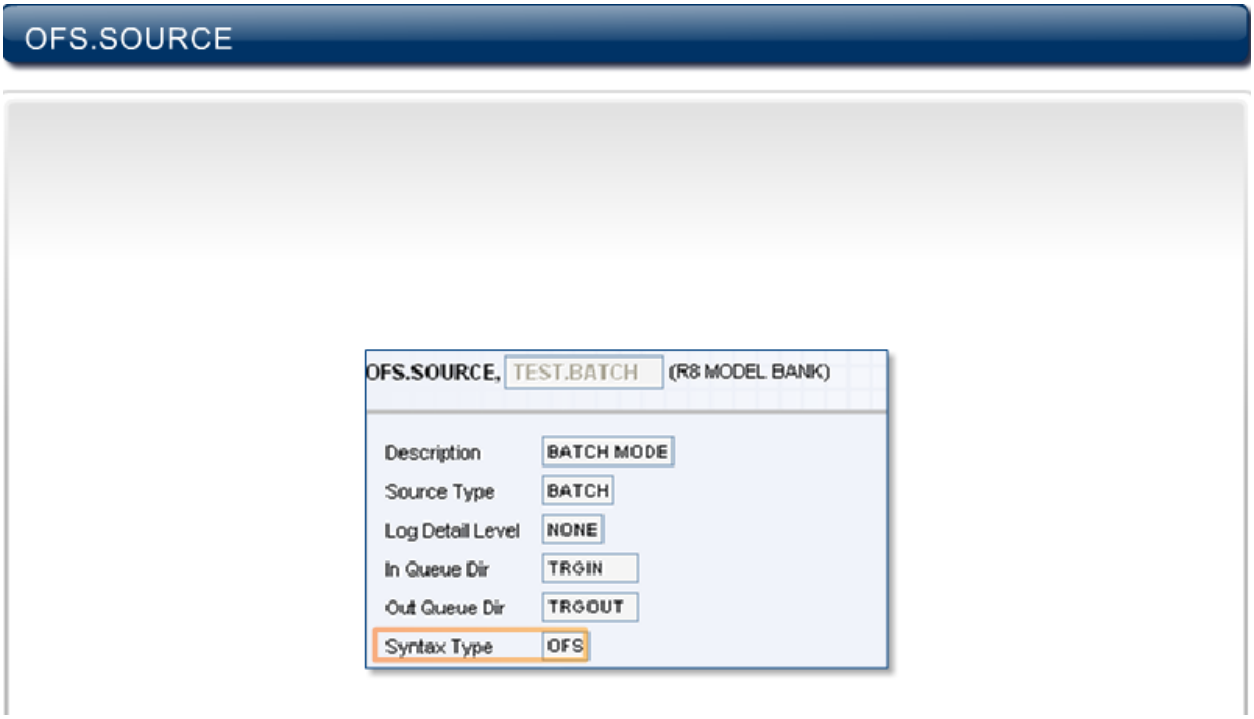
SYNTAX.TYPE

This field is used to indicate the syntax type of the messages. The allowed values are:

GTS - Request and response follows the old style GTS syntax. This syntax is deprecated. OFS supports this syntax to be downward compatible with legacy code.

OFS - Request and response follows the standard OFS syntax. This is the standard syntax.

XML - Request and response are in XML format. This is specifically used for browser connections.

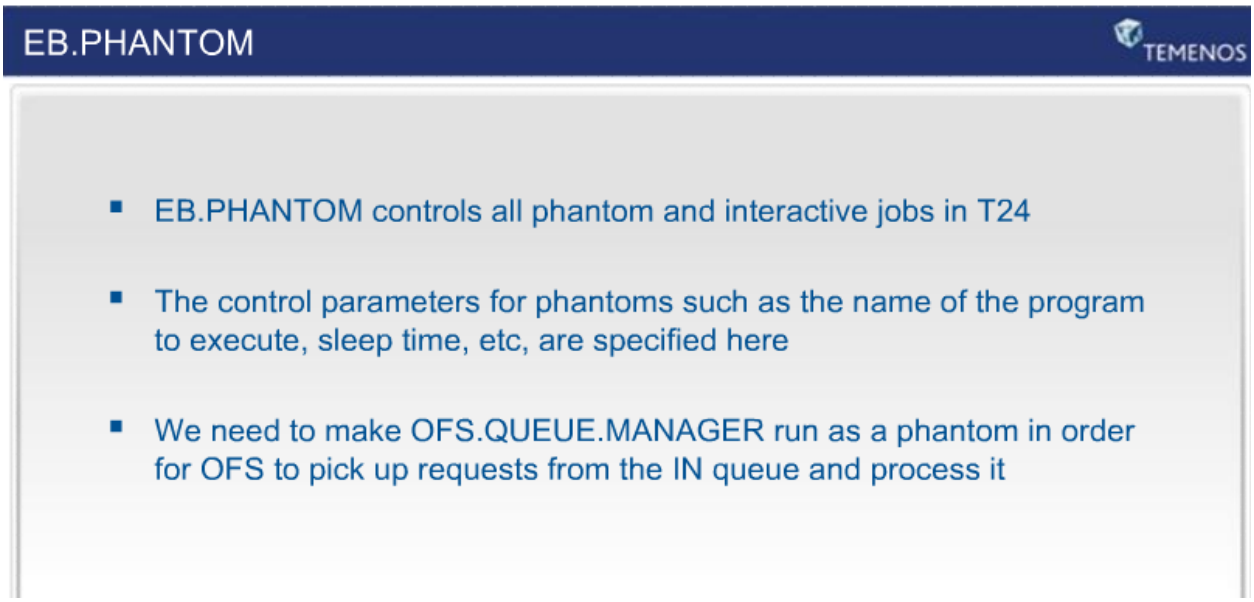


You will now configure EB.PHANTOM. What does EB.PHANTOM do?

Well, EB.PHANTOM controls all phantom and interactive jobs in T24.

The control parameters for phantoms such as the name of the program to execute, sleep time, etc, are specified here. Why do you need it?

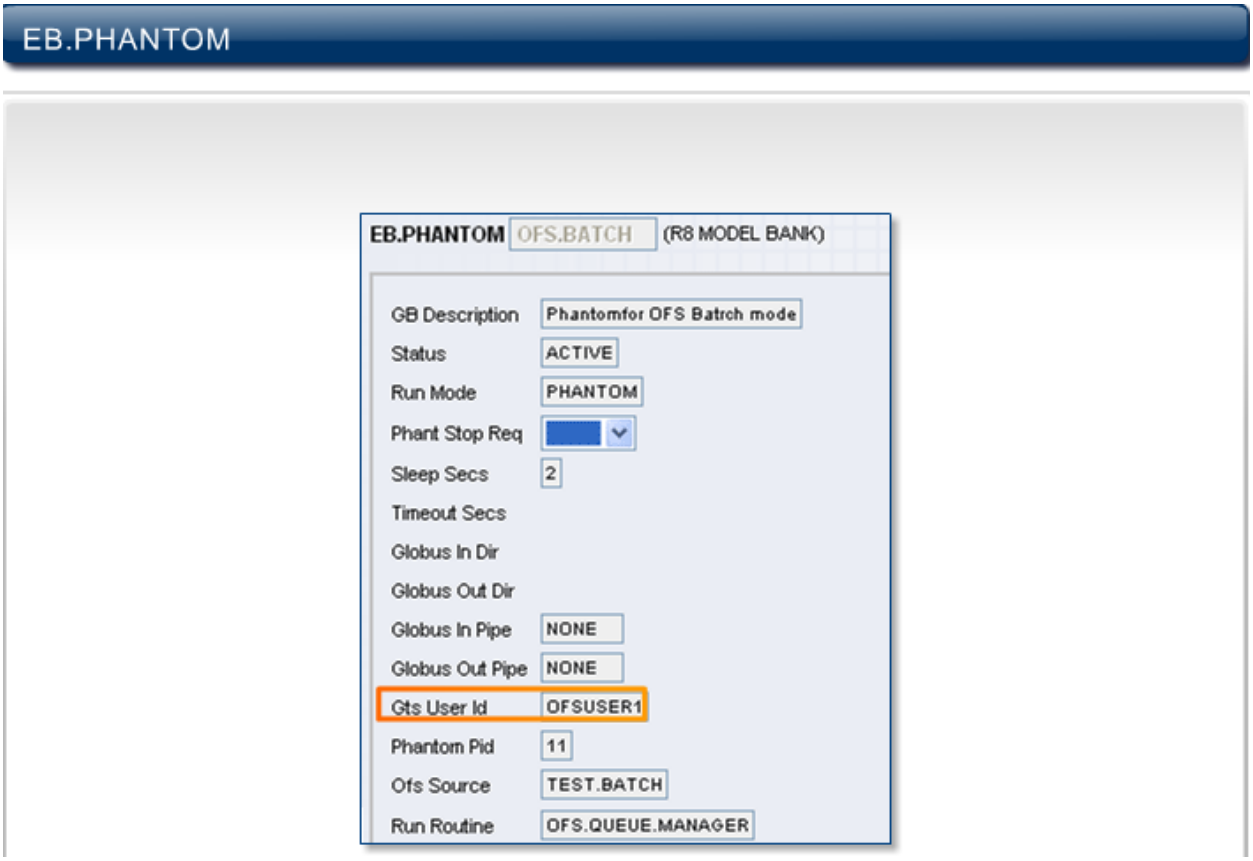
OFS.QUEUE.MANAGER run as a phantom process that picks up requests from the IN queue. Therefore you need a corresponding EB.PHANTOM record to configure as well as control OFS.QUEUE.MANAGER.



Let us create a new EB.PHANTOM record. The important fields are given below:

Id: The id can be any alpha numeric text.

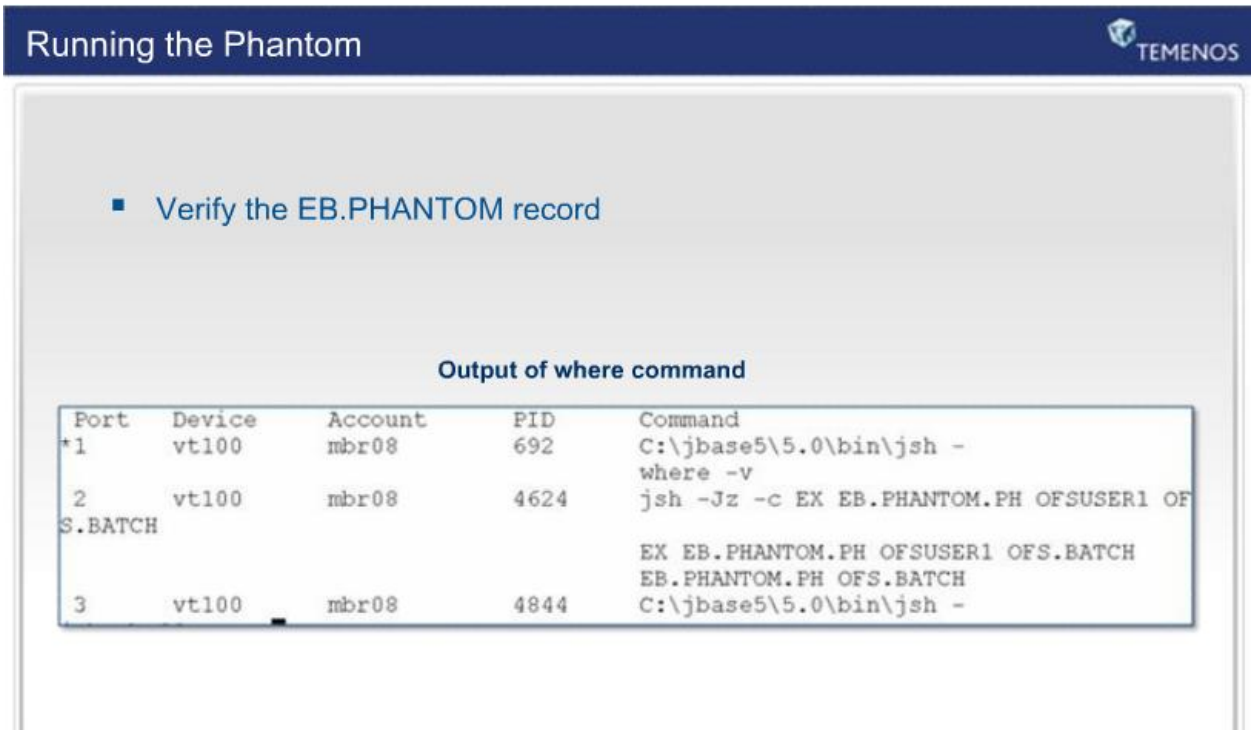
Status: This is a Read-only field which is populated at runtime by T24. Possible values are Active or Closed. Active denotes that the process is running. A status of Closed denotes that the process is inactive.



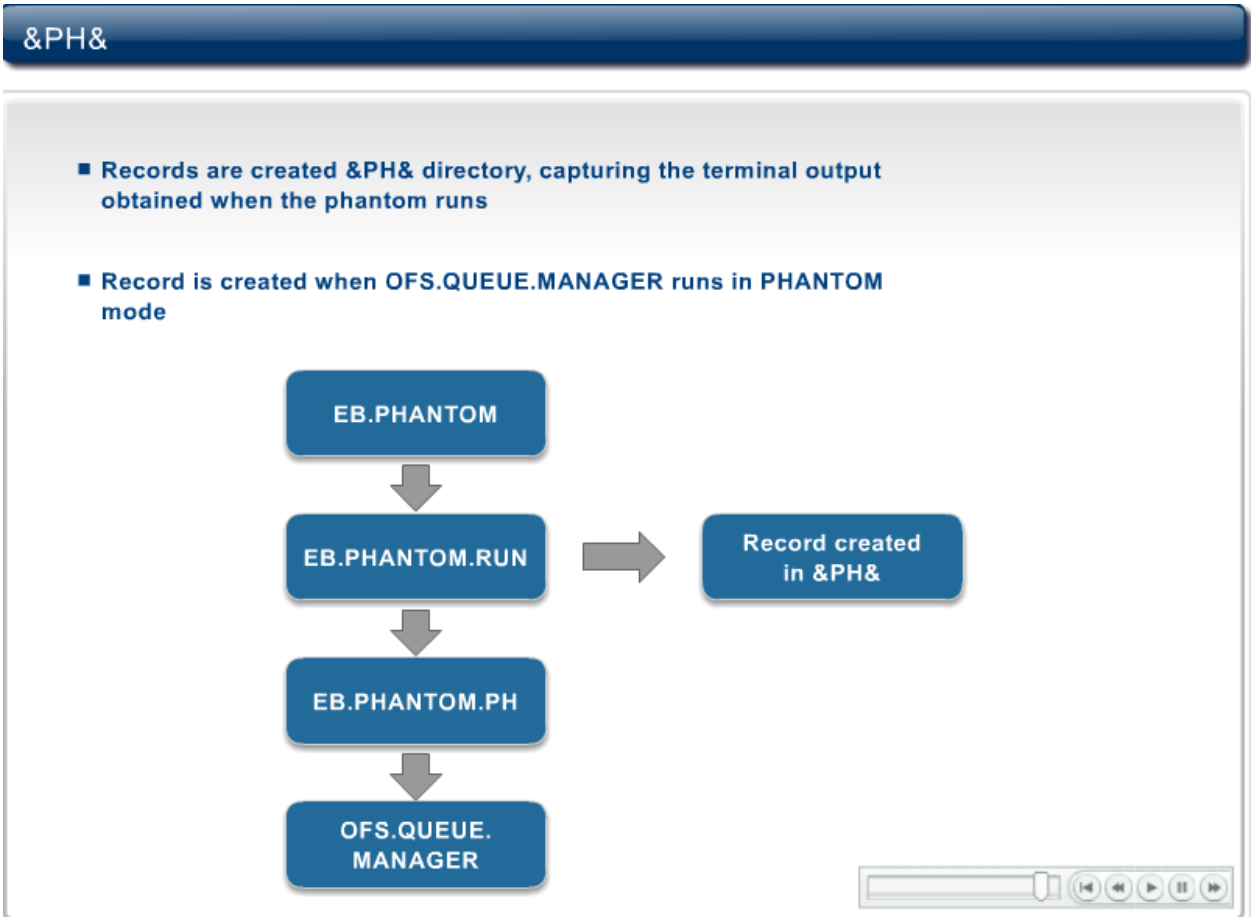
Verify the EB.PHANTOM record to run the OFS.QUEUE.MANAGER as a phantom.

You may check for the phantom by giving a where –v command from the jshell.

Note that this does not display the name of the phantom process (i.e. OFS.QUEUE.MANAGER), but rather the name of the parent process (i.e. EB.PHANTOM.PH) followed by the EB.PHANTOM record id.



When OFS.QUEUE.MANAGER is started as a Phantom (in PHANTOM mode), by verifying EB.PHANTOM record, it updates &PH&. EB.PHANTOM.RUN updates &PH& file when a phantom is started in Phantom mode.



The key to &PH& starts with the program that is executed in the background and in EB.PHANTOM's case, EX is executed in the background. So all the keys to &PH& will start with EX, in the case of EB.PHANTOM phantom processes. The record is created ONLY IF the phantom is started in the phantom mode.

Record Key - EX_ddddd_ sssss _u

EX - Program executed in the background. For EB.PHANTOM, it is EX.

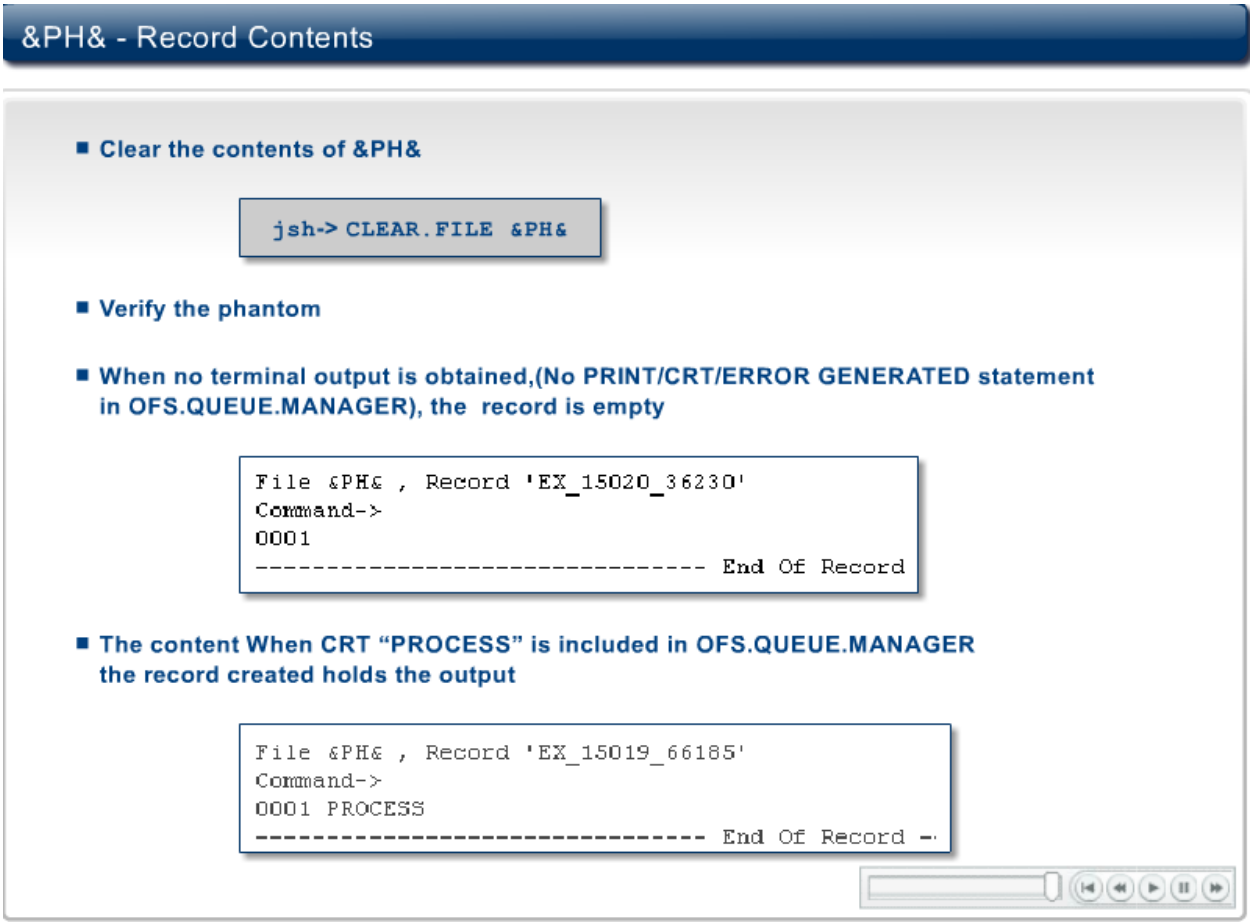
ddddd - Internal Date

sssss - Internal time

u - Unique Number

Phantom mode - The record in &PH& is empty if the phantom (OFS.QUEUE.MANAGER) does not produce any output on the screen, i.e., there are no PRINT or CRT statements. If it contains any terminal output is obtained, they are captured in this record.

In interactive mode, the contents of the record appears to the user on the screen. There are no records created in &PH&.



Let us use this sample enquiry to test the OFS batch mode:

ENQUIRY.SELECT,,INPUTT/123456,%ACCOUNT

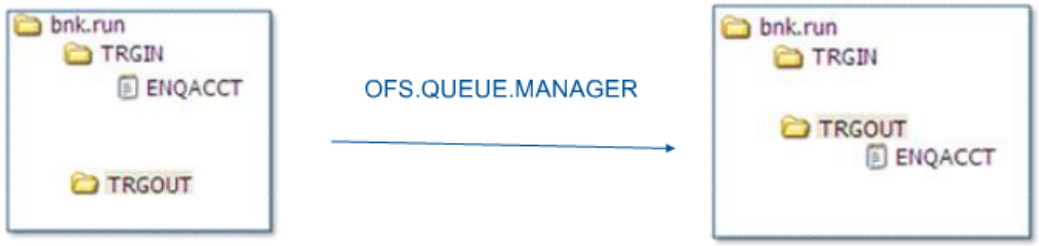
Create a record in your input queue (i.e.TRGIN) say with an id of ENQACCT and enter the sample enquiry within it. To do this, execute a JED TRGIN ENQACCT from the jshell, type the enquiry in the first line and save by pressing escape F I

The OFS message, i.e. your enquiry, will be picked up by the OFS.QUEUE.MANAGER and dispatched for processed. Once processed, the request will be removed from the IN queue and the response written to the OUT queue into a record with the same ID.

Testing the Batch Mode

TEMENOS

- Sample enquiry
  ENQUIRY.SELECT,,INPUTT/123456,%ACCOUNT
- Create a record in your input queue (i.e. within TRGIN with an ID of say ENQACCT)
- Enter the your enquiry into this record
- Once processed, the request will be removed from the IN queue
- The response is written to the OUT queue into a record with the same ID



You could type the command CT TRGOUT ENQACCT to view the output. The output should look like the one shown.

Check the Output

TEMENOS

- Does your output look like this?

Output of CT TRGOUT ENQACCT

```
    ENQACCT
001 %ACCOUNT//1,,@ID::@ID/MNEMONIC::MNEMONIC/ACCOUNT.OFFICER::ACCOUNT.OFFICER/
    CATEGORY::CATEGORY/CURRENCY::CURRENCY/ONLINE.ACTUAL.BAL::ONLINE.ACTUAL.BAL
    /LIMIT.REF::LIMIT.REF/POSTING.RESTRICT::POSTING.RESTRICT/INT.NO.BOOKING::I
    NT.NO.BOOKING/CONDITION.GROUP::CONDITION.GROUP/CATEGORY::CATEGORY,"
      10057"."TRAUD2    "."Client Of|"."Current Account"."AUD"."
         "."          "."  "."        "."   1"."  1001","         22117"."TRAU
    D3    "."Client Of|"."Current Account"."AUD"."                    "."
        "."  "."        "."   1"."  1001","         22179"."TRAUD4    "."Clien
    t Of|"."Current Account"."AUD"."                    "."          "."  "."
         "."   1"."  1001","         10014"."TRAUD1    "."Retail Ba|"."Curren
    t Account"."AUD"."             14.79"."          "."  "."        "."   1"
```

A sample transaction request is given. Enter this into a record NEWACCOUNT in TRGIN.

Note:

All the mandatory fields of the application have to be filled. The ID will be automatically generated by the system.

One More Test - A Transaction Request

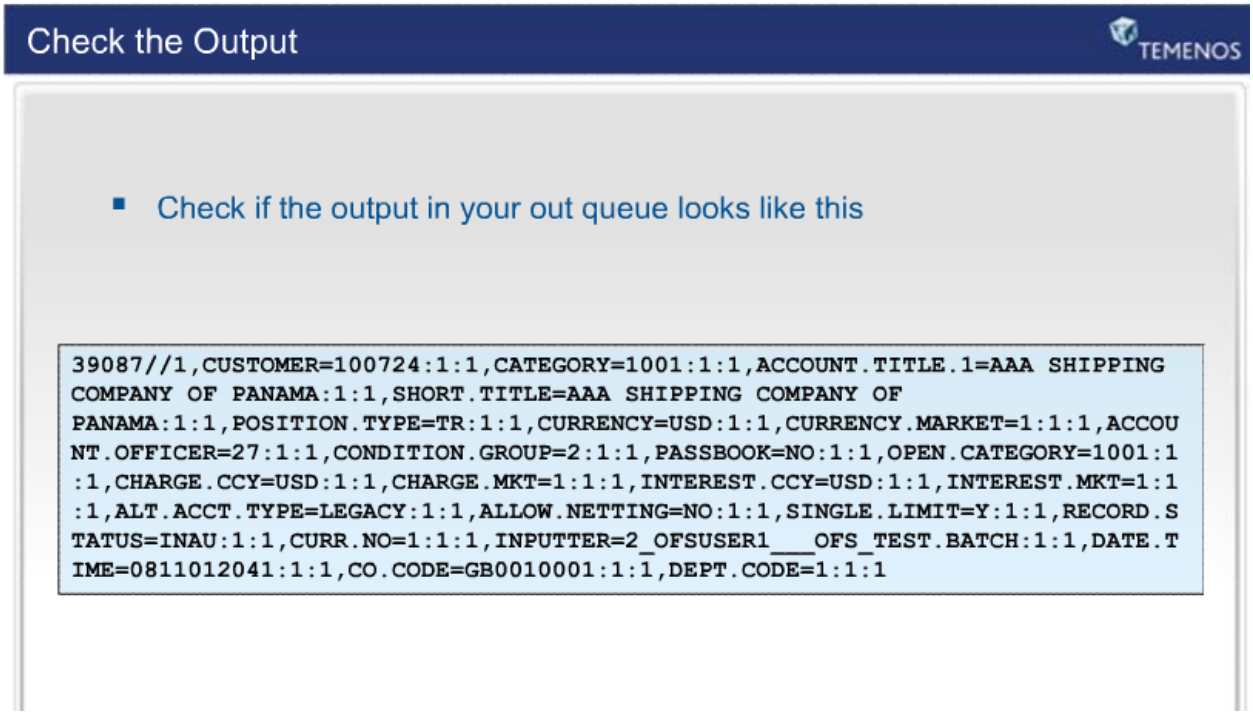
TEMENOS

```
ACCOUNT, /I/PROCESS, INPUTT/123123,,CUSTOMER=100724,CATEGORY=1001,
CURRENCY=USD
```

Note:

- All the mandatory fields of the application have to be filled
- The id will be automatically generated by the system

Execute a command CT TRGOUT NEWACCOUNT from the jshell. Check if your output looks like the one shown.

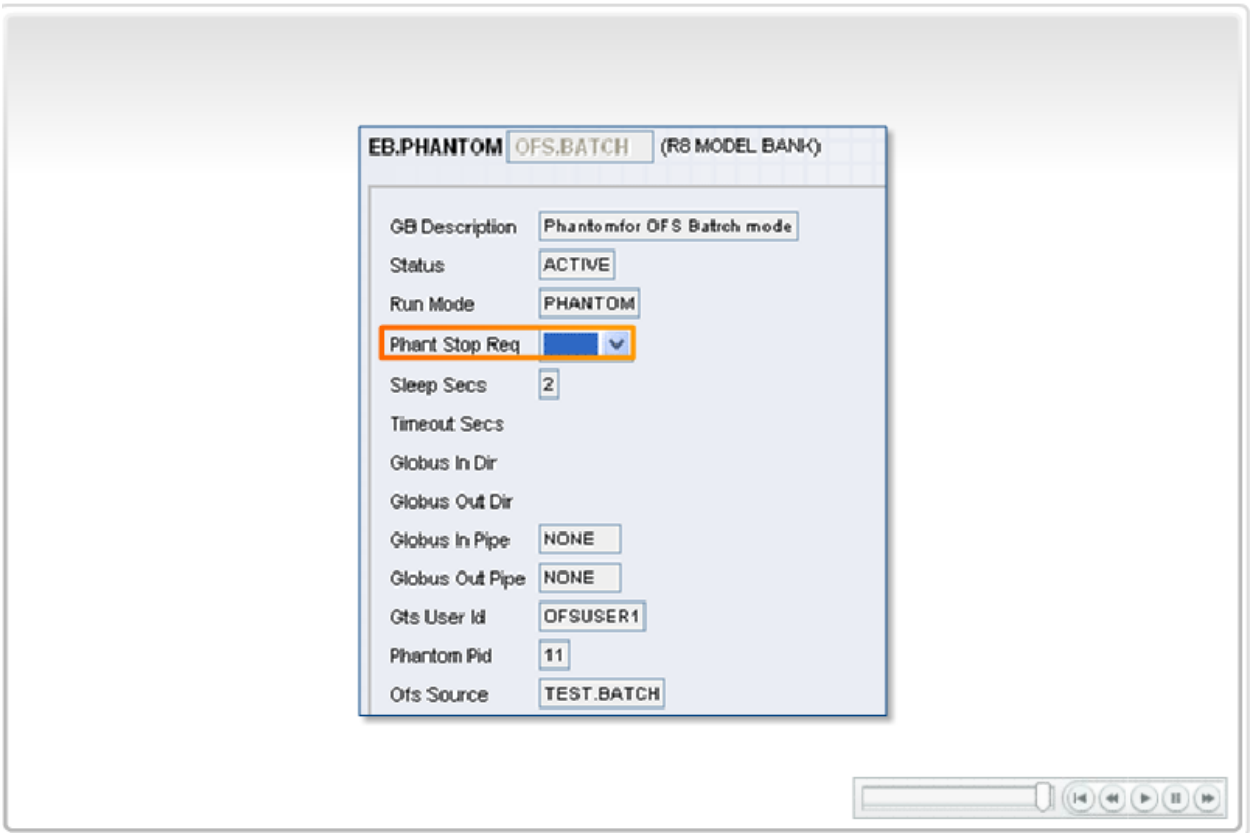


Two more fields become visible when you verify the EB.PHANTOM record. They are the Status and Phantom Stop Req fields. The Status field obviously displays the status and the Phantom Stop Req field is used to stop the phantom.

Status: This is a read-only field which is populated at runtime by T24. Possible values are Active or Closed. Active denotes that the process is running. A status of Closed denotes that the process is inactive.

Phant Stop Req: Enter STOP here to stop the phantom.

Controlling the Phantom



You can now try to write a OFS message to input a SECTOR record with id 1200; write an OFS message to input a new ABBREVIATION called UL for listing the USER records and write a OFS message to input a CUSTOMER record in T24. Use zero authorisers in your message.

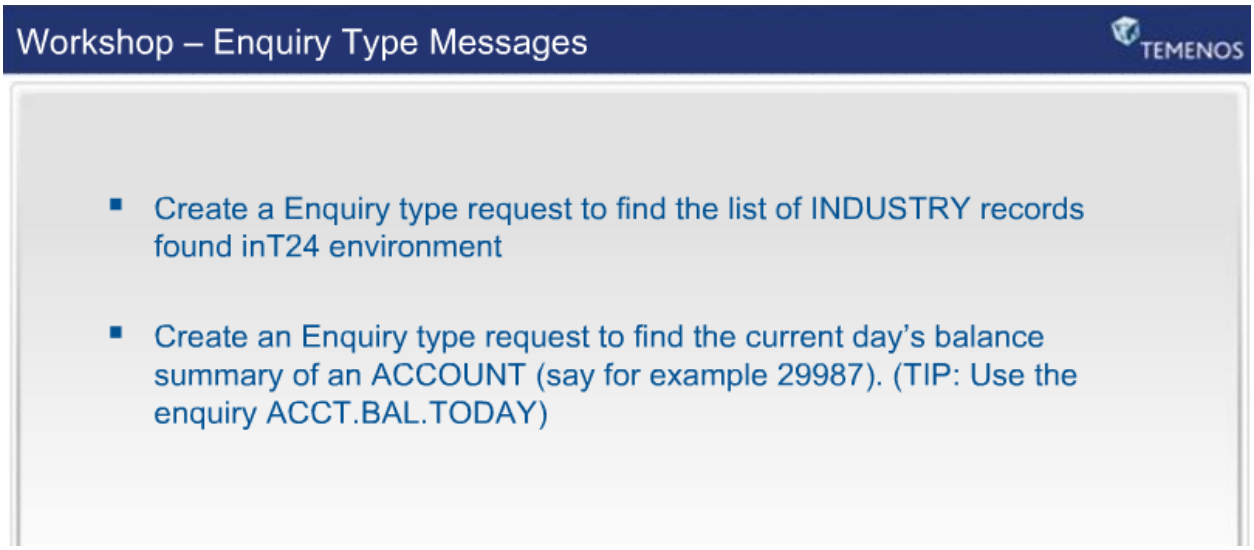
Workshop

TEMENOS

Test the following OFS messages you wrote in earlier workshop:

- Write an OFS message to input a SECTOR record with ID 1200
- Write an OFS message to input a new ABBREVIATION called UL for listing the USER records
- Write an OFS message to input a CUSTOMER record in T24. Use zero authorisers in your message

You can now try to create a enquiry type request to find the list of INDUSTRY records found inT24 environment and create an enquiry type request to find the current day's balance summary of an ACCOUNT (say for example 29987). (TIP: Use the enquiry ACCT.BAL.TODAY)



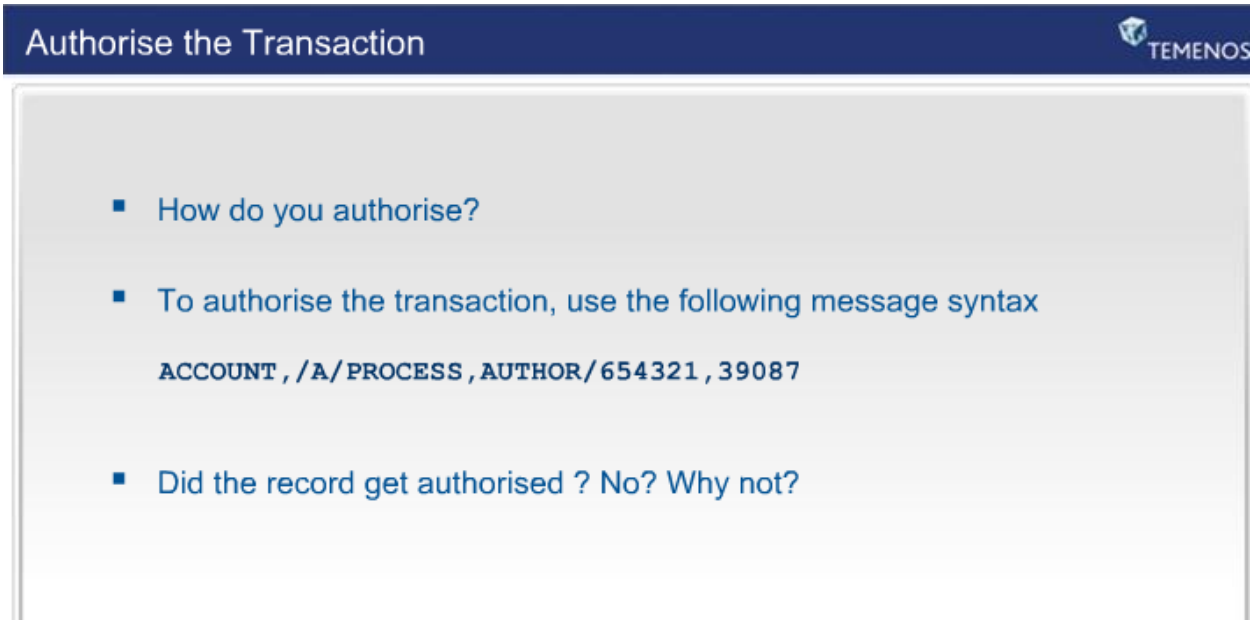
How do you authorise? Give another OFS message of course. But you need the record id. Get it from the response to your input.

39087//1,CUSTOMER=100724:1:1,CATEGORY=1001:1:1,ACCOUNT.TITLE.1=AAA SHIPPING COMPANY OF PANAMA:1:1,S

To authorise the transaction, use the following message syntax.

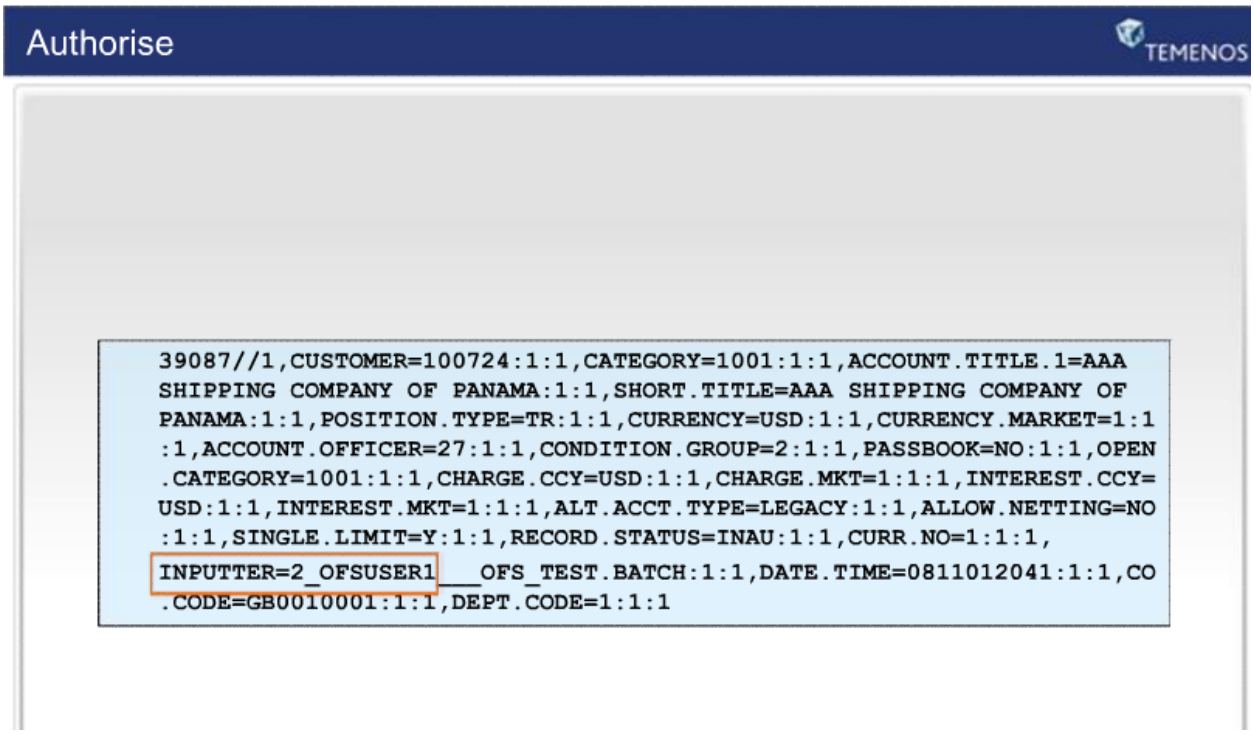
ACCOUNT,/A/PROCESS,AUTHOR/654321,39087

Did the record get authorised? No? Why not?



To find the reason, look at the response to the OFS message to input a record. A sample is shown here.

Look at the INPUTTER field. What is the user ID recorded there? Is this the user in your message? If not, who?

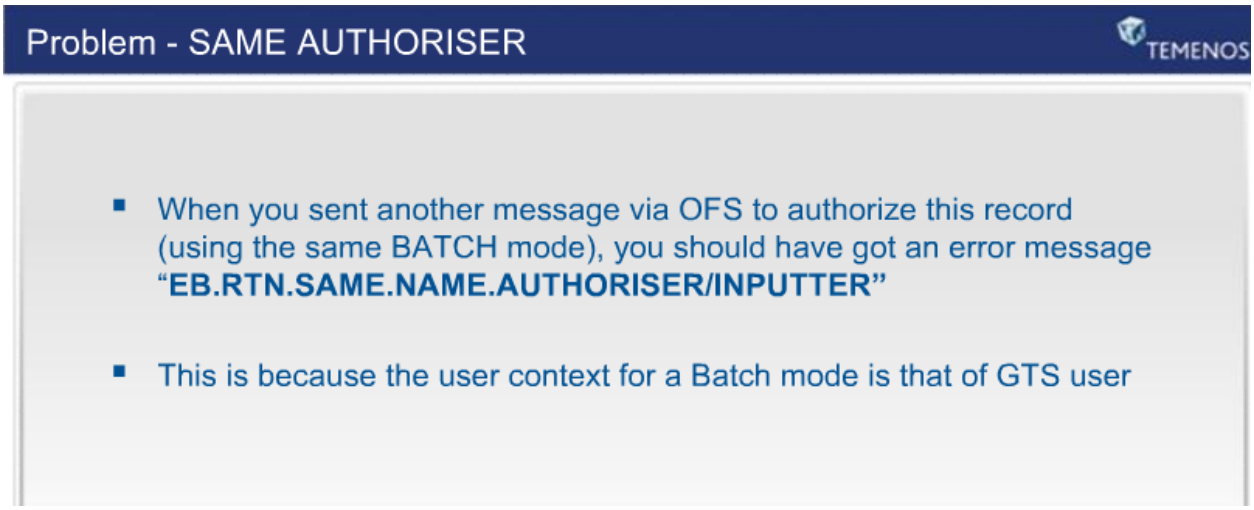


When you input a transaction using OFS (in any mode), based on the no of authorizers set in the version, the record will either get to INAU or live status.

In this case, the record is in INAU status and the record has been input using the BATCH mode of T24.

Now, if you send another message via OFS to authorize this record (using the same BATCH), you will get an error message “EB.RTN.SAME.NAME.AUTHORISER/INPUTTER”.

This is because the user context is that of GTS user if we run a Phantom as a Phantom. When you use the same batch record for inputting and authorising, the user context and therefore, the inputter and authoriser will be the same.

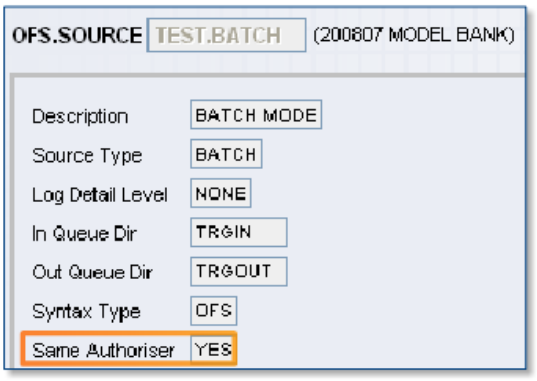


How do we solve this?

To overcome this, set the field SAME.AUTHORISER in OFS.SOURCE to YES.

When this field is set, the same phantom can be used for input as well as authorisation. i.e. we can use the same EB.PHANTOM record. Think about what you will do if you did not have the SAME.AUTHORISER field.

Solution - SAME AUTHORISER



You can now try to write an OFS message to authorise the SECTOR record with ID 1200 and write an OFS message to authorise the new ABBREVIATION called UL.

Workshop

TEMENOS

- Write an OFS message to authorise the SECTOR record with id 1200
- Write an OFS message to authorise the new ABBREVIATION called UL

This workshop simulates processing of batch OFS messages.

Workshop – Batch Messages

- Bank XYZ has it's own software for capturing new account details. Since it uses T24 as the backend, the new account details are sent in a group to T24 using OFS
- You will simulate a batch of messages to create 4 accounts. Use only a single batch file to store all 4 messages
- Make sure that you input one message with missing parameters
- Check what happens to the messages that come after this

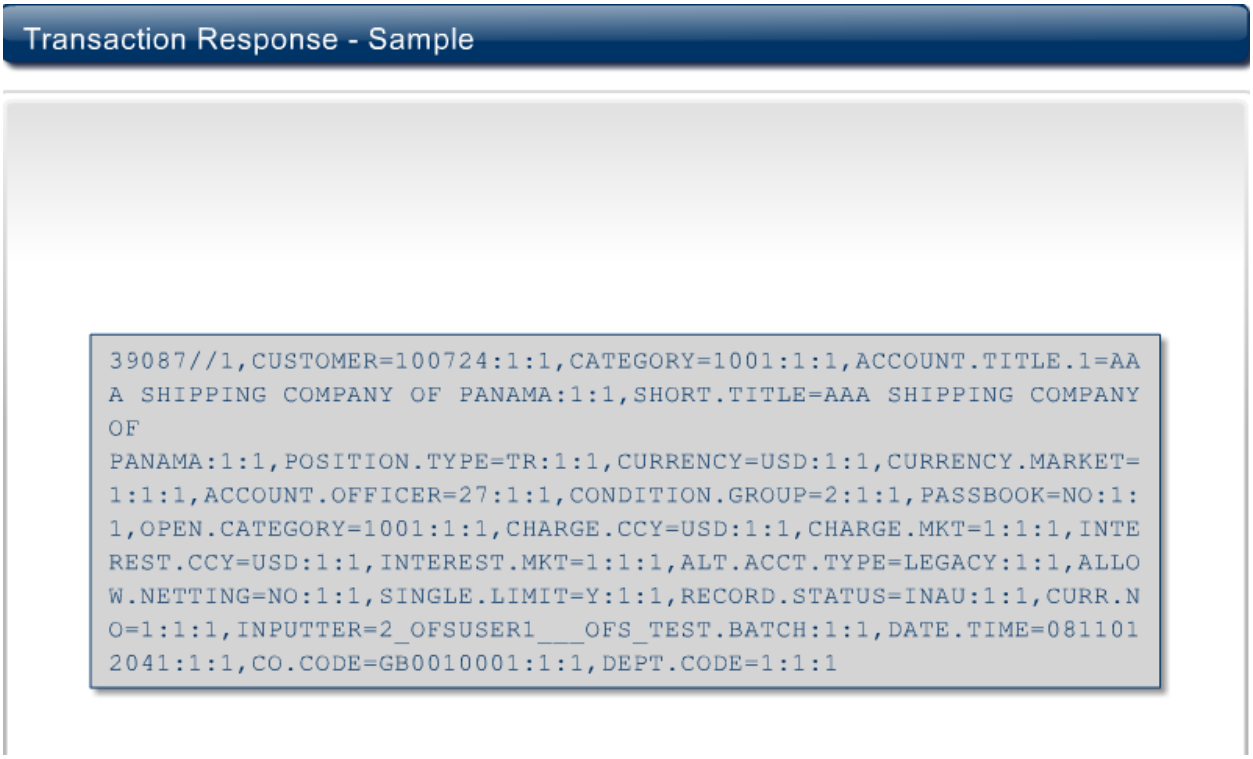
Important Points

- Batch mode allows us 'pump' messages in a queue
- A queue may be a folder or a particular file within a folder
- The important fields to setup in OFS.SOURCE are **Id,Source Type, In.Queue.Dir, Out.Queue.Dir and Syntax.Type**
- EB.PHANTOM must be configured to invoke OFS.QUEUE.MANAGER
- The important fields in EB.PHANTOM are **Status, Phant Stop Req, Run Mode, Sleep secs, Globus In Pipe, Globus Out Pipe, Gts User Id,Ofs Source and Run Routine**
- When the Run Mode is specified as Phantom, the background process runs under the SMS context of the user given in this field. This user name is updated in the audit fields of OFS based transactions
- The Same Authoriser field in OFS.SOURCE is used to overcome the issue of "same authoriser / inputter"

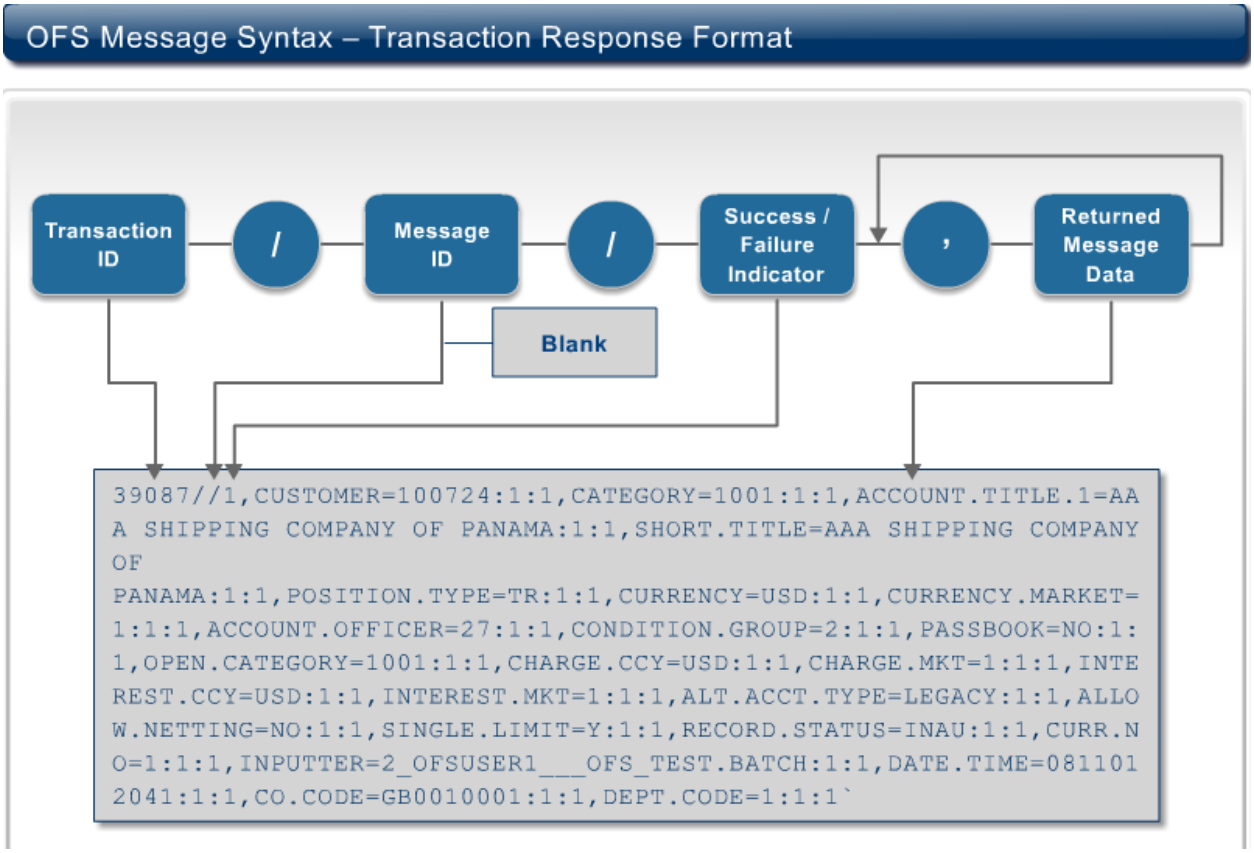
Response messages

You can see the - by now familiar – response to input an account.

It seems to be composed of essentially two sections, One which contains the id of the record (account) that was created and two some information about the account itself.



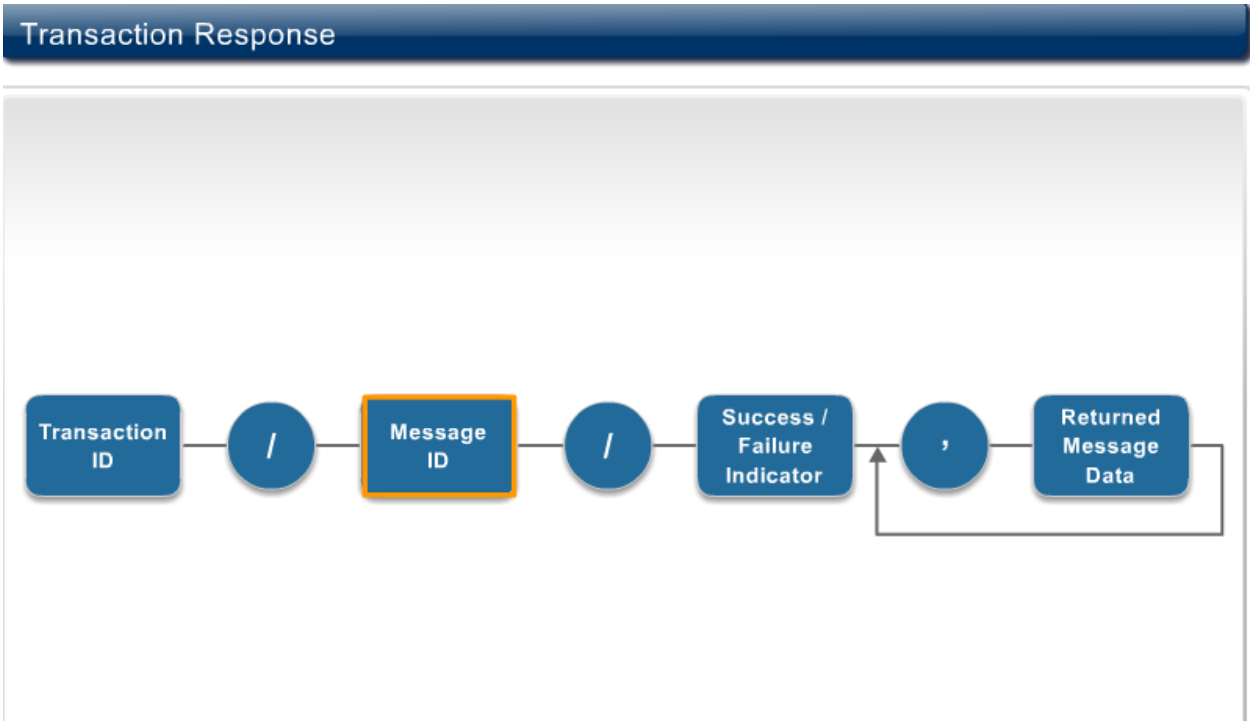
We can describe the syntax as above. You will now see each of the parts of the response in detail.



TRANSACTION ID :

The transaction ID contains either the value supplied for the transaction in the request or the value that is automatically generated by the T24 application (when no such value had been supplied in the request).

MESSAGE ID : The Message ID contains the value of the Message ID if it had been supplied in the request.



SUCCESS / FAILURE INDICATOR

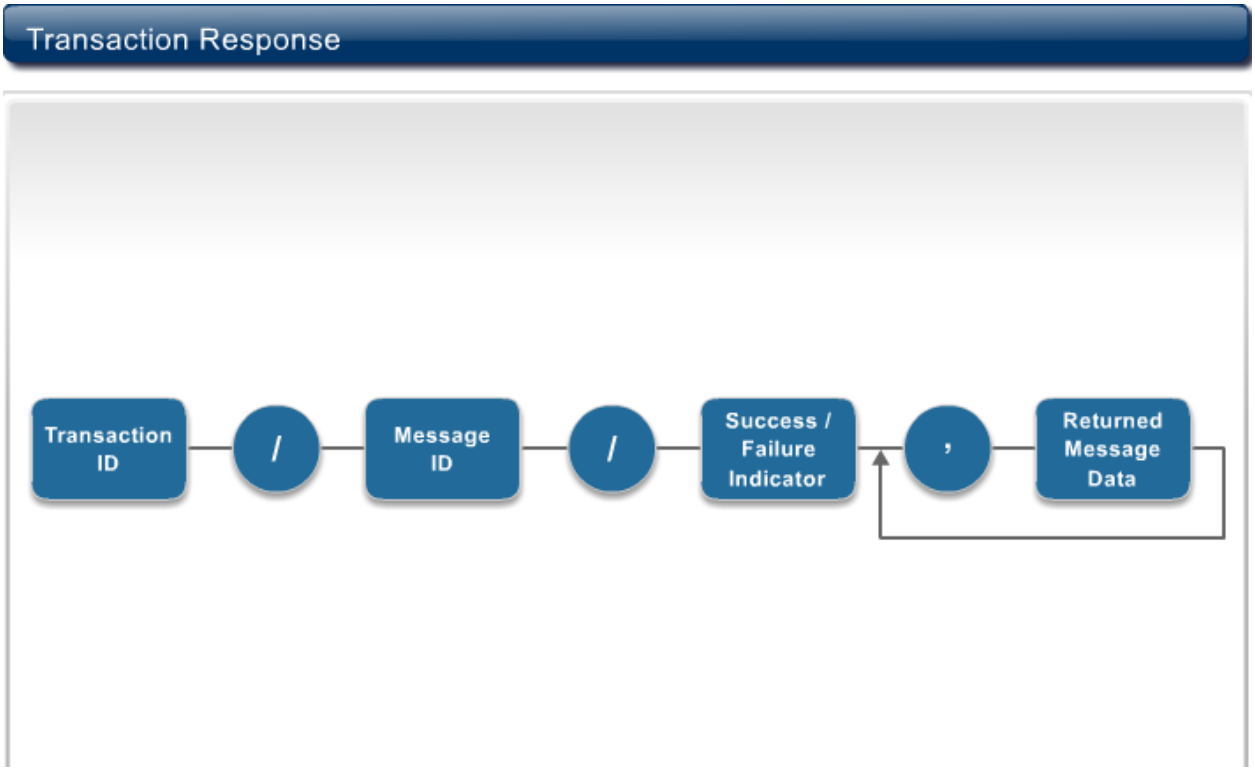
This part of the message indicates the status of the transaction request processed. OFS returns one of the following values:

1 Successful transaction

-1 Errors encountered during processing

-2 Override condition (s) encountered during processing

-3 T24 server is offline



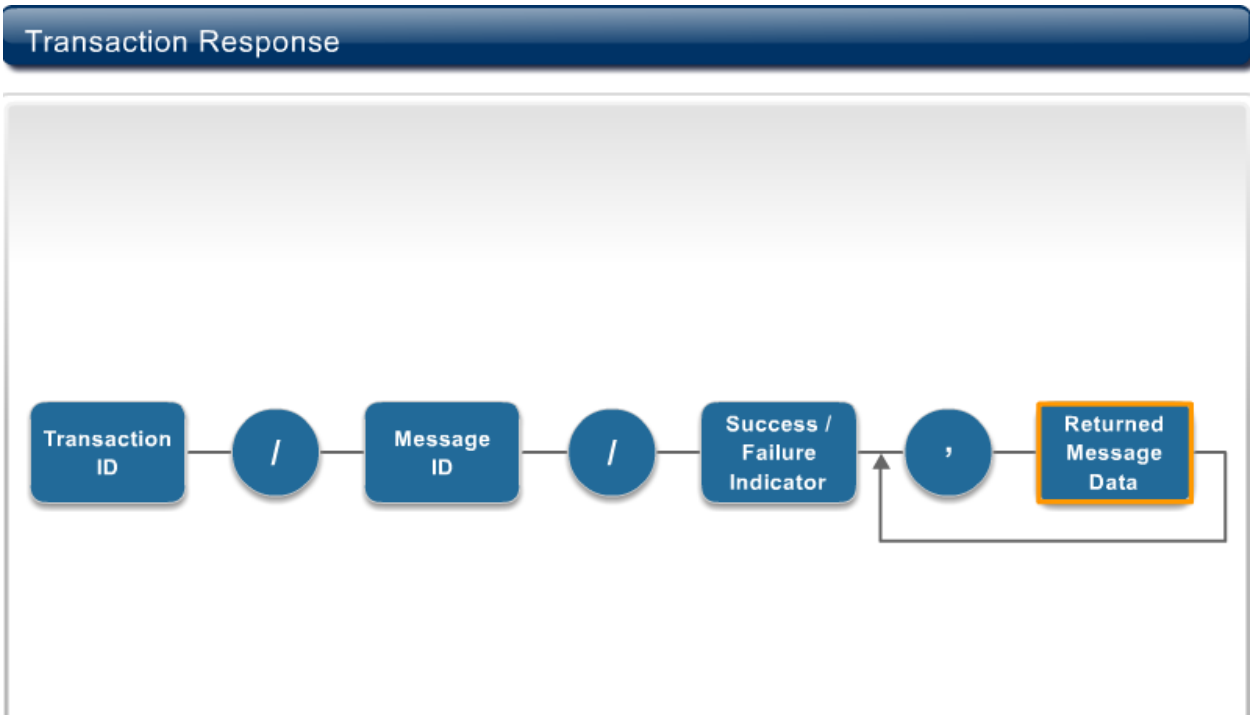
RETURNED MESSAGE DATA

A successfully processed message will contain all the fields populated in the transaction.

Field and value pairs are separated by commas.

The format is the same as that of the request message data.

However for an unsuccessful transaction, the returned message data has a slightly different format.



The structure for the Returned Message Data portion of an OFS transaction response is similar to that of the request.

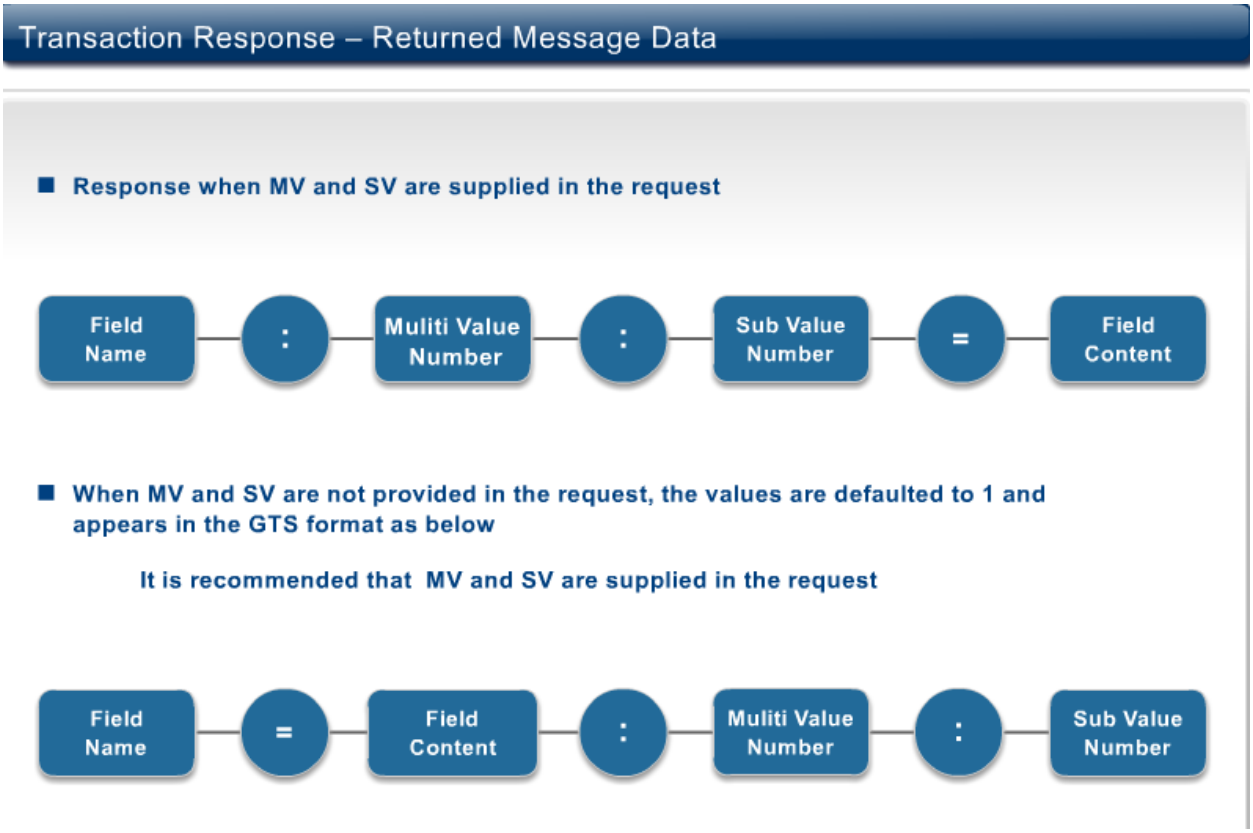
The response varies depending on whether the MV and SV are supplied or not. If the values are supplied, then the response is in the format Field Name : Multi value : Sub value = Field Content. If the Multi value and Sub value parameters are not supplied, then it is defaulted to 1 and appears in the GTS format as Field Name=Field value : Multi Value: Sub Value. It is recommended that we supply the MV and SV in the request.

Each field in this is further subdivided in to 4 different parts.

The four different parts of a Field part of the RETURNED MESSAGE DATA are:

1. Field Name – The field name is as in the STANDARD.SELECTION record of the application
2. Multi value number – the multi value field number is returned
3. Sub value number – the sub value field number is returned
4. Field content – the contents of the field

Note that unlike in a request the multi-value and sub-value positions appear next to the values and not next to the field name.



We shall now see various examples of OFS Transaction type requests & responses. The example here shows a FUNDS.TRANSFER transaction – input request and its response.

In this example a function and a VERSION for FUNDS.TRANSFER application are NOT supplied. While an input function (I) will be assumed by OFS, a comma (,) VERSION will NOT be used. An ID for the FUNDS.TRANSFER transaction is has been supplied in the message either, allowing id auto generation to take place.

The transaction ID automatically generated by the FUNDS.TRANSFER application, is shown in the reply. The record is only in unauthorized state (RECORD.STATUS is INAU), implying that a comma or a zero authorizer version was not automatically used by OFS as default.

Transaction Response – Example 1

```
FUNDS.TRANSFER,,ARCUSER01/654321,,TRANSACTION.TYPE=AC,DEBIT.ACCT.NO=1903
8,DEBIT.CURRENCY=EUR,DEBIT.AMOUNT=1000,CREDIT.ACCT.NO=19089,DEBIT.VALUE.
DATE=20080109
```

```
FT080090DL32//1,TRANSACTION.TYPE=AC:1:1,DEBIT.ACCT.NO=19038:1:1,CURRENCY.MKT.D
R=1:1:1,DEBIT.CURRENCY=EUR:1:1,DEBIT.AMOUNT=1000.00:1:1,DEBIT.VALUE.DATE=20080
109:1:1,CREDIT.ACCT.NO=19089:1:1,CURRENCY.MKT.CR=1:1:1,CREDIT.CURRENCY=EUR:1:1
,CREDIT.VALUE.DATE=20080109:1:1,PROCESSING.DATE=20080109:1:1,CHARGE.COM.DISPLA
Y=NO:1:1,COMMISSION.CODE=DEBIT PLUS CHARGES:1:1,CHARGE.CODE=DEBIT PLUS
CHARGES:1:1,PROFIT.CENTRE.CUST=100288:1:1,RETURN.TO.DEPT=NO:1:1,FED.FUNDS=NO:1
:1,POSITION.TYPE=TR:1:1,AMOUNT.DEBITED=EUR1000.00:1:1,AMOUNT.CREDITED=EUR1000.
00:1:1,CREDIT.COMP.CODE=GB0010001:1:1,DEBIT.COMP.CODE=GB0010001:1:1,LOC.AMT.DE
BITED=1420.00:1:1,LOC.AMT.CREDITED=1420.00:1:1,CUST.GROUP.LEVEL=99:1:1,DEBIT.C
USTOMER=100288:1:1,CREDIT.CUSTOMER=100381:1:1,DR.ADVICE.REQD.Y.N=Y:1:1,CR.ADVI
CE.REQD.Y.N=Y:1:1,CHARGED.CUSTOMER=100381:1:1,TOT.REC.COMM=0:1:1,TOT.REC.COMM.
LCL=0:1:1,TOT.REC.CHG=0:1:1,TOT.REC.CHG.LCL=0:1:1,RATE.FIXING=NO:1:1,TOT.REC.C
HG.CRCCY=0:1:1,TOT.SND.CHG.CRCCY=0:1:1,STMT.NOS=VAL:1:1,OVERRIDE=WITHDRAWL.LT.
MIN.BAL}WITHDRAWL MAKES A/C BAL LESS THAN MIN
BAL:1:1,OVERRIDE=ACCT.UNAUTH.OD}Unauthorised overdraft of & & on account
&.{EUR}2000}19038{EUR{2000{19038{100288{213{{:2:1,RECORD.STATUS=INAU:1:1,CURR.
NO=1:1:1,INPUTTER=213_ARCUSER___OFS_GCS:1:1,DATE.TIME=0902201108:1:1,CO.CODE=G
B0010001:1:1,DEPT.CODE=1:1:1
```

The example here shows a request to authorize an already input FUNDS.TRANSFER transaction and its response.

In this example only the ID of the already input transaction, in unauthorized status, is supplied. Also see that an authorize function A is supplied. There is also no Message Data portion in the message. Check the status field in the response. Yes, it is missing, since the record has become live.

Transaction Response – Example 2

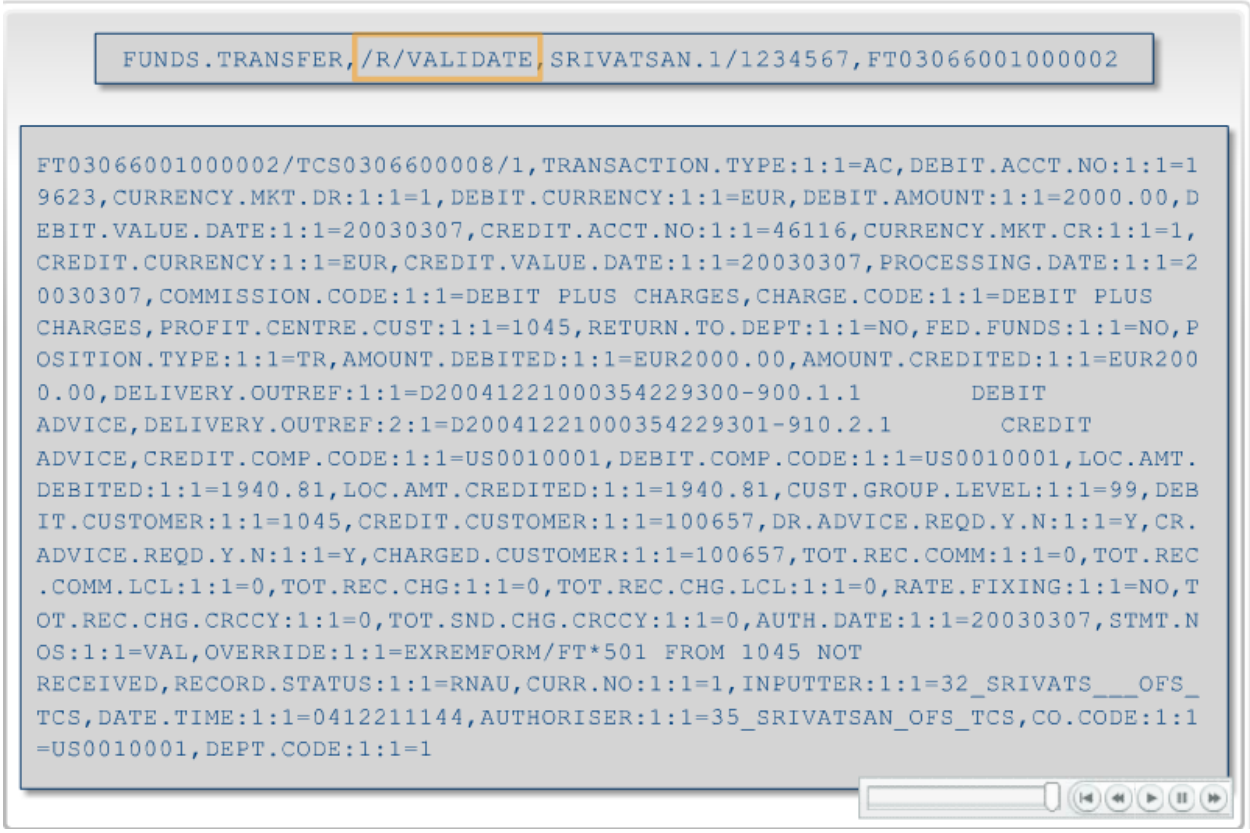
```
FUNDS.TRANSFER,/A,SRIVATSAN.1/1234567,FT03066001000002

FT03066001000002/TCS0306600006/1,TRANSACTION.TYPE:1:1=AC,DEBIT.ACCT.NO:1:1=1
9623,CURRENCY.MKT.DR:1:1=1,DEBIT.CURRENCY:1:1=EUR,DEBIT.AMOUNT:1:1=2000.00,D
EBIT.VALUE.DATE:1:1=20030307,CREDIT.ACCT.NO:1:1=46116,CURRENCY.MKT.CR:1:1=1,
CREDIT.CURRENCY:1:1=EUR,CREDIT.VALUE.DATE:1:1=20030307,PROCESSING.DATE:1:1=2
0030307,COMMISSION.CODE:1:1=DEBIT PLUS CHARGES,CHARGE.CODE:1:1=DEBIT PLUS
CHARGES,PROFIT.CENTRE.CUST:1:1=1045,RETURN.TO.DEPT:1:1=NO,FED.FUNDS:1:1=NO,P
OSITION.TYPE:1:1=TR,AMOUNT.DEBITED:1:1=EUR2000.00,AMOUNT.CREDITED:1:1=EUR200
0.00,DELIVERY.OUTREF:1:1=D20041221000354229300-900.1.1 DEBIT
ADVICE,DELIVERY.OUTREF:2:1=D20041221000354229301-910.2.1       CREDIT
ADVICE,CREDIT.COMP.CODE:1:1=US0010001,DEBIT.COMP.CODE:1:1=US0010001,LOC.AMT.
DEBITED:1:1=1940.81,LOC.AMT.CREDITED:1:1=1940.81,CUST.GROUP.LEVEL:1:1=99,DEB
IT.CUSTOMER:1:1=1045,CREDIT.CUSTOMER:1:1=100657,DR.ADVICE.REQD.Y.N:1:1=Y,CR.
ADVICE.REQD.Y.N:1:1=Y,CHARGED.CUSTOMER:1:1=100657,TOT.REC.COMM:1:1=0,TOT.REC
.COMM.LCL:1:1=0,TOT.REC.CHG:1:1=0,TOT.REC.CHG.LCL:1:1=0,RATE.FIXING:1:1=NO,T
OT.REC.CHG.CRCCY:1:1=0,TOT.SND.CHG.CRCCY:1:1=0,AUTH.DATE:1:1=20030307,STMT.N
OS:1:1=135050003542291.00,STMT.NOS:2:1=1-2,OVERRIDE:1:1=EXREMFORM/FT*501
FROM 1045 NOT
RECEIVED,CURR.NO:1:1=1,INPUTTER:1:1=32_SRIVATS___OFS_TCS,DATE.TIME:1:1=04122
11144,AUTHORISER:1:1=35_SRIVATSAN_OFS_TCS,CO.CODE:1:1=US0010001,DEPT.CODE:1:
1=1
```

The example here shows two things – The use of the Reverse function and the use of the VALIDATE option.

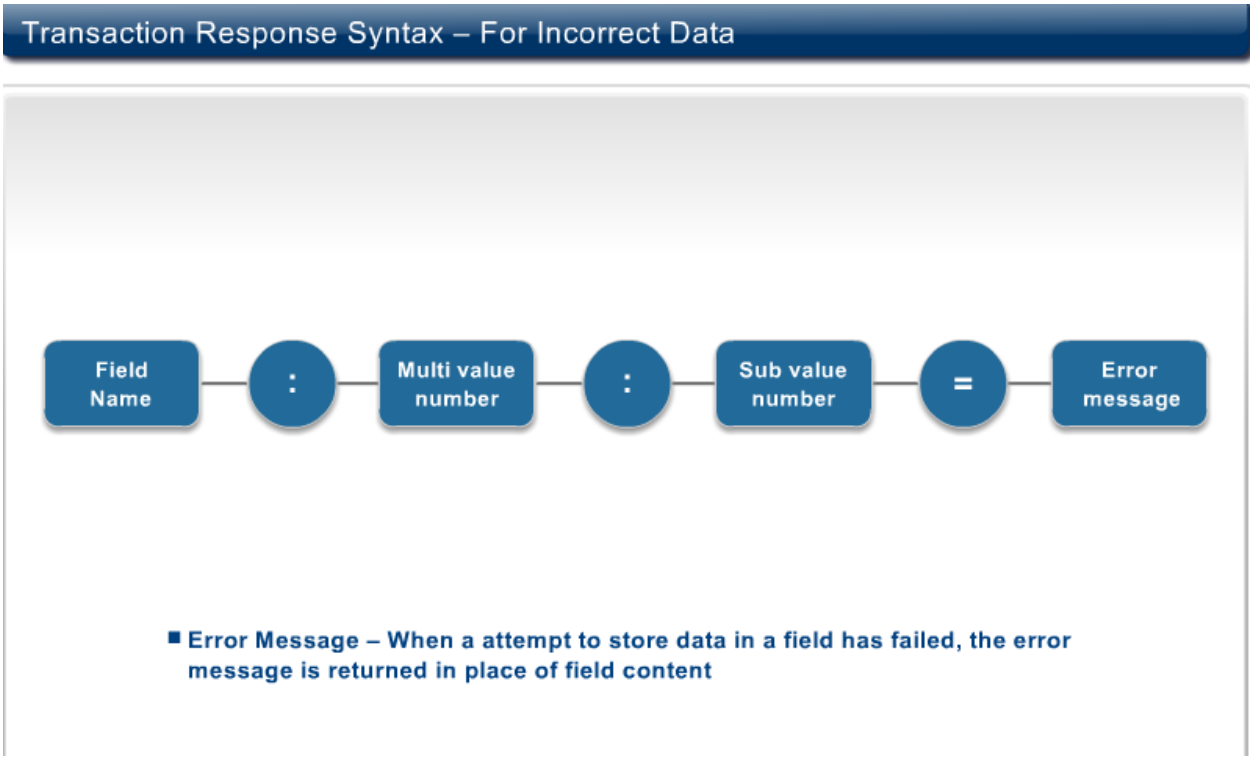
In the example, notice that only the ID of the already input transaction in unauthorized status is supplied in the request, when a reverse function (R) is supplied. There is no Message Data portion in the message. The VALIDATE is supplied in the Options portion, meaning that normal processing takes place without actually updating the database.

Notice that the record status in the response shows RNAU.

Transaction Response – Example 3



The response format is slightly different when there are errors. When a attempt to store data in a field has failed, the error message is returned in place of field content.



A sample message is shown here. The currency of the debit account is CHF whereas in the message it is given as AUD. This results in an error. The error message is returned in place of the field value. “NO” in the response just indicates that the transaction did not get through successfully.

Transaction Response – For Incorrect Data Portion

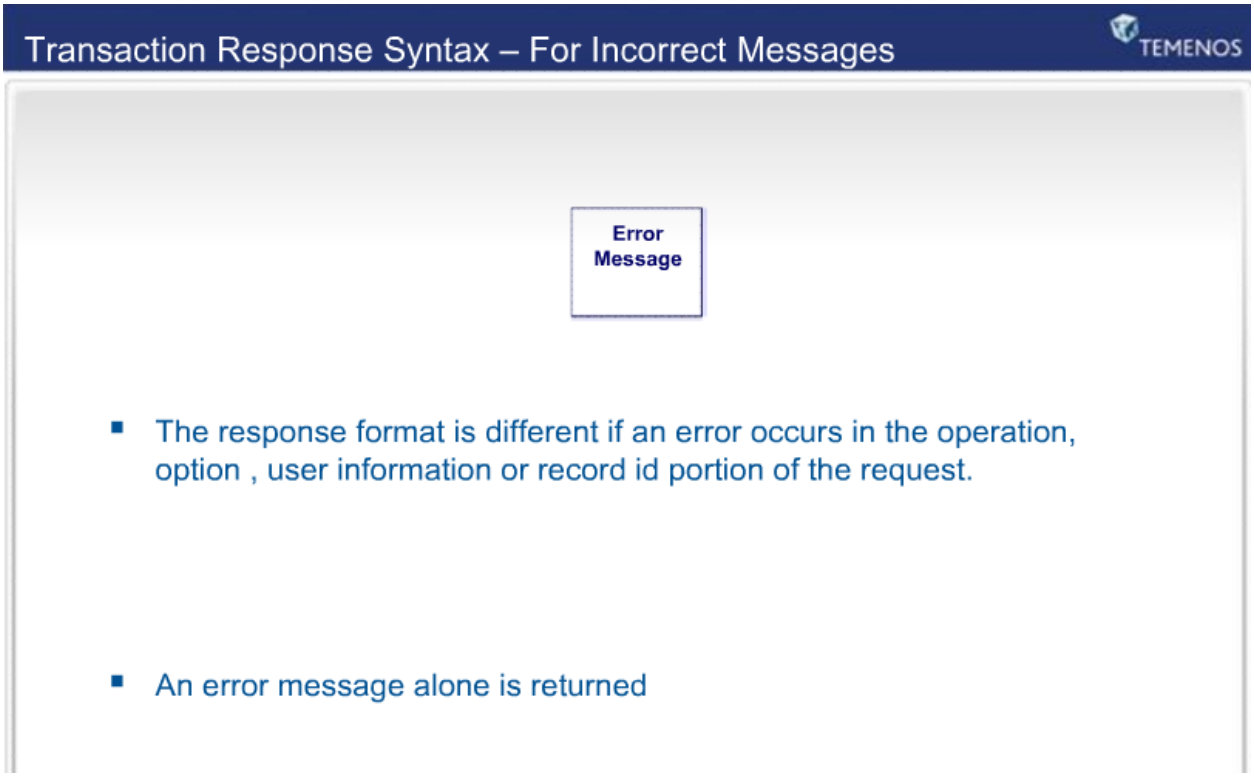
```
FUNDS.TRANSFER,/I/VALIDATE,INPUTT/123123,,
TRANSACTION.TYPE=AC,DEBIT.ACCT.NO=10103,
DEBIT.CURRENCY=AUD,DEBIT.AMOUNT=300,CREDIT.ACCT.NO=10138
,CREDIT.CURRENCY=CHF
```

- **Sample message contains debit currency which is NOT the currency of the account**

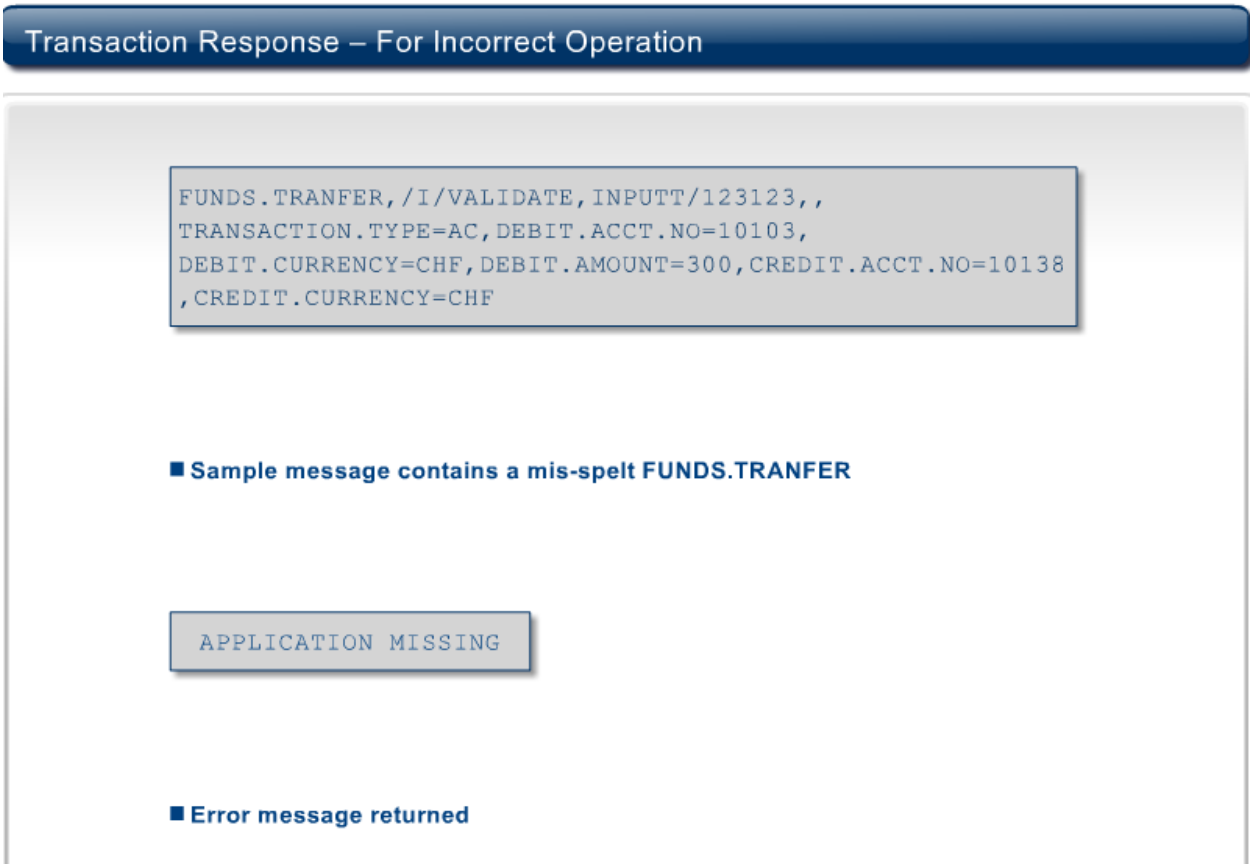
```
FT080091P41D//-1/NO,DEBIT.CURRENCY:1:1=DEBIT ACCT CCY NOT EQ
DEBIT CCY
```

- **Error message returned in place of Debit Currency value**

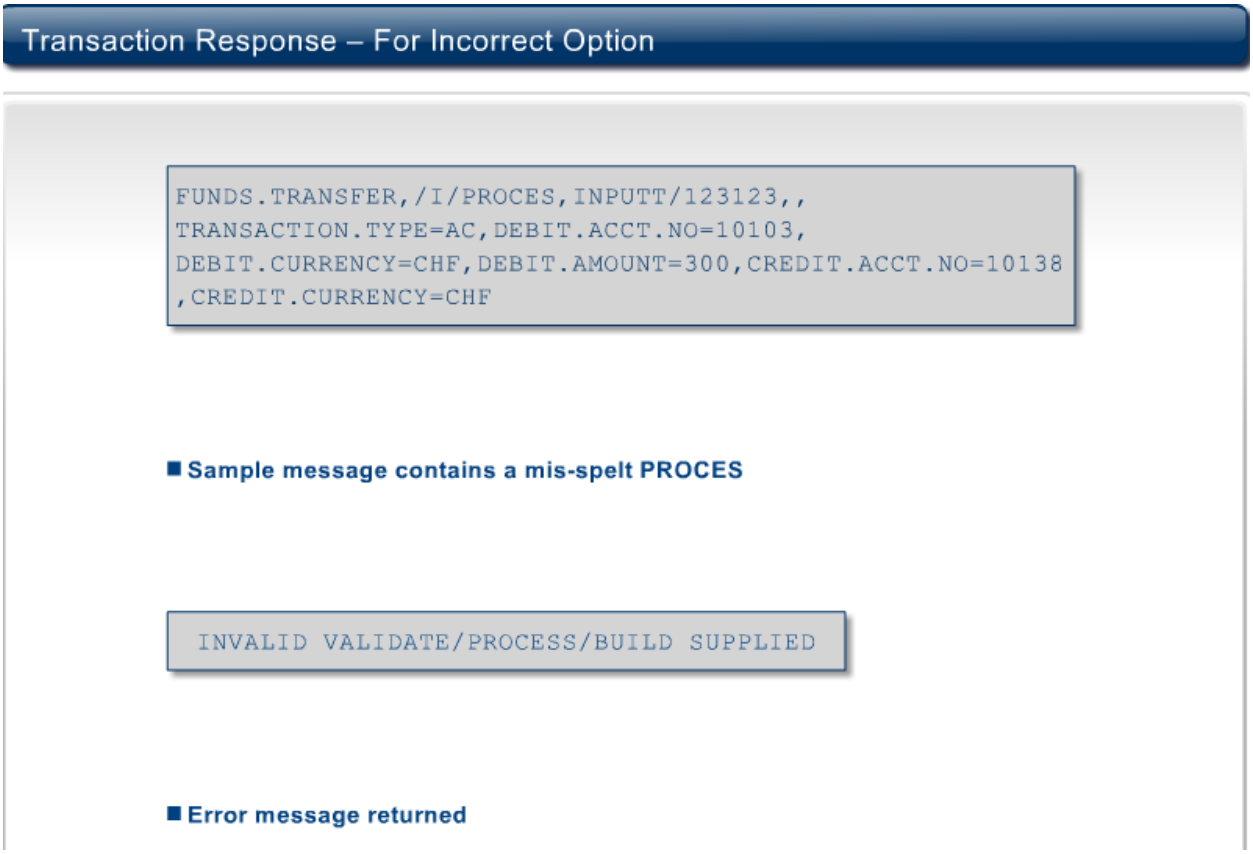
The response format is different if an error occurs in the operation, option, user information or record id portion of the request. An error message alone is returned.



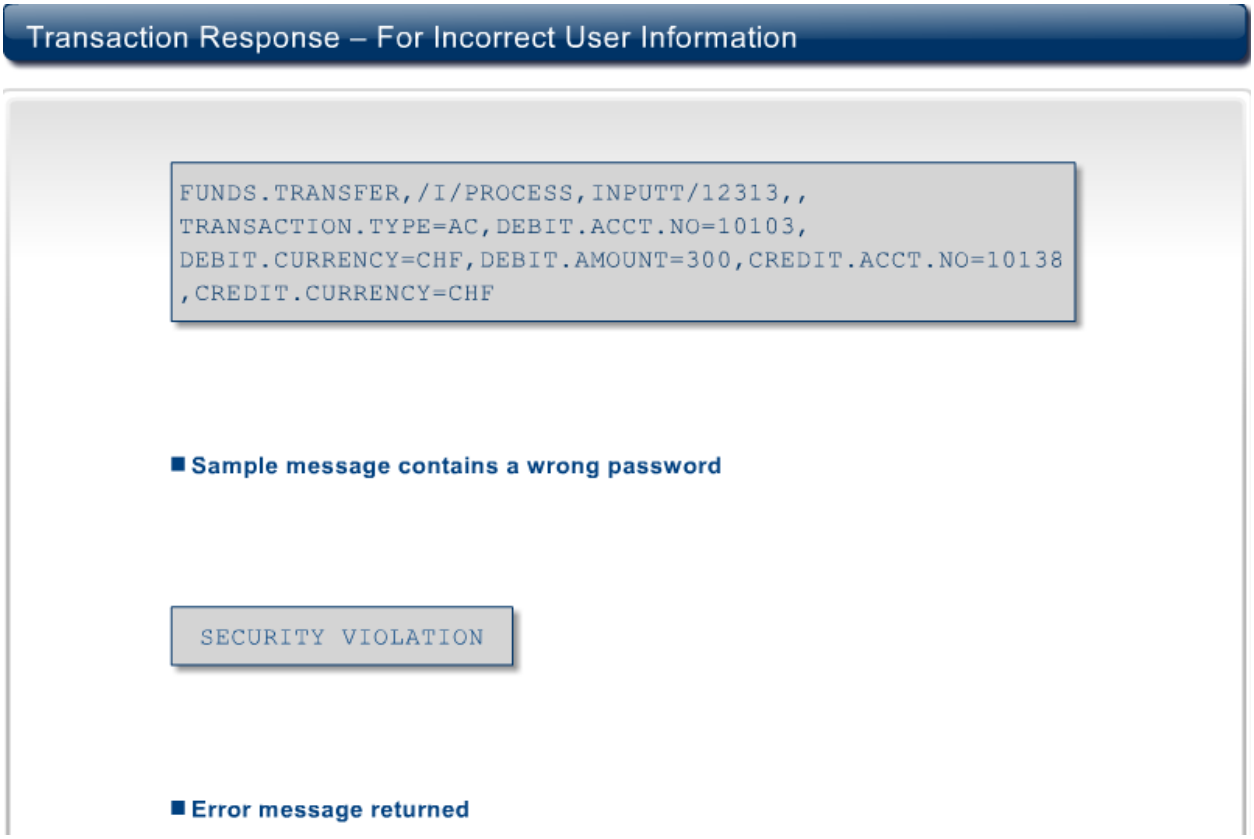
Here you see the transaction response when an incorrect operation, i.e. a wrong application name is given.



Here you see the transaction response when an incorrect option, i.e. an incorrect process type is given.



Here you see the transaction response when an incorrect option, i.e. a incorrect password is given.



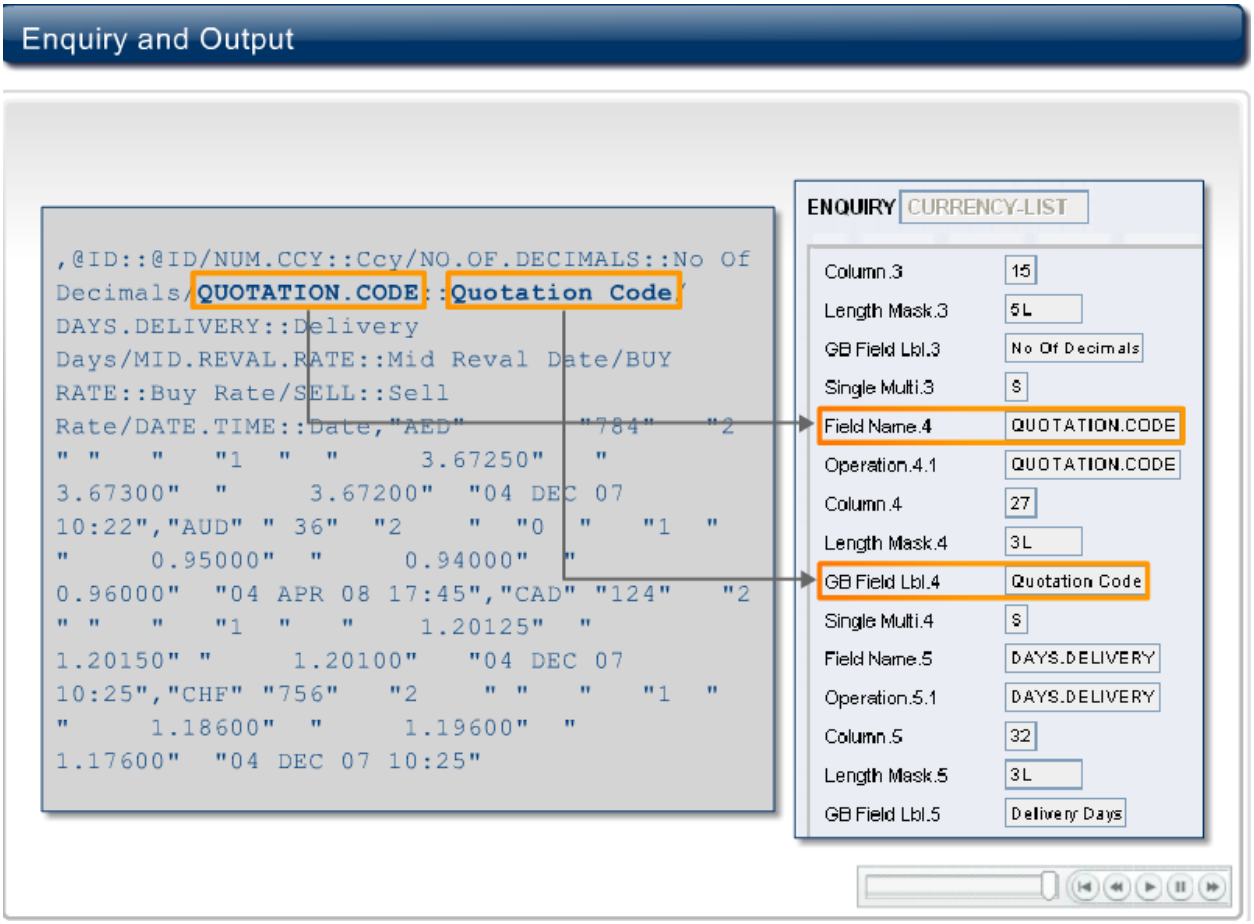
A sample enquiry request and response are shown above. The response is in three parts if you observe carefully. First part is blank, which is the reason that the response begins with a comma. The second part contains some labels (or descriptions), and the third the actual data from the currency application.

Enquiry Response - Sample

```
ENQUIRY.SELECT,,INPUTT/123123,CURRENCY-LIST
```

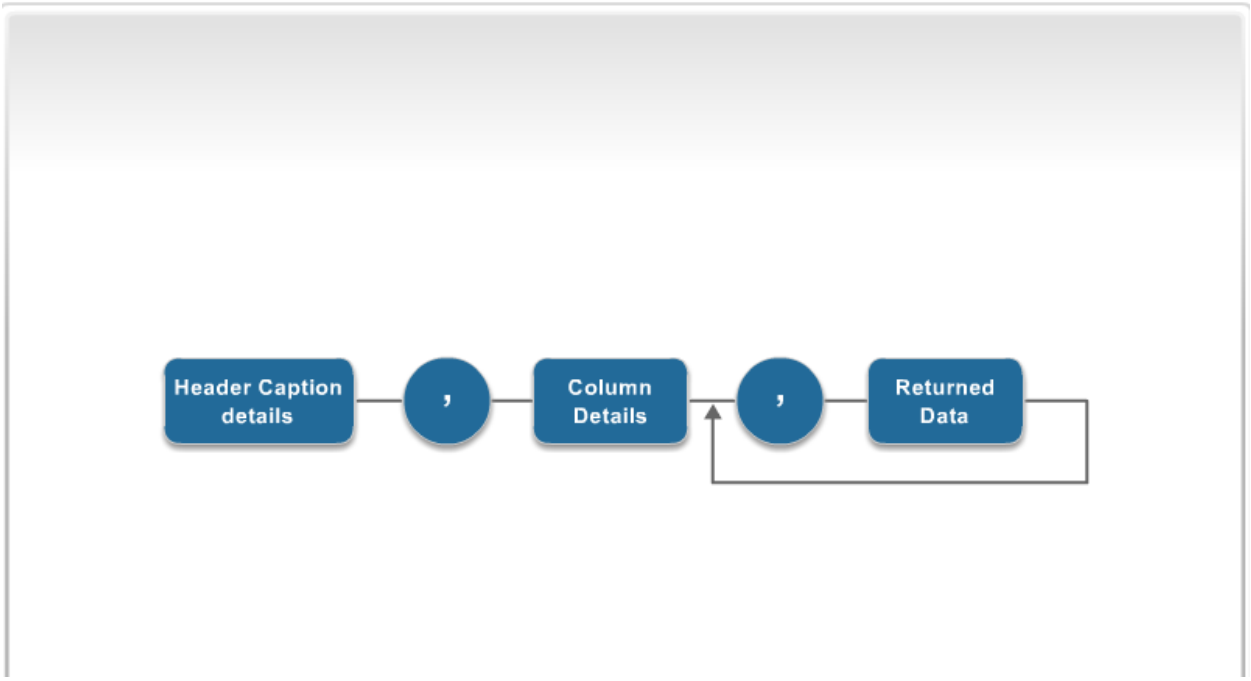
```
,@ID::@ID/NUM.CCY::Ccy/NO.OF.DECIMALS::No Of
Decimals/QUOTATION.CODE::Quotation
Code/DAYS.DELIVERY::Delivery Days/MID.REVAL.RATE::Mid
Reval Date/BUY RATE::Buy Rate/SELL::Sell
Rate/DATE.TIME::Date,"AED"       "784"   "2    " "   "
"1  "  "     3.67250"   "     3.67300"  "     3.67200"
"04 DEC 07 10:22","AUD" " 36"  "2    "  "0  "   "1  "   "
0.95000"  "     0.94000"  "     0.96000"  "04 APR 08
17:45","CAD" "124"   "2    " "   "   "1  "   "
1.20125"  "     1.20150" "     1.20100"   "04 DEC 07
10:25","CHF" "756"   "2    " "   "   "1  "   "
1.18600"  "     1.19600"  "     1.17600"  "04 DEC 07
10:25"
```

A portion of the enquiry CURRENCY-LIST is shown here. Let's try to relate this to the response. You can see that the second part of the response shows Field name and Field Label pairs separated by a double colon (::). Each pair is separated from another by a forward slash (/).



The diagram shows the syntax structure of an OFS Enquiry response. We will now see each of these portions in detail.

OFS Enquiry Response - Syntax



This diagram shows the syntax structure of the Header portion of an OFS Enquiry type response. Each Header Caption Detail is further subdivided in to 2 different parts as shown in the diagram.

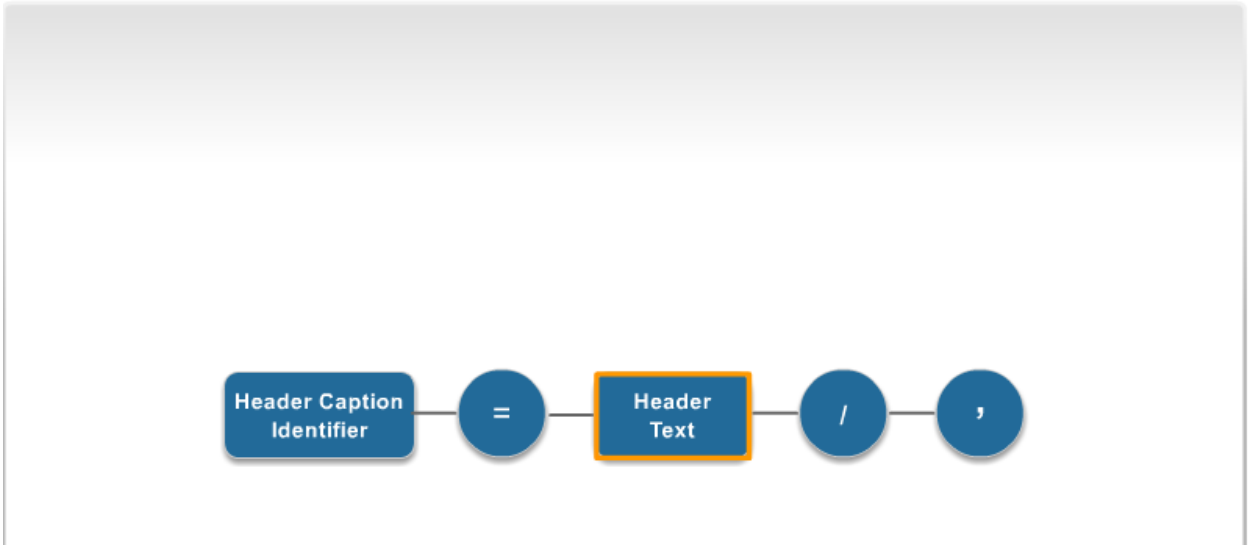
Header Caption Identifier:

This contains an identifier to determine which element of the header or caption is being defined. May be alphanumeric defined in the underlying ENQUIRY.

Header Text:

The Header Text contains the text for the corresponding identifier. Each repeating series of header definitions is separated by a forward slash (/) character.

Enquiry Response Syntax – Header Caption Part



Look at the sample enquiry and its response shown above. The highlighted portion shows the header part of the response.

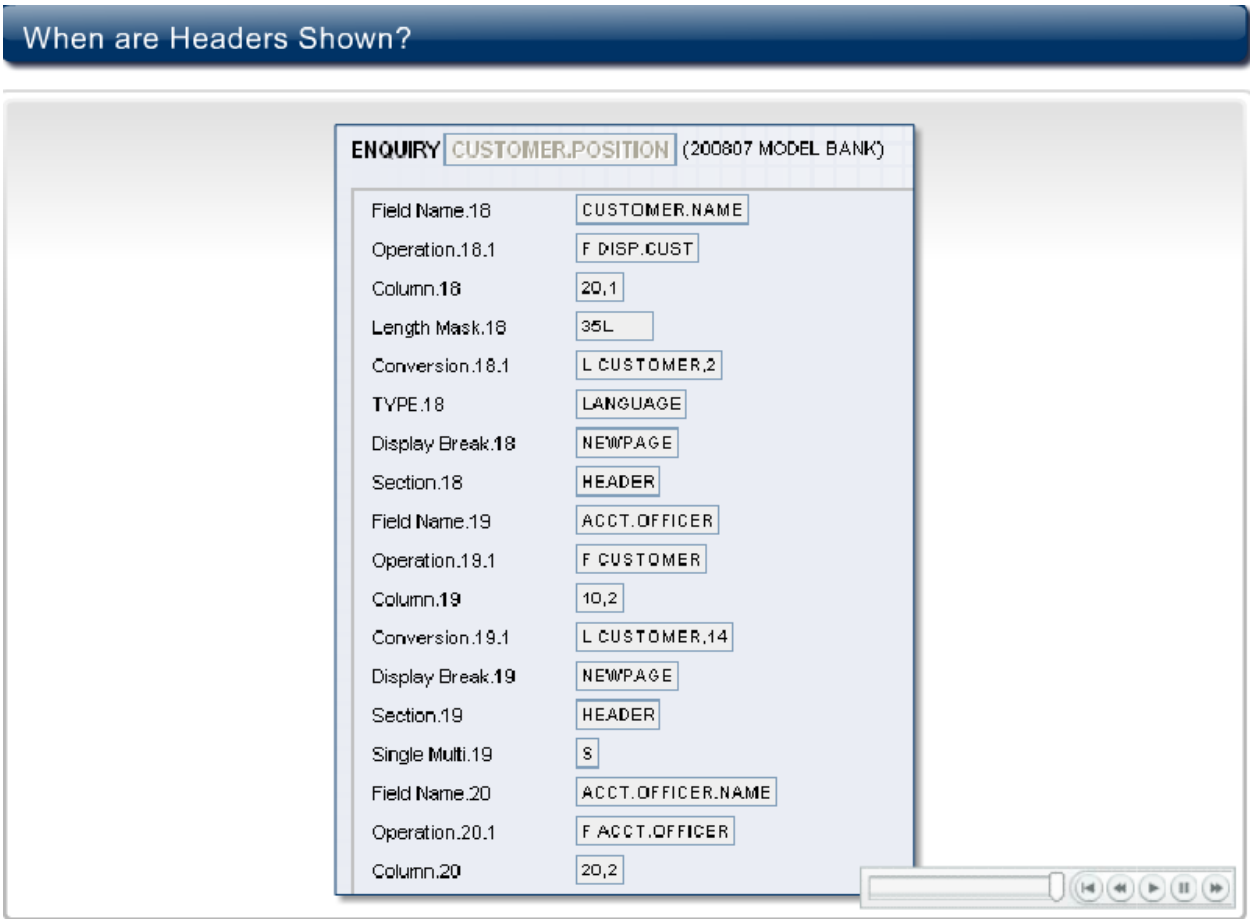
Header - Sample

```
ENQUIRY.SELECT,,INPUTT/123123,CUSTOMER.POSITION,
CUSTOMER.NO:EQ=100285
```

```
HEADER=@ID/HEADER=F DISP.CUST/HEADER=F CUSTOMER/HEADER=F
ACCT.OFFICER, MODULE::Module/TXN.REF::Transaction.Ref/
DISPLAY.NARRATIVE::Display Narrative/DEAL.CCY::Deal
CCY/DISP.AMT::Display AMT/MARGIN.DISP::Margin.Disp/
FORWARD.IND::Forward.IND/
COLL.RGHT.COVER::COLL.RIGHT.COVER/COLL.RIGHT::COLL.RIGHT,
"AC"     "14362" "MICROSOFT"     "USD"    "
2,780,234,562.14 "    ""       ""       "              "
"                ","100285"       "MICROSOFT
","""       "                                  ","" ""
"ACCRUED INTEREST -667,667.69"v
```

Headers are shown when an enquiry field contains data meant to be displayed as a header. Consider the field CUSTOMER.NAME in the CUSTOMER.POSITION enquiry. This is treated as a header. Why? Though

this has not been entered under the header fields of the enquiry, but rather under the field's portion, the column position is fixed at 20, 1 implying that it is to be printed only once.



The diagram shows the syntax structure of the Column Details portion.

Column Identifier :

This contains the FIELD.NAME value of the underlying ENQUIRY.

Column Format Type :

This contains the type of data contained in the column. This information can be used for formatting.

Possible types include :

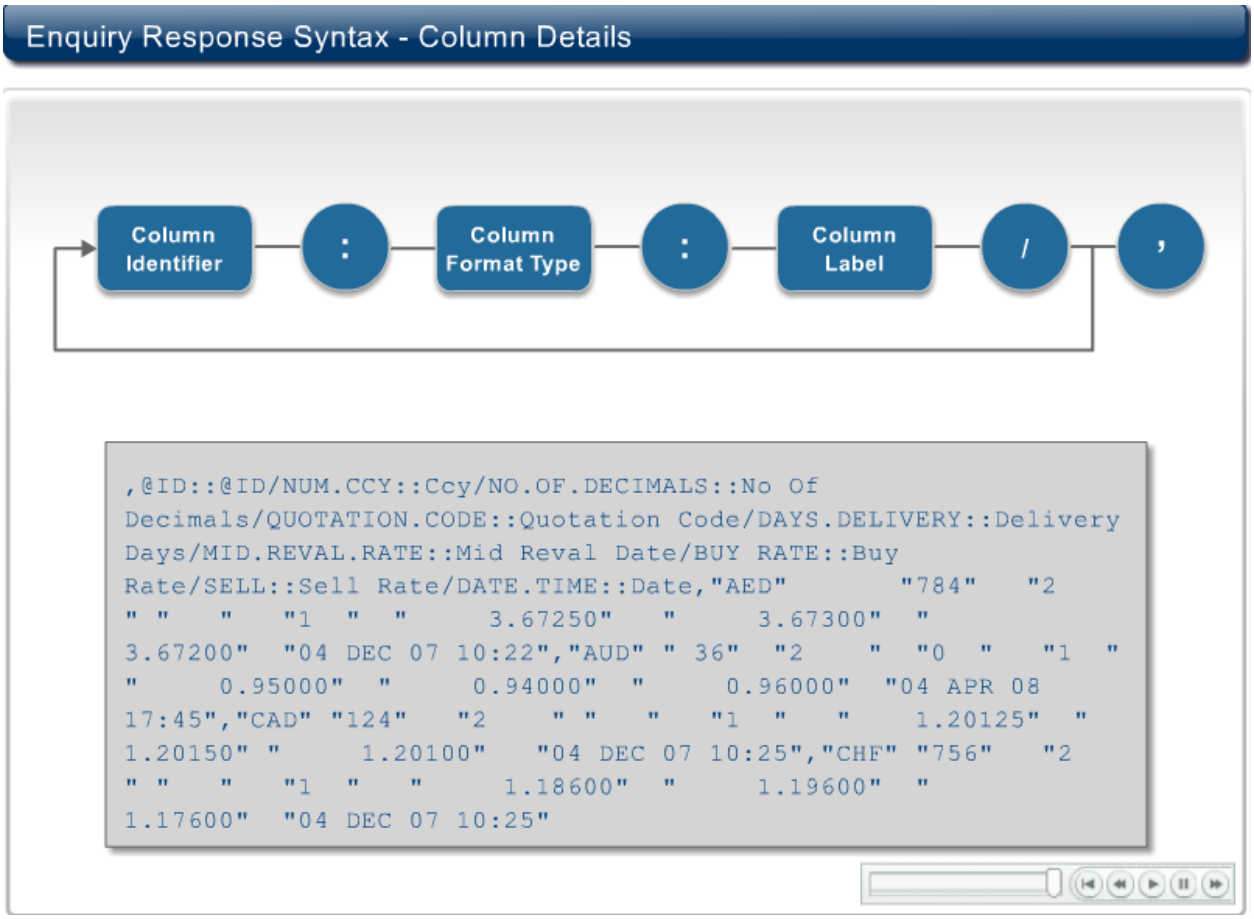
DATE – Formatted using standard date formats.

AMOUNT – Formatted to an amount with decimal format

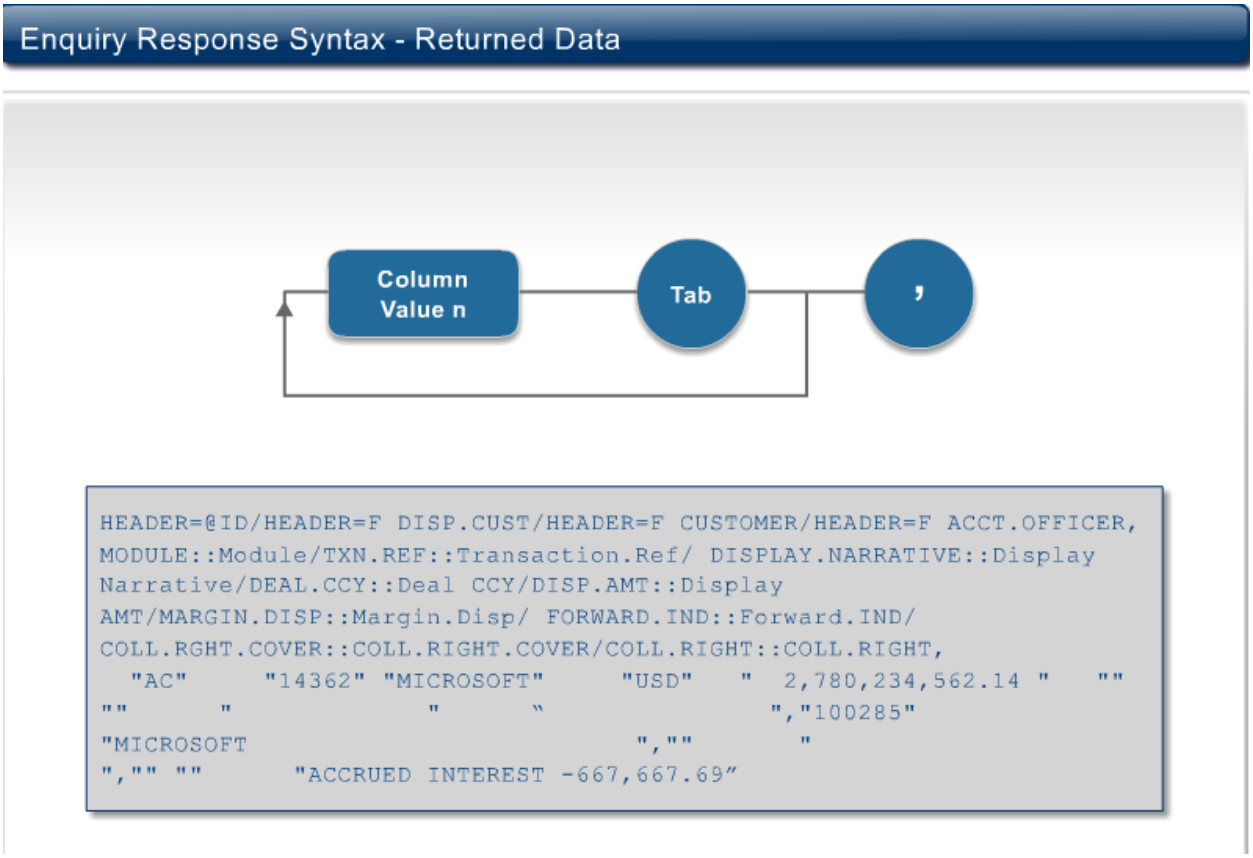
Column Label :

This contains the field label as specified in the FIELD.LBL of the underlying ENQUIRY. Column label defaults to the value of column identifier, in the absence of an entry in FIELD.LBL.

Each series of column details is followed by a forward slash (/) character.



Returned data comprises of a number of rows. Each row is delimited by a comma (,). Each row comprises of a number of columns as defined in the previous element of the message data. For each column, the data value will be returned delimited by a <tab> character.



Telnet mode

You see a sample OFS.SOURCE record for the Telnet mode. Compared to the Batch mode, the differences here are the SOURCE.TYPE, LOGIN.ID and GENERIC.USER fields.

Login Id :

You need to enter the id of an OS level user. It is used to determine whether a direct Telnet connection should be allowed for that particular user. This field is not validated. Since the direct connection is no longer used, this field entry is not relevant. However you need to enter some value here since it is a mandatory field. The sample entry here shows ANY for example.

Syntax Type :

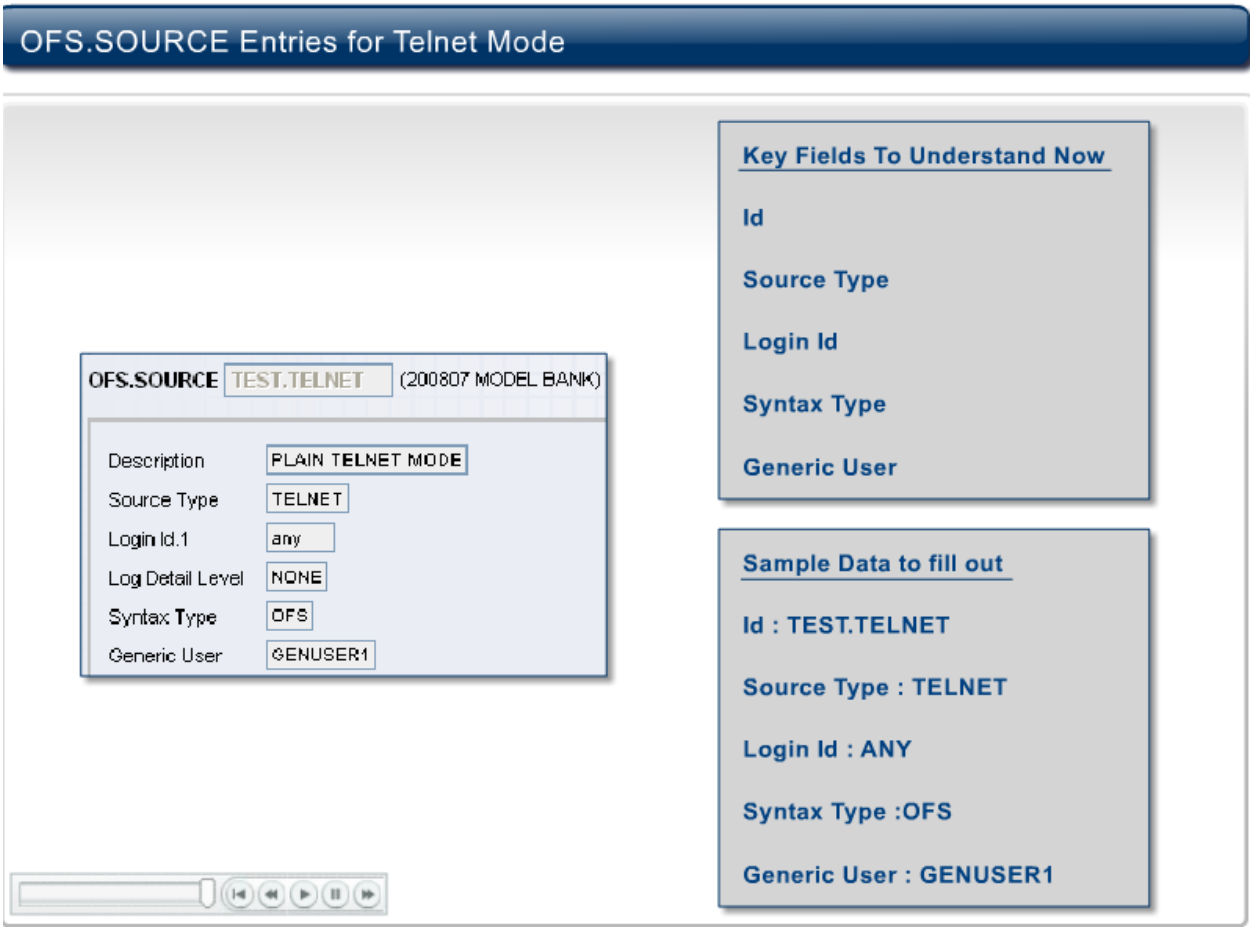
Syntax type is always OFS for the Telnet mode.

Generic User :

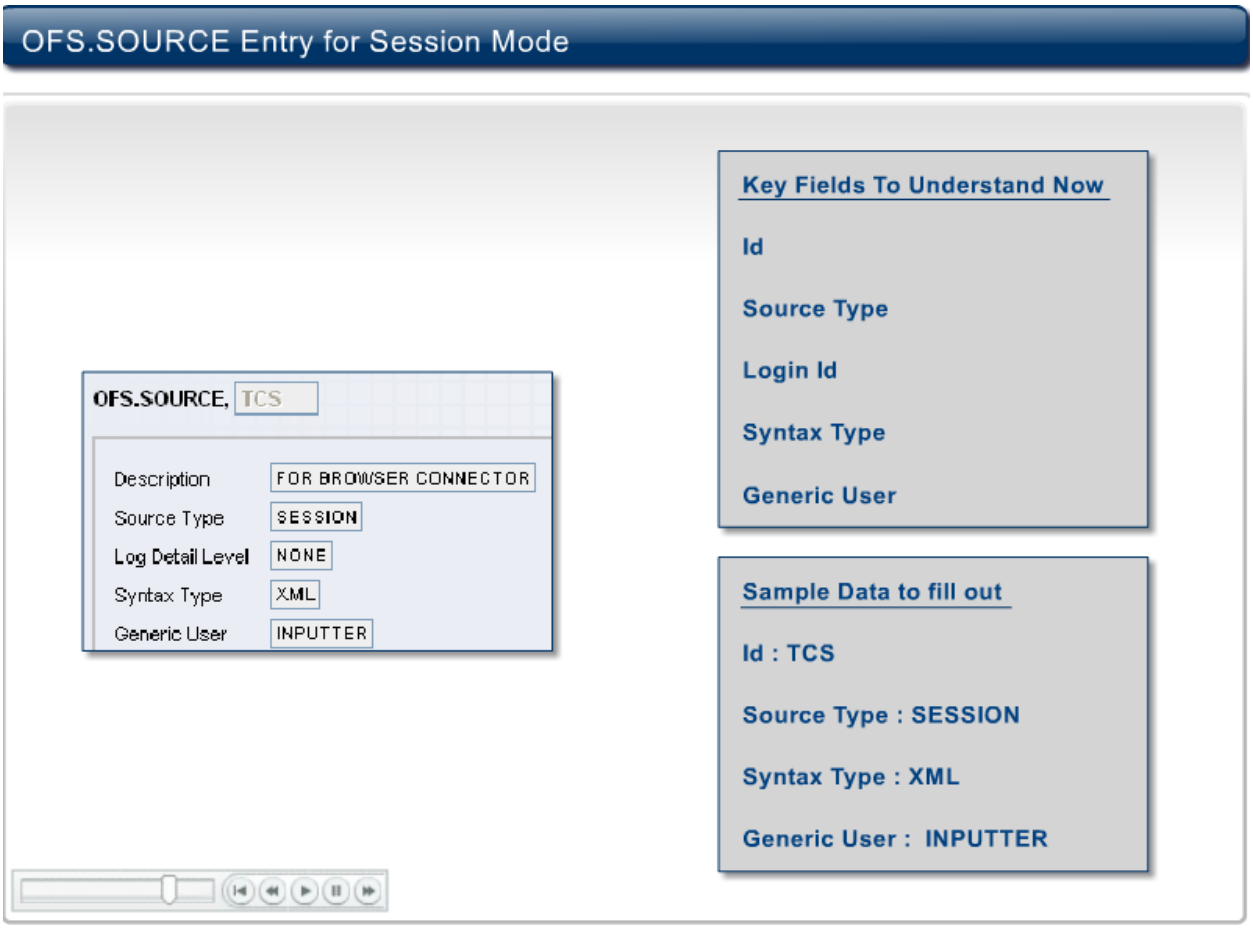
Generic User is mandatory for online connections. This is not relevant for connections via TCS. So then what was its use?

This specified the T24 user context used by the OFS.ONLINE.MANAGER phantom process if the Source Type was Telnet. This user should have had security access to EB.PHANTOM.PH.

If the IB.USER.CHECK was set to yes, which was the case for OFS connections meant for the Temenos Internet Banking product, this user provided the SMS profile for all transactions.



The source type to be used for browser connections is SESSION. However OFS.SESSION.MANAGER checks if the message is an xml type message (i.e. whether it begins with <? xml), and if so sets the Source Type to SESSION.



You can invoke tSS from the jshell prompt to simulate an online connection. Type tSS followed by an OFS.SOURCE record id (whose source type is TELNET). For e.g. tSS TEST.TELNET.

You should see a screen similar to the one showed.

Simulating an Online Connection

Test if the online connection works by entering an OFS Enquiry request.

E.g..

ENQUIRY.SELECT,,AUTHOR/123123,CATEGORY-LIST

Testing Telnet Mode

Response

```
ENQUIRY.SELECT,,AUTHOR/098765,CATEGORY-LIST
,@ID::@ID/Short Name::Short Name,"  1000"        "Demand Acct     ","  1001"     "
Current Account","  1002"       "Currnet Acct 2 ","  1003"      "Current Acct 3
","  1004"      "Current Acct 4 ","  1005"      "Current Acct 5 ","  1006"     "
Call 2          ","  1999"       "Demand Acct    ","  2000"      "Vostro Acct
","  2001"      "Vostro Accounts","  2999"      "Vostro Acct    ","  3000"     "
Loan Accts      ","  3001"       "Mortgage Acct  ","  3101"      "AA Comm. Loan
","  3102"      "AA Pers Loan    ","  3103"      "AA Mortgage    ","  3106"     "
AA LOC          ","  3107"       "AA LOC         ","  3199"      "Mortgage Ac
","  3200"      "All in One Loan","  3201"      "Retail Loan 1  ","  3202"     "
```

Test if connection works with a transaction request as well. The example is shown here.

Note the INPUTTER audit field. It shows the that the inputter is the user that you supplied along with your message, and the OFS source is the one that you supplied as a parameter while invoking tSS.

Testing Telnet Mode

Response

```
ABBREVIATION,/I/PROCESS,AUTHOR/123123,CUST,ORIGINAL.TEXT=CUSTOMER
CUST//1,ORIGINAL.TEXT=CUSTOMER:1:1,RECORD.STATUS=INAU:1:1,CURR.NO=1:1:1,INPUTTER
=15_AUTHORISER__OFS_TEST.TELNET:1:1,DATE.TIME=0811101525:1:1,CO.CODE=GB0010001:
1:1,DEPT.CODE=1:1:1
```

The second method of testing an online connection is to use the raw-tcp listener of TC Server.

A sample TC Server configuration is shown above. Watch for the highlighted portions. The GCS parameter is the id of an Ofs source record. This must be of type telnet. The type of the listener must raw-tcp. The port used must be unique.

Online Connection Through TCS

- **Use the TCS raw-tcp listener**
- **Sample TCS parameters**

```
<ADAPTER id="MBR08MAY-TELNET">
      <MAX_SESSION> 1 </MAX_SESSION>
      <MIN_SESSION> 1 </MIN_SESSION>
      <TIMEOUT>120</TIMEOUT>
      <LOGIN_CONTEXT></LOGIN_CONTEXT>

<STARTIN>C:\localhost\MBR08MAY\bnk\bnk.run</STAR
TIN>
      <JBASEPATH>C:\jbase5\5.0</JBASEPATH>
      <PROGRAM>tSS</PROGRAM>
      <PARAMETER>GCS</PARAMETER>
 </ADAPTER>
 <LISTENER Name="raw.tcp" type="raw-tcp"
active="true">
      <ADAPTERID>MBR08MAY-TELNET</ADAPTERID>
      <PORT> 7023 </PORT>
 </LISTENER>
```

Once you have setup your TC Server, you can connect to the designated port by using any telnet client. For e.g. you can use the default windows telnet client from the command prompt by typing,

TELNET LOCALHOST 7023

After you get the TC Server prompt, you can type in an OFS request and press enter to get it processed.

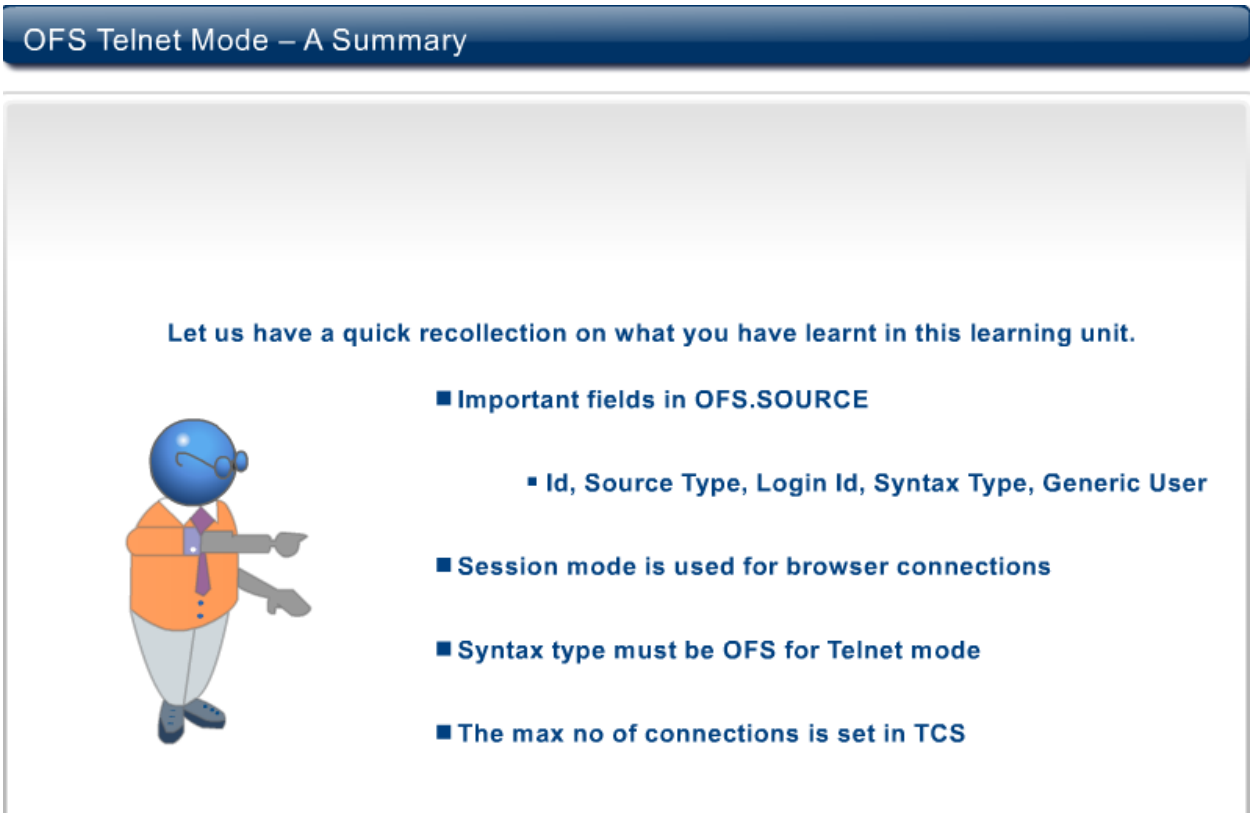
Online Connection Through TCS

TEMENOS

- Telnet to the port that you have configured
  - E.g. type TELNET LOCALHOST 7023
- You should get a prompt similar to

```
TCServer V. tc_1.5.2_1
2008-11-10 16:06:20
Type 'exit' to quit.
```

- Enter a OFS request

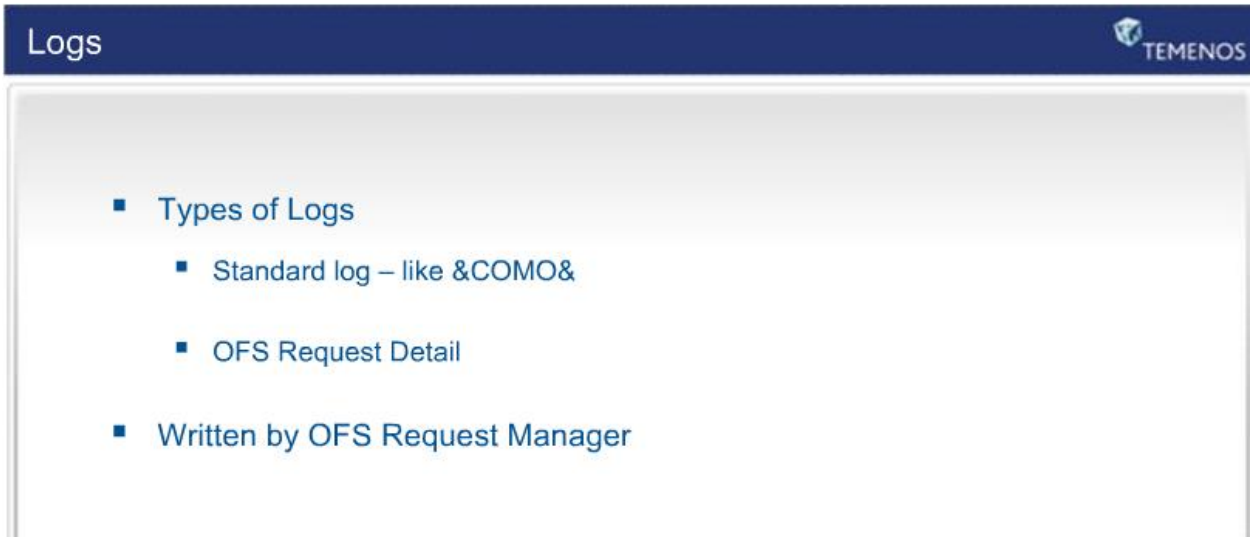


Logging and special fields in OFS

Now, you are going to learn how to configure logs in OFS.

There are two logs that you can set within OFS. One is a log very much like COMO, in that it records on a file, the request and response that could have appeared on a telnet screen.

The second log, uses an application called the OFS.REQUEST.DETAIL (commonly called as ORD). Logging is initiated by the OFS Request Manager.



You need to use two fields in your OFS.SOURCE record to enable logging

Log File Dir

This field contains the name or relative path and name to the log file directory. If the name alone is specified (as in the sample data shown), the log file is created under the T24 Home directory.

This log file is in essence a UD type directory. Yes, it is very much like COMO, in that it records the screen output from OFS, i.e. the request and response. Each session is logged into a record (i.e. a physical file). The record id (i.e. the filename) comprises of the OFS.SOURCE id followed by an underscore followed by a 5 digit number.

Log Detail Level

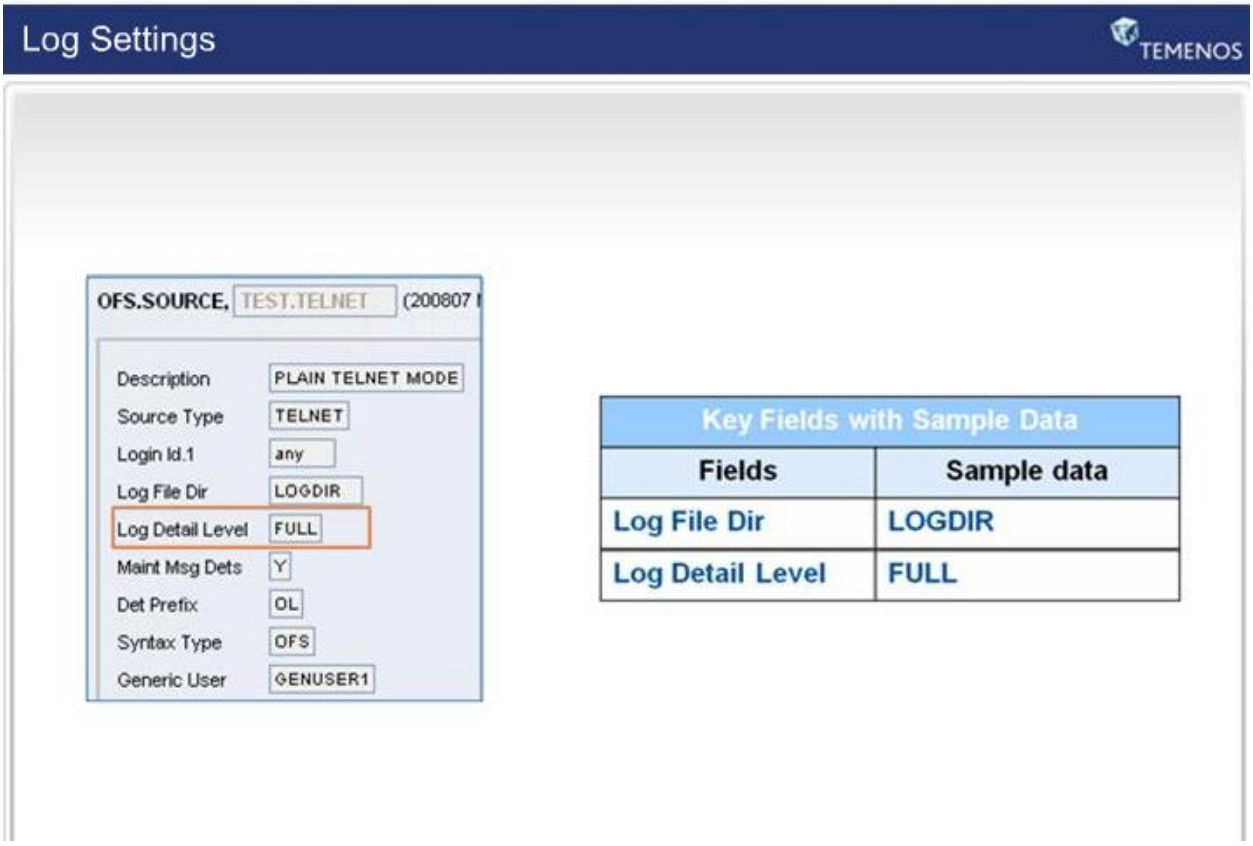
You can choose from three levels, described below;

NONE - Don't log anything.

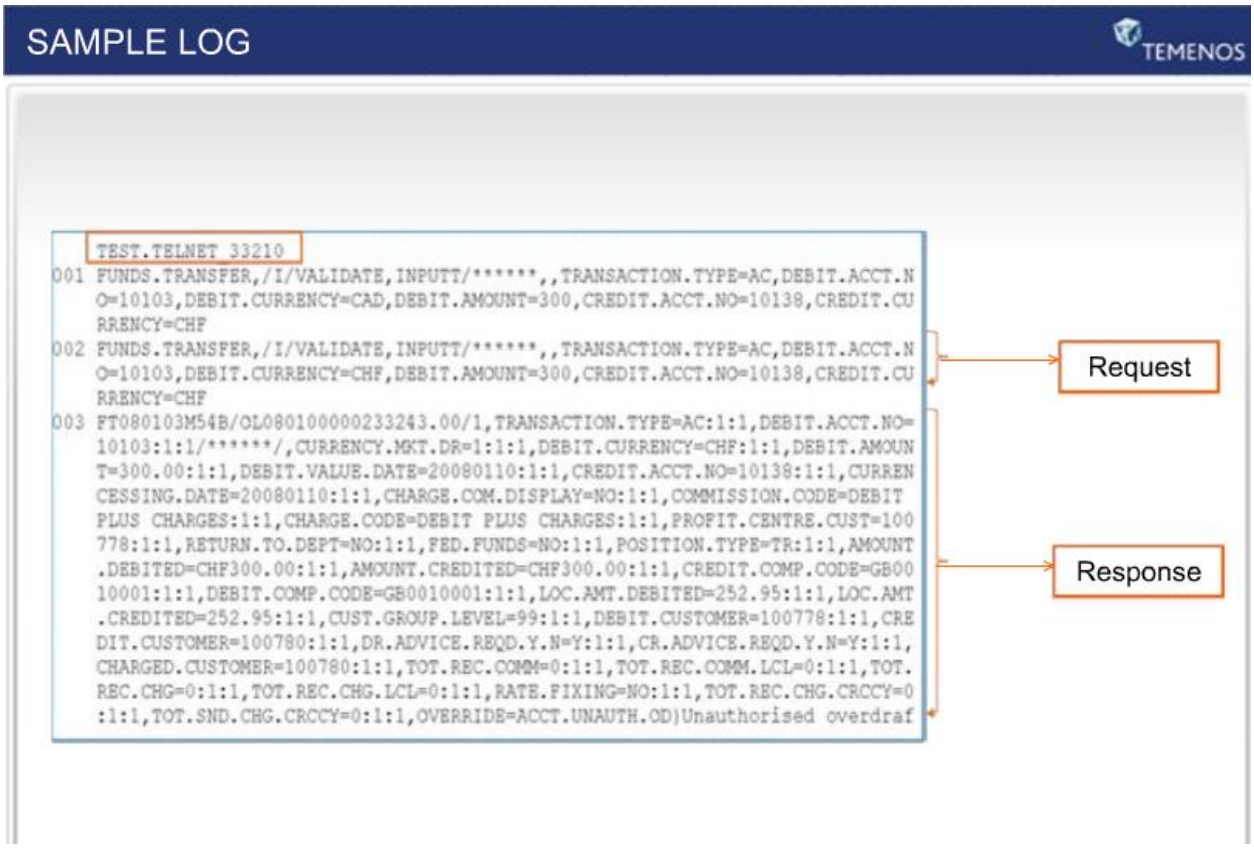
EXCEPT - Log only the Exceptions.

FULL - Logs everything, i.e. log the request and response for all messages.

OPEN - Captures log information, when a background process is started/closed.



You can see a sample log file output here. Notice the record id. Each request is written in one line and a response in the next.



ORD records information such as Application, Function, Transaction Reference, User Name, Company, Date and Time received and processed, the status of the message – whether it was processed or resulted in an error, the message in and message out as well as the GTS control value. Each transaction is recorded in a separate record. The record id could be the transaction id (if your OFS message contained one), or ORD will generate one of its own. Since the ORD is an application it may be queried like any other T24 application.

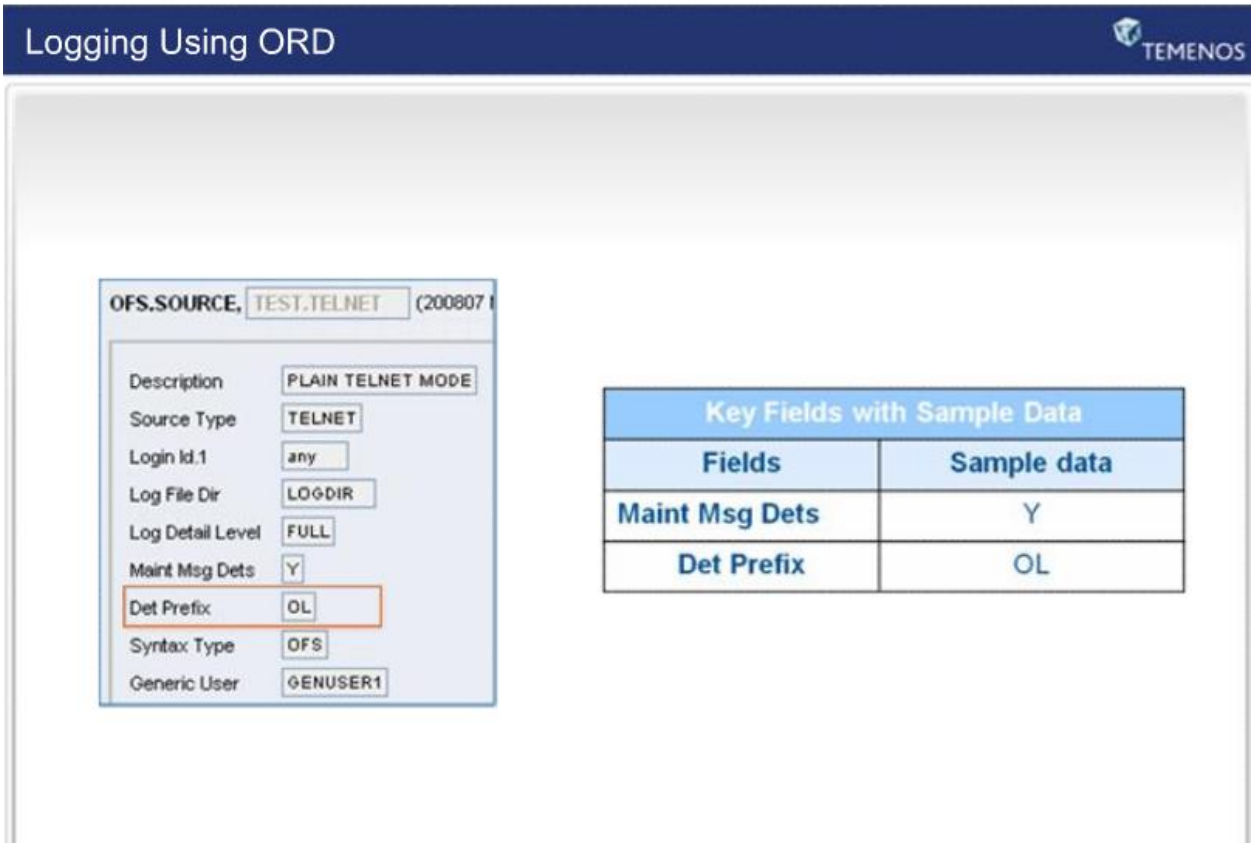
You need to use two fields in your OFS.SOURCE record to enable OFS.REQUEST.DETAIL.

Maint Msg Dets:

This can contain value Y or blank. Y indicates that the ORD has been activated.

Det Prefix:

ORD is a common application used by all the OFS Source records to log information. If you have specified the message id in your OFS message itself you wouldn't have difficulty in locating the ORD record. However , if you haven't, ORD would generate it's own id , and you may find it tedious to locate logs pertaining to your messages from among the many others. ORD eases this by allowing you to specify your own prefix. ORD would then attach the generated id to the prefix and form a record id.



See the list records in ORD. Observe the record ids.

For instance, ABCEDFG10 - This is a message id given by the user as part of the OFS message.

OL080100000233213.01 - This id is generated by OFS. The generated id's follow the format PPPPPPJJJJJUUUUUTTTTT.NN.

Where:

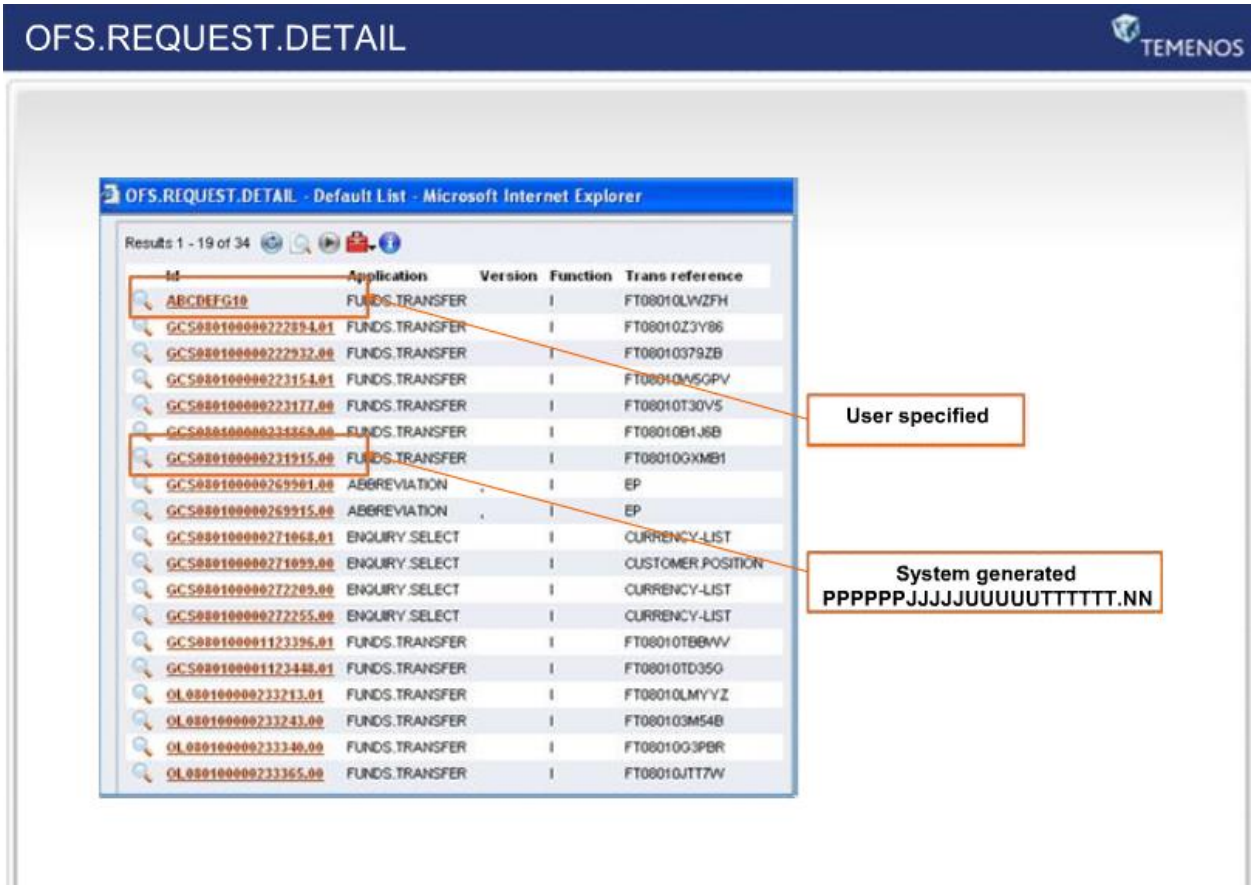
PPPPPP = Six char (max) prefix

JJJJJ = Last five digits of Julian date

UUUUU = User number

TTTTTT = Time in five digits (i.e. seconds since midnight)

NN = Two digit sequence number



You can see two ORD records on the screen. The first one is a log of a message that was successful. The second is that of a message which failed. The status field tells us the message status.

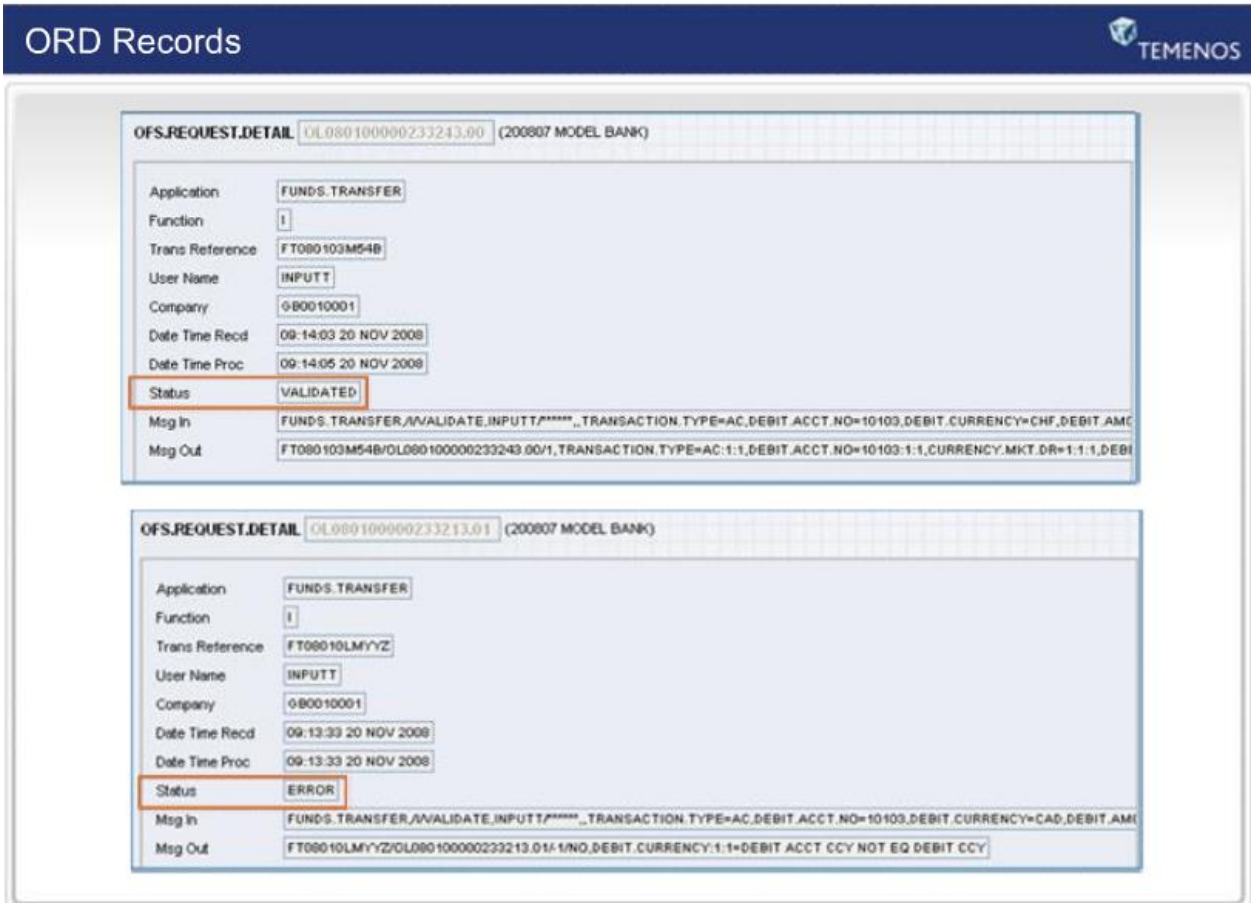
The status of a message could be:

RECEIVED - message has been received by OFS but not processed nor validated.

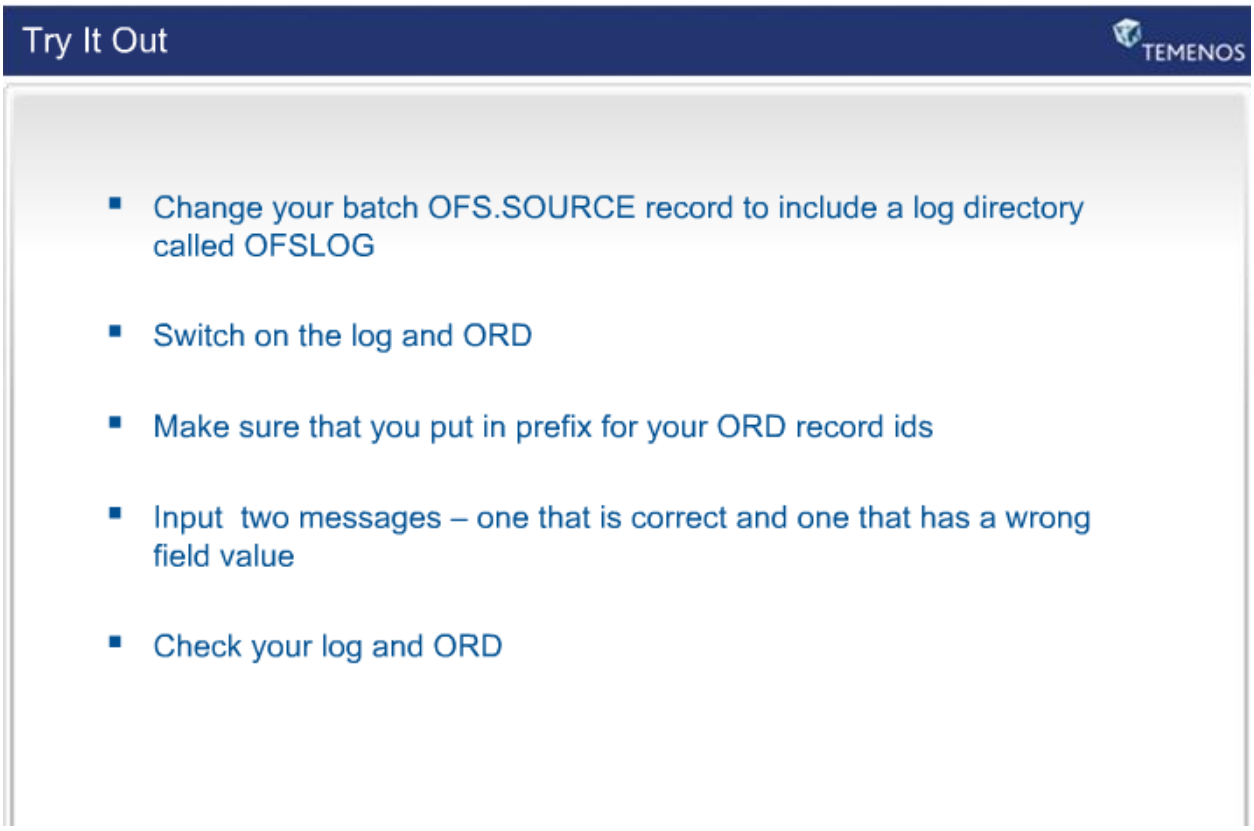
VALIDATED - message has been validated.

PROCESSED - message has been processed.

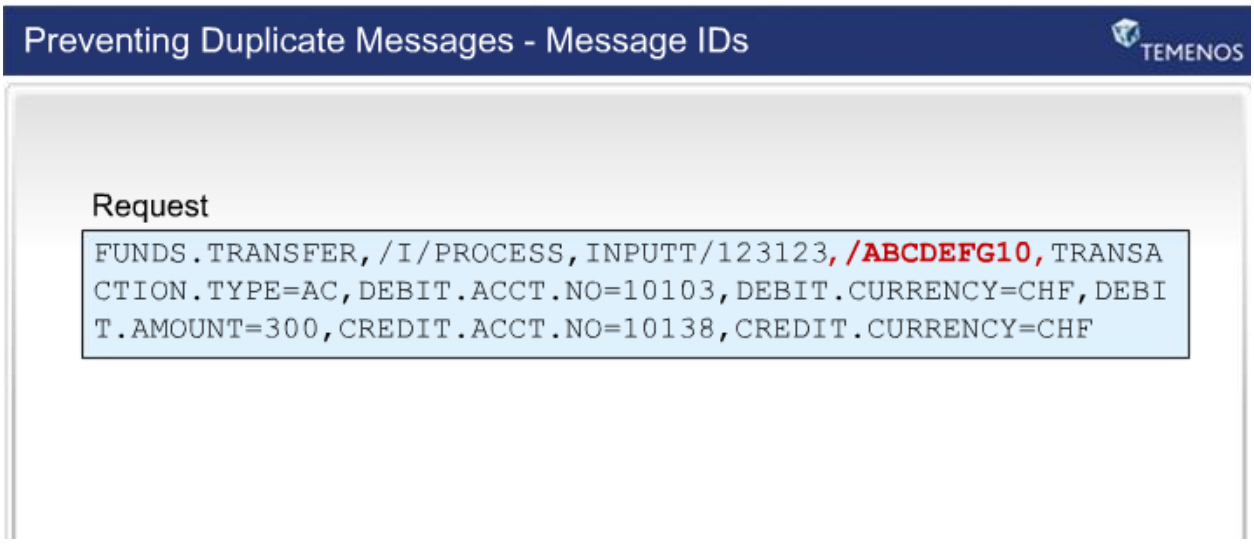
ERROR - message resulted in an error.



Now, you try to change your batch OFS.SOURCE record to include a log directory called OFSLOG. Switch on the log and ORD. Make sure that you put in prefix for your ORD record ids. Input two messages one that is correct and another that has a wrong field value. Finally, check your log and ORD.

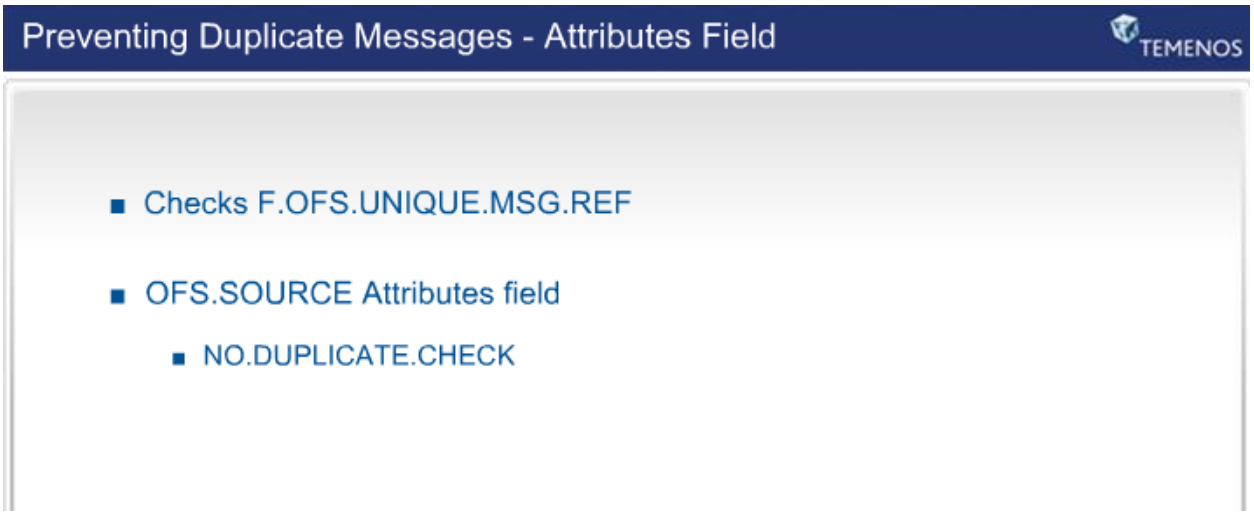


Now, let us see other OFS.SOURCE fields. An OFS message could have a message id embedded within it.

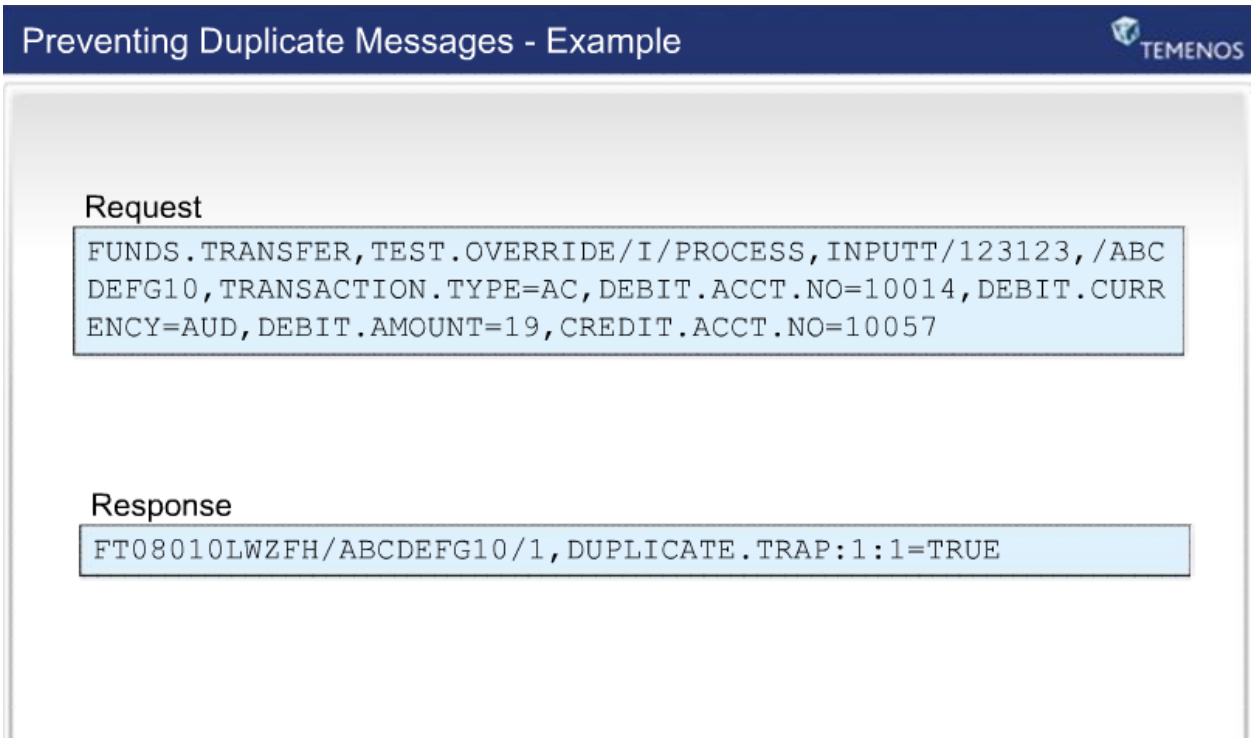


When OFS messages are sent from a third party system, there is a possibility that the same message might be sent twice. How does OFS ensure that a message is not repeated? Simple! It uses the message reference id. This means the message id has to be included within a OFS message and OFS has to store this message reference so that it can check against it later. Where does it store the message reference? In a file called F.OFS.UNIQUE.MSG.REF.

How can we stop OFS from checking for duplicate messages? By entering NO.DUPLICATE.CHECK in the OFS.SOURCE record's Attributes field.



This is a different FT but using the same message id as the previous. Notice the error message. The FT does not get processed because of the duplicate id and a DUPLICATE.TRAP response is returned.

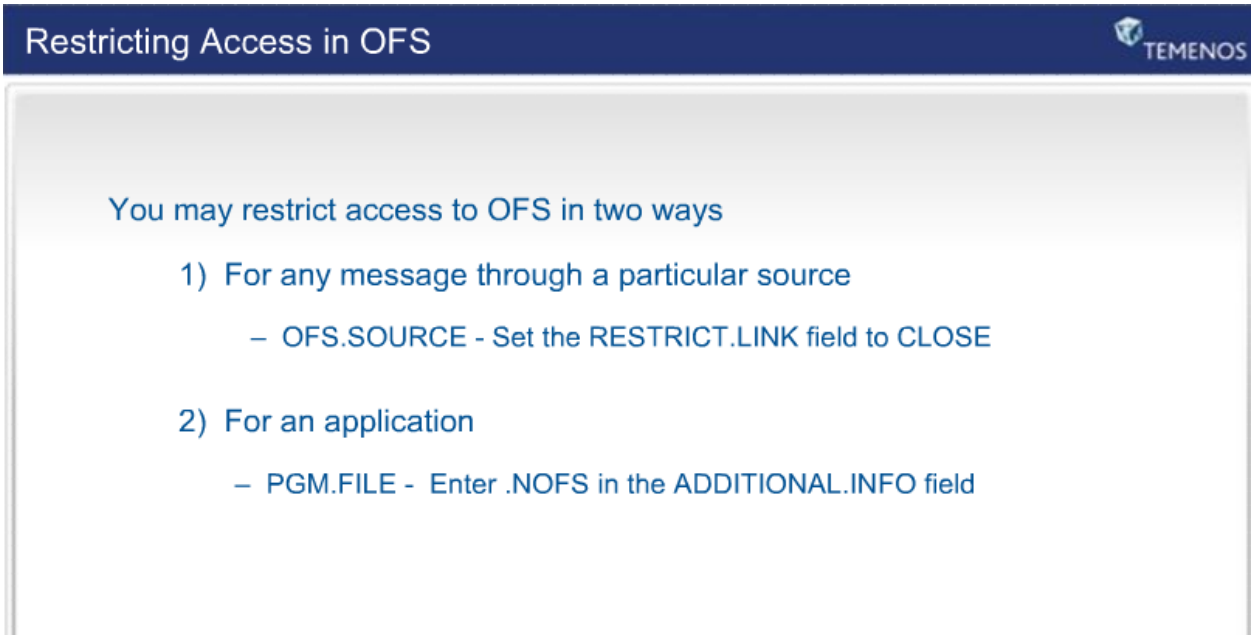


You may restrict access to OFS in two ways.

First, you may restrict at an OFS.SOURCE level. This affects all interactions that passed through a particular source. You do this by setting the RESTRICT.LINK in OFS.SOURCE field to CLOSE.

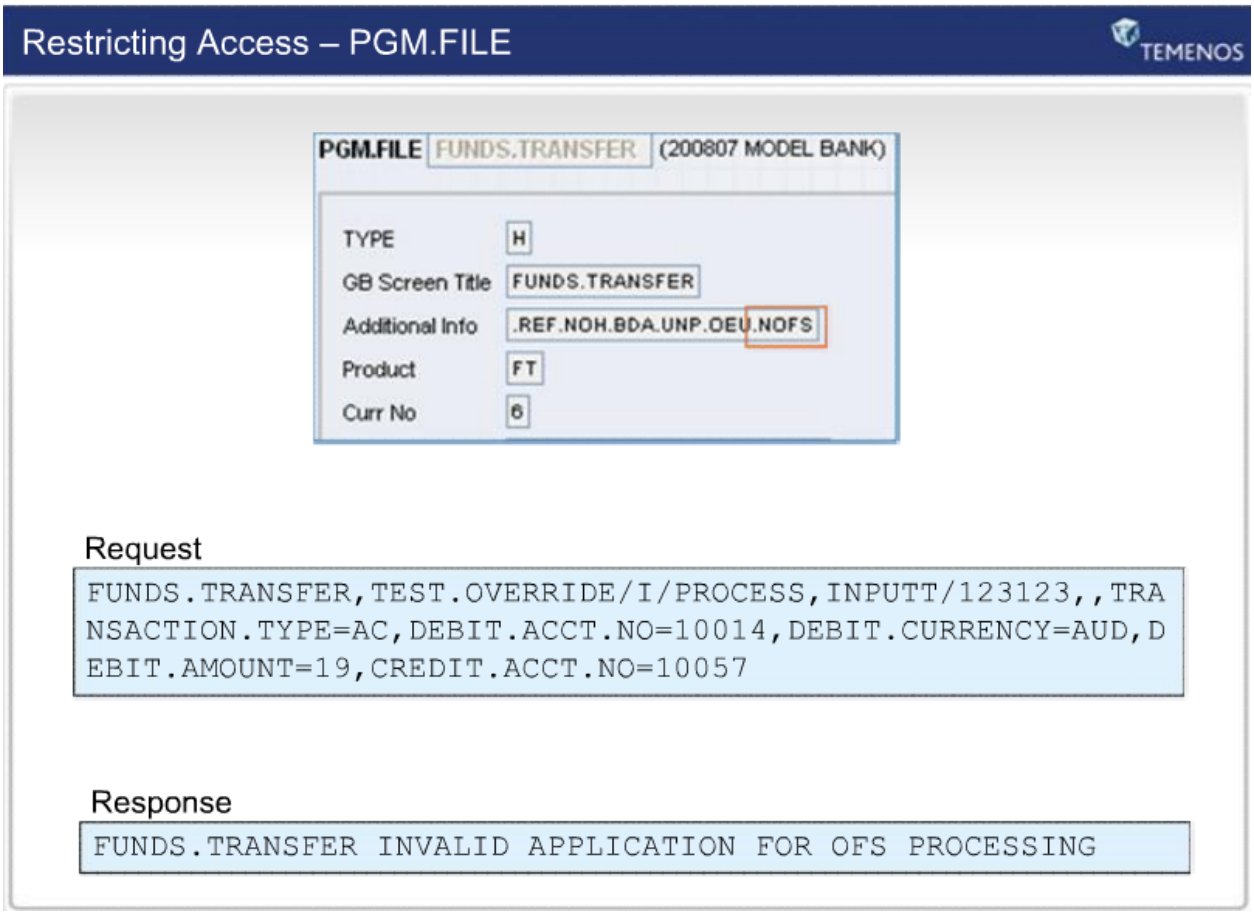
Second, you may restrict at an application level. You have to enter .NOFS in the ADDITIONAL.INFO field, in the PGM.FILE entry for that application.

Please note, Restrict Link has one more option Enq, which allows only enquiries through that OFS source.



The screen shot shows the PGM.FILE entry for FUNDS.TRANSFER with .NOFS included in the ADDITIONAL INFO field.

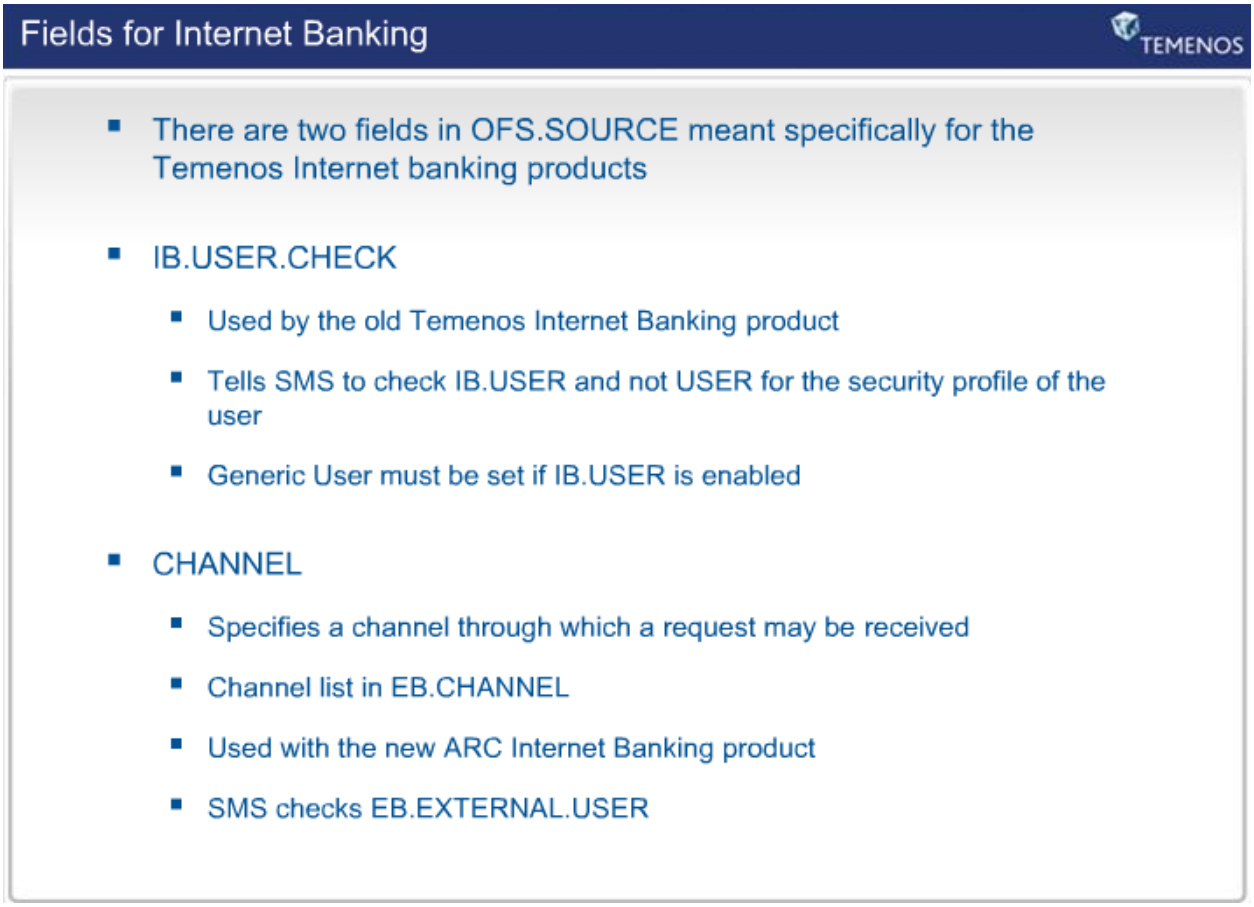
You can see a sample OFS Request to carry out a Funds Transfer with its response. The response clearly indicates that the application does not permit OFS processing.



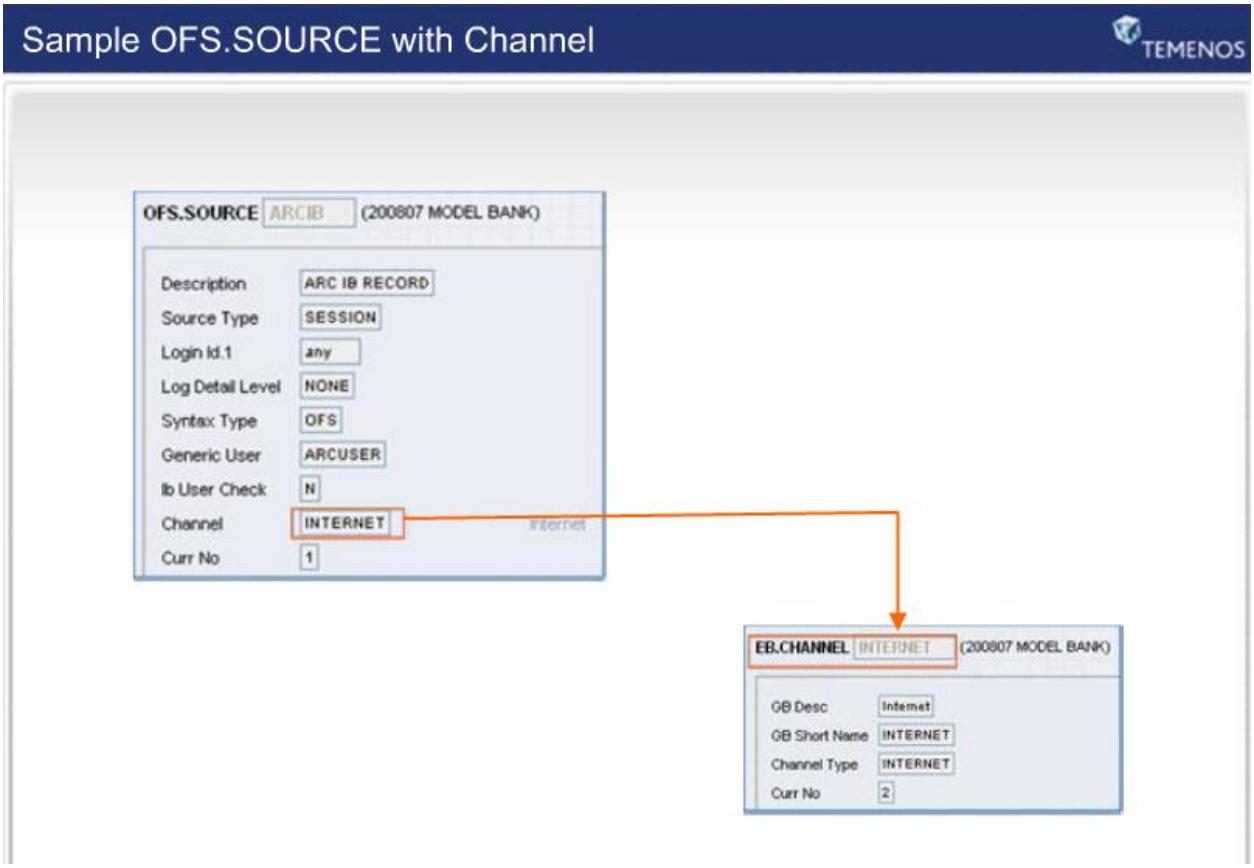
There are two fields in OFS.SOURCE meant specifically for the Temenos Internet banking products. The internet banking products of Temenos do not use the standard USER application to maintain security profiles.

IB.USER.CHECK field is used by the old Temenos Internet Banking product. Setting this tells SMS to check IB.USER and not USER for the security profile of the user. The Generic User in OFS Source must be specified and valid if IB.USER is enabled. This is because access to the T24 system is controlled using security profile of the Generic User.

CHANNEL specifies a channel through which a request may be received. A list of channels are maintained in EB.CHANNEL. This is used with the new ARC Internet Banking product. When this is enabled, SMS checks EB.EXTERNAL.USER to validate the user.



A sample OFS.SOURCE record containing a Channel field is shown in this page. The channel is defined in EB.CHANNEL.

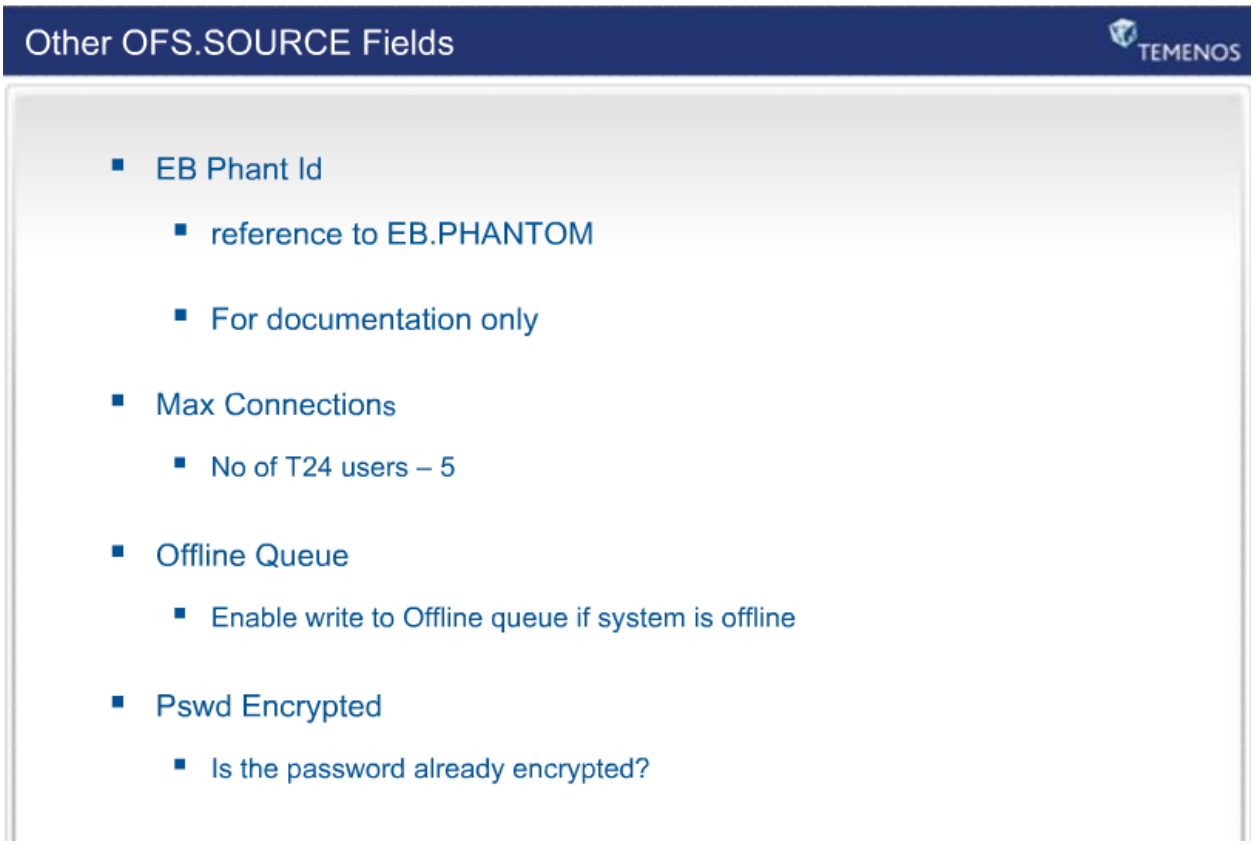


EB Phant Id serves to document the EB.PHANTOM id within OFS Source if you are using the Batch mode.

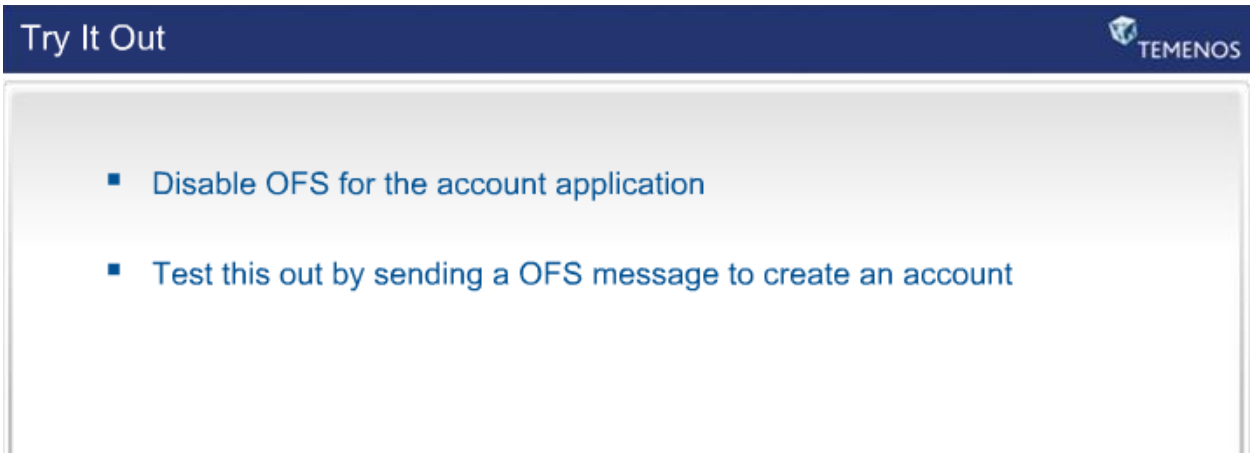
Max Connections may specify the max number of connections permitted through a OFS source. There is a check in T24 whereby it does not allow a value which is more than that of the no of users in SPF minus 5.

Offline Queue Setting enables messages to be written to the Offline queue if system is offline. This is no longer relevant with the advent of Non Stop processing.

Pswd Encrypted Setting field informs T24 if the password is already encrypted or not. If left blank or set to no, T24 will assume that the password is not encrypted and therefore encrypt the password and use the encrypted value for storing or for checking.



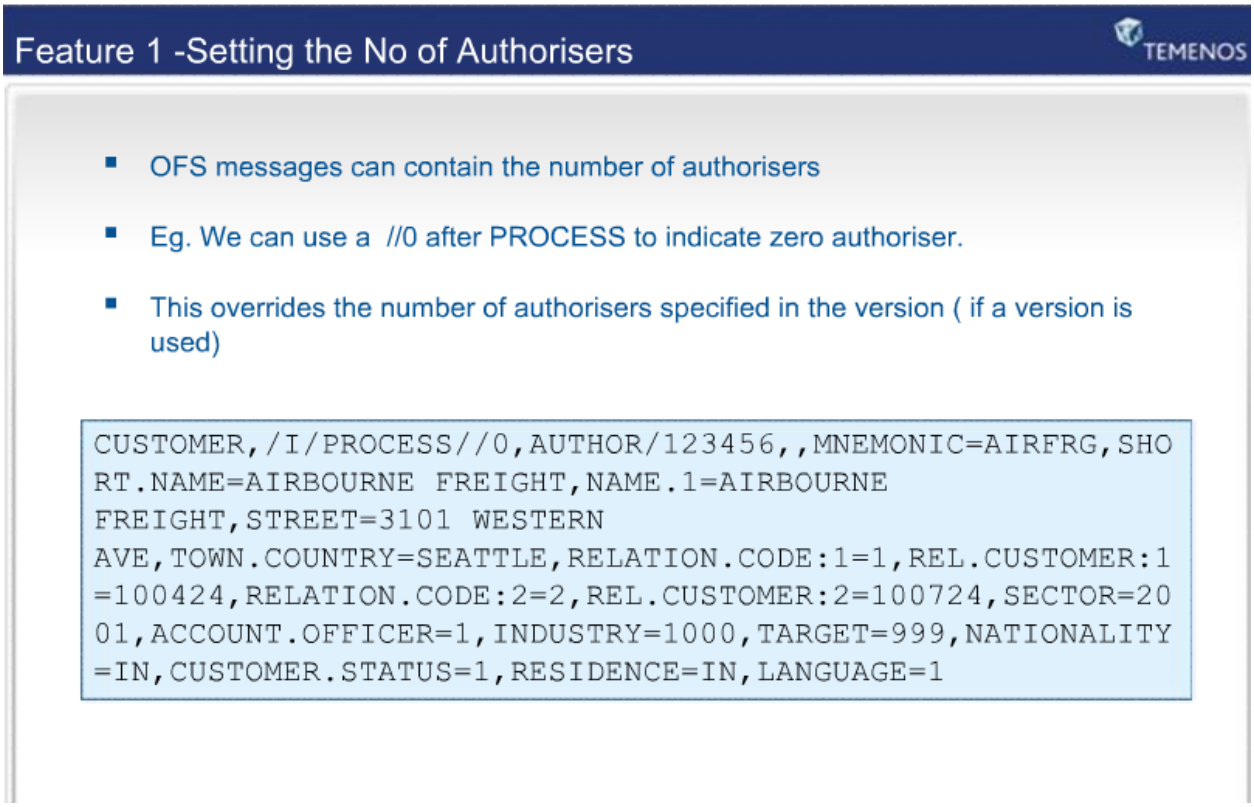
Now, you try to disable OFS for the account application and test this out by sending an OFS message to create an account.



Special features of messages

The OFS message allows us to specify the number of authorisers for a given transaction. For instance, we can use a //0 after PROCESS to indicate zero authorisers. What happens if you have used a version ? Remember the version also contains number of authorisers! Well, the parameter in the OFS message supersedes the number of authorisers specified in the version.

Look at the sample OFS message shown. This is an OFS message to input as well as authorise a new customer.



This slide shows the response for the request seen earlier. Note that the record is in the authorised state. i.e. Status is blank.

Setting the No of Authorisers… - Response

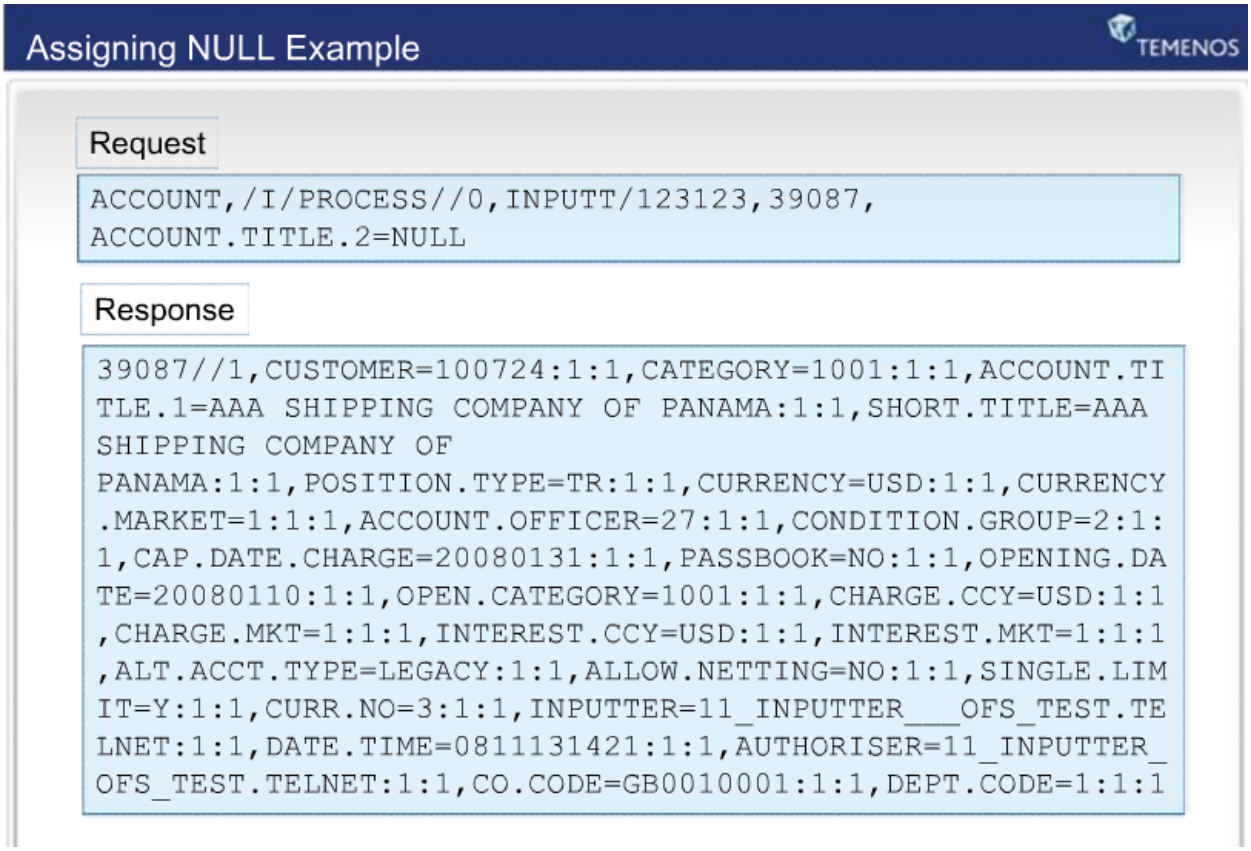
```
100966/IM0723600001/1,MNEMONIC=AIRFRG:1:1,SHORT.NAME=AIRB
OURNE FREIGHT:1:1,NAME.1=AIRBOURNE
FREIGHT:1:1,STREET=3101 WESTERN
AVE:1:1,TOWN.COUNTRY=SEATTLE:1:1,RELATION.CODE=1:1:1,RELA
TION.CODE=2:2:1,REL.CUSTOMER=100424:1:1,REL.CUSTOMER=1007
24:2:1,REVERS.REL.CODE=10:1:1,REVERS.REL.CODE=12:2:1,SECT
OR=2001:1:1,ACCOUNT.OFFICER=1:1:1,INDUSTRY=1000:1:1,TARGE
T=999:1:1,NATIONALITY=IN:1:1,CUSTOMER.STATUS=1:1:1,RESIDE
NCE=IN:1:1,LANGUAGE=1:1:1,COMPANY.BOOK=GB0010001:1:1,CLS.
CPARTY=NO:1:1,OVERRIDE=INTRO/CUS*100 FROM 100966 NOT
RECEIVED:1:1,CURR.NO=1:1:1,INPUTTER=15_AUTHORISER___OFS_T
AABS:1:1,DATE.TIME=712131510:1:1,AUTHORISER=15_AUTHORISER
_OFS_TAABS:1:1,CO.CODE=GB0010001:1:1,DEPT.CODE=1:1:1
```

You can remove the entry from a field by setting a field value to NULL. Use this if you want to wipe out the entire field value. Do not use this for getting rid of multi values or sub values. You can remove a multi value or sub value by assigning a minus to it.

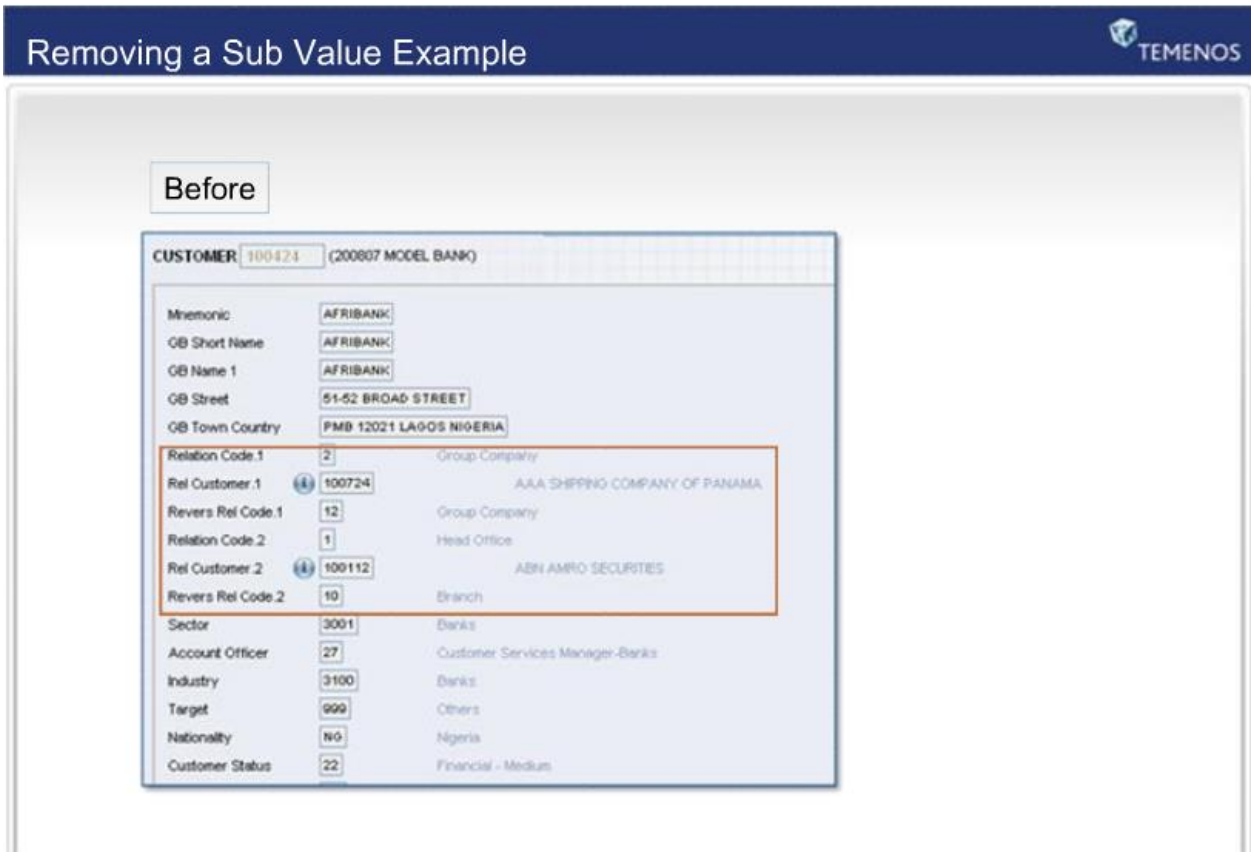
Feature 2 – Changing Field Values

- Null
- Removing subvalues

Note that ACCOUNT.TITLE.2 is null and therefore not displayed in the response.



We will now look at removing a sub-value. Let's illustrate this with an example. Look at the sample customer shown. This has two Relationships defined. We will remove one entry using an OFS request.



This example illustrates removing a multi-value field. You can see in the response that only the first multi-value of RELATION.CODE has been retained.

Removing a Multi Value Example

TEMENOS

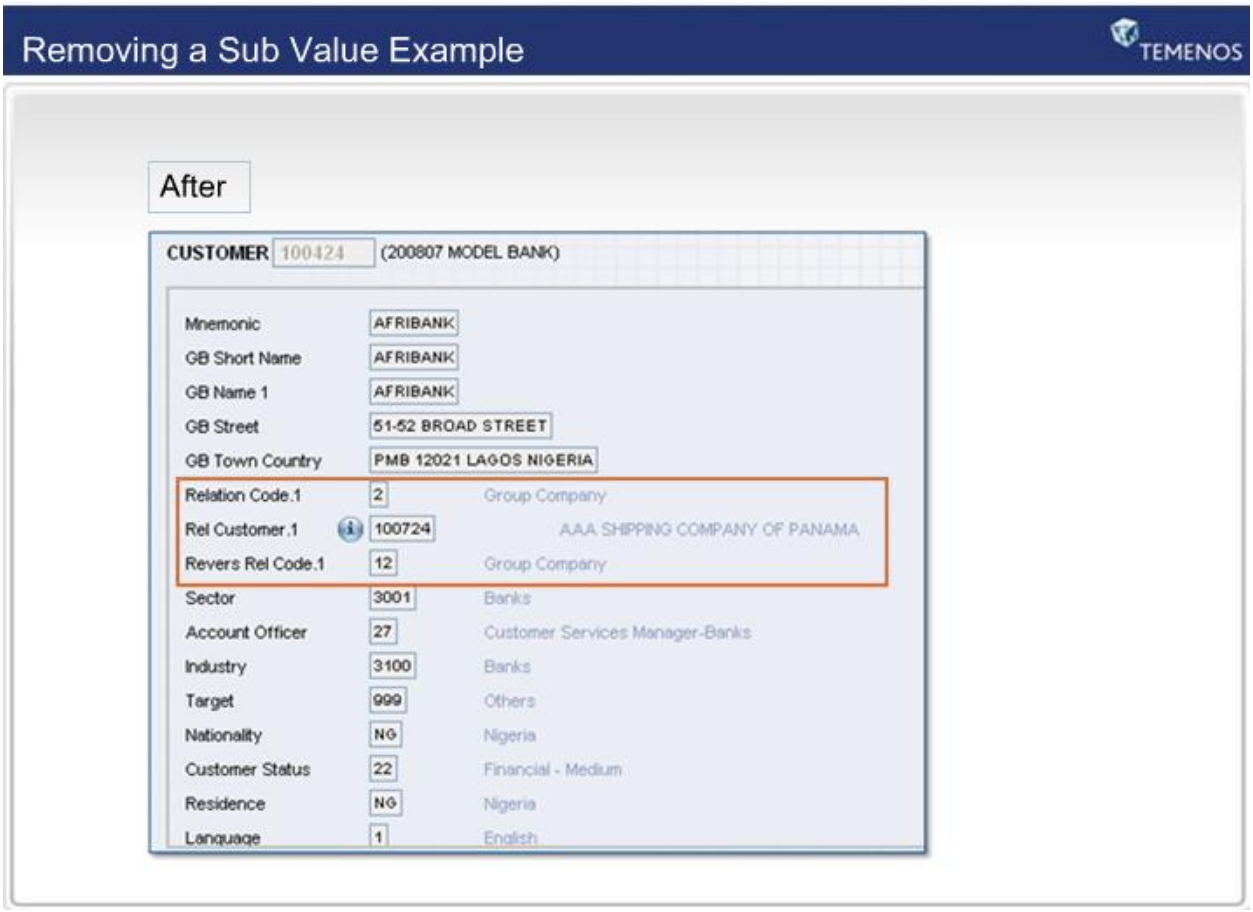
Request

```
CUSTOMER,/I/PROCESS//0,INPUTT/123123,100424,
RELATION.CODE:2:1=-
```

Response

```
100424//1,MNEMONIC:1:1=AFRIBANK,SHORT.NAME:1:1=AFRIBANK,N
AME.1:1:1=AFRIBANK,STRE
ET:1:1=51-52 BROAD STREET,TOWN.COUNTRY:1:1=PMB 12021
LAGOS NIGERIA,
RELATION.CODE:1:1=2,REL.CUSTOMER:1:1=100724,
REVERS.REL.CODE:1:1=12,SECTOR:1:1=3001,ACCOUNT.OF
FICER:1:1=27,INDUSTRY:1:1=3100,TARGET:1:1=999,NATIONALITY
:1:1=NG,CUSTOMER.STATUS:1:1=22,RESIDENCE:1:1=NG,LANGUAGE:
1:1=1, COMPANY.BOOK:1:1=GB0010001,CLS.CPARTY:1:1=NO,
CURR.NO:1:1=3,INPUTTER:1:1=4_INPUTTER___OFS_TEST.TELNET,
DATE.TIME:1:1=0811141247,AUTHORISER:1:1=4_INPUTTER_OFS_TE
ST.TELNET,CO.CODE:1:1=GB0010001,DEPT.CODE:1:1=1
```

Notice that only one relationship code is left.



What happens when you send an OFS message to modify an existing record? You would expect fields that you have mentioned your message to be altered. But what happens to data in other fields which have not been included in the message? Yes, those fields would not be altered. What happens then to multi-value fields? Well, the same principle applies. The specific multi-value alone would be altered.

From R7 there is a special option, that clears the existing data in the records and re-input data into the record. This option when used, clears existing multi-values also.

Feature 3 – Replace Option

- Amendments to existing records affect only the fields specified in the message
- From R7 there is a special option, that will allow records to be cleared and re-input.
- This option when used, clears existing multi-values also

Lets illustrate by using the CUSTOMER record created previously.

This has 2 multivalue's for the fields Relation.code,Rel.customer.

Relation.code.1=1

Rel.customer.1=100424

Relation.code.2=2

Rel.customer.2=100724

How do you change only the first multi-value?

Using a normal OFS message

How do you replace the entire multi-value set ?

Use an OFS message with replace option

Replace Option Example

TEMENOS

- Lets illustrate by using the CUSTOMER record created previously.
- This has 2 multivalue's for the fields Relation.code,Rel.customer.

  Relation.code.1=1

  Rel.customer.1=100424

  Relation.code.2=2

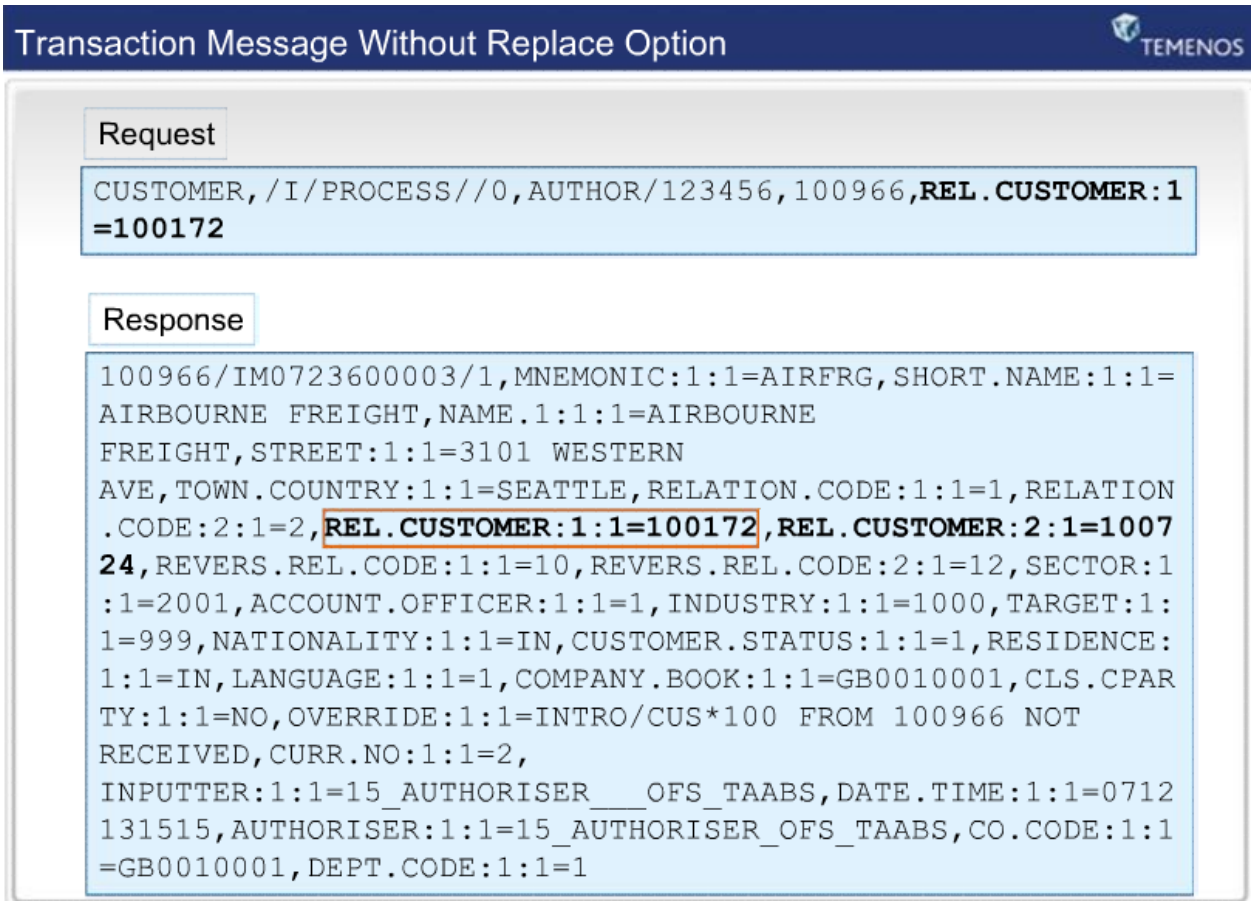
  Rel.customer.2=100724

- How do you change only the first multi-value?
- How do you replace the entire multi-value set ?

The request message given above with its response, illustrates what happens normally.

Only the specific multi-value is affected.

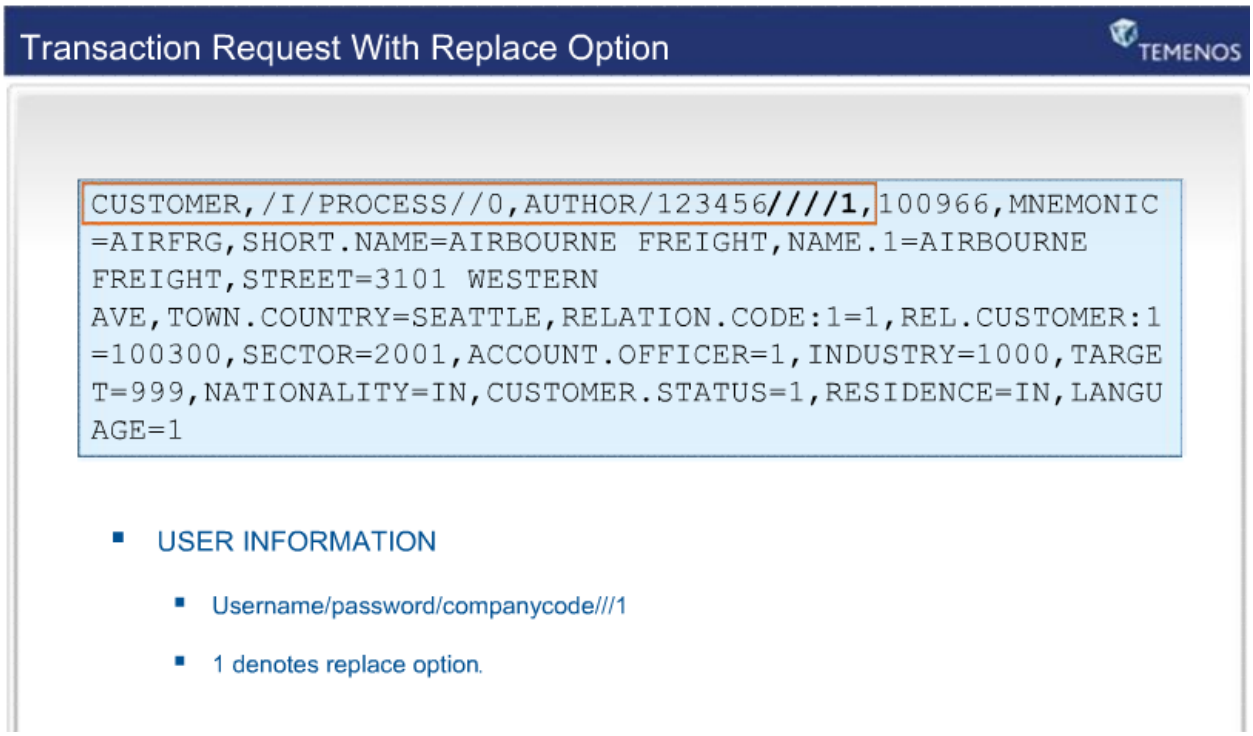
Other multi-values within the same field are not changed.



The replace option allows us to clear existing multi-values. This is specified as a number one at the end of the user information

Eg . Username/password/companycode///1

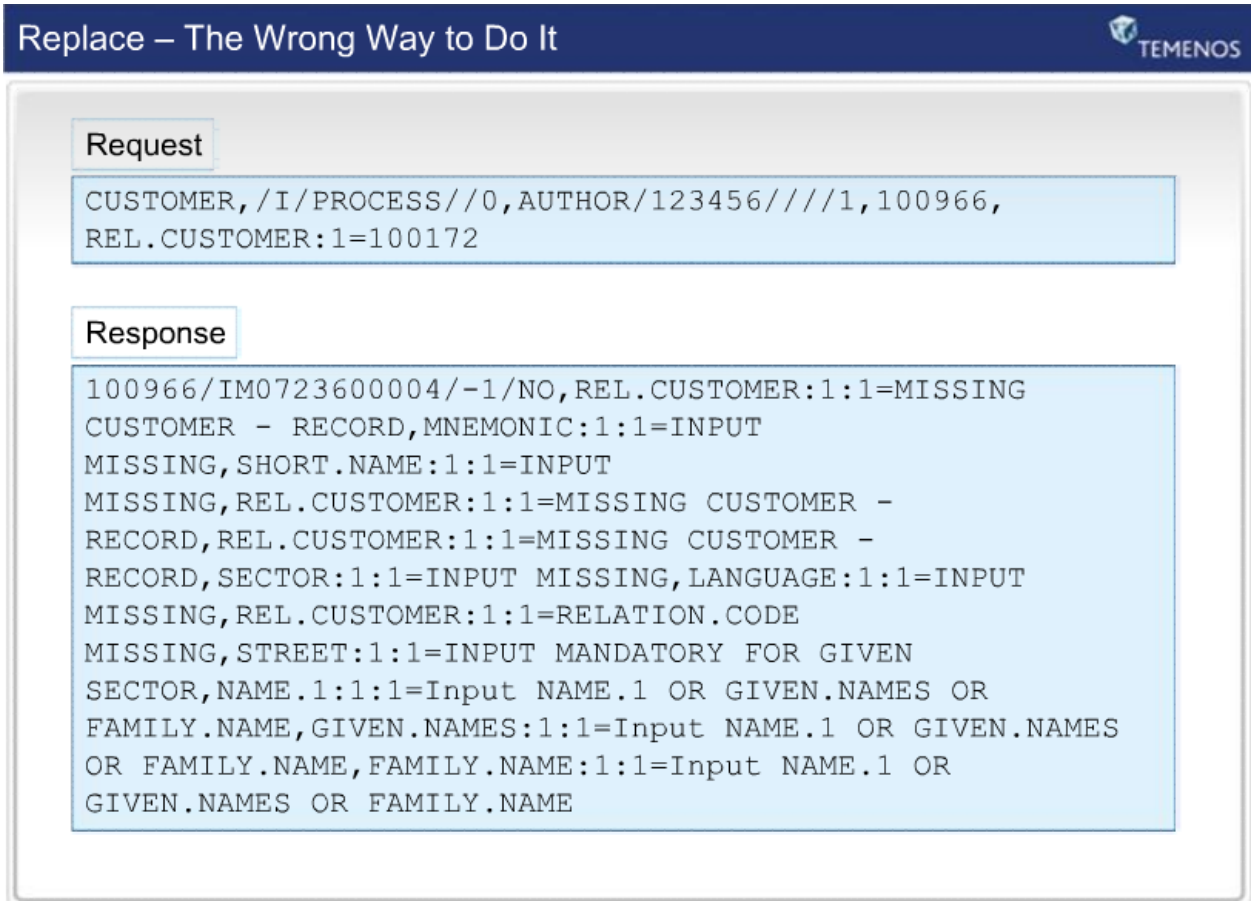
(Username followed by a slash, followed by password, followed by another slash, followed by the companycode, followed by three slashes and then a one) You can see an example in the slide. But why give values to all the fields?



The replace option allows us to clear existing multi-values. To understand this lets take at look at the above example. Normally (if replace had not been used) only the first multi-value position of RELATION.CODE would have been replaced by the new value. The other existing multi-values would have been left untouched. However, since the replace option has been specified, the record has been cleared, and ALL the fields replaced with the values specified in the OFS request.

Therefore, mandatory fields, though they already posses values, needed to be specified again in the OFS message.

The replace option replaces the entire record. Therefore ALL MANDATORY fields must be specified in the OFS message – even though the record has existing values. You can see from the response that OFS was unable to process the request.



We use token numbers to identify messages uniquely within a Browser environment. Similarly OFS Unique Identification numbers are used to identify each OFS message.

The checking for this OFS Unique Reference Number is done only if

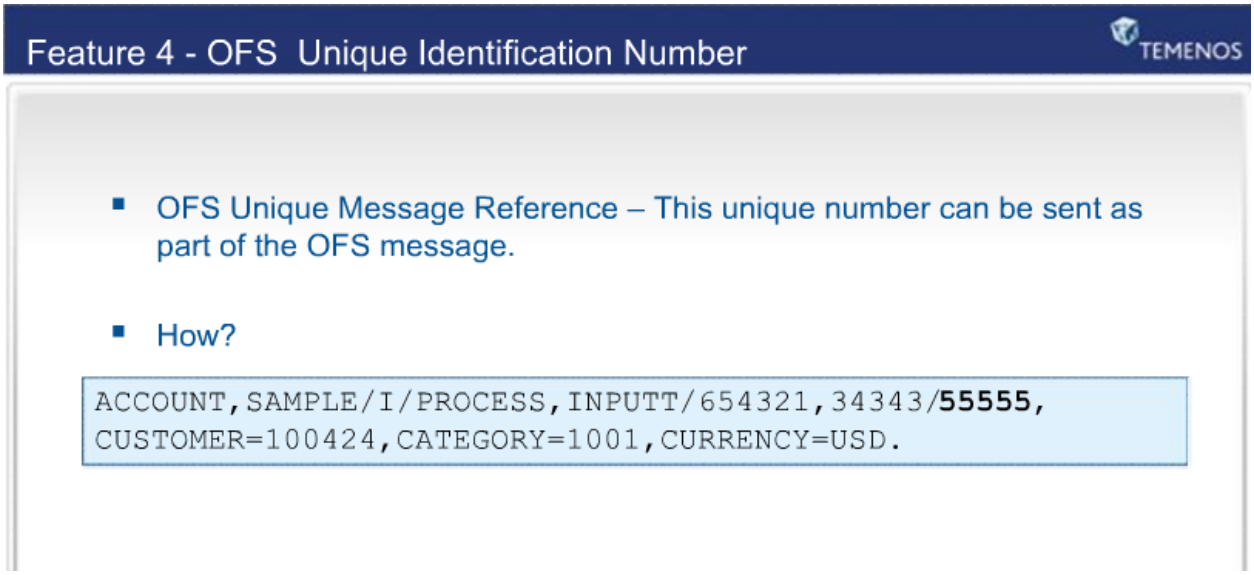
The word PROCESS is supplied as a part of the OFS message

The field ATTRIBUTES in OFS.SOURCE is left blank

How does T24 keep track of Unique Message References? T24 uses a file called F.OFS.UNIQUE.MSG.REF to keep track of Unique message references. When you send a message with a Unique Message Reference, a record with the supplied OFS Unique Message Reference as ID is created in the F.OFS.UNIQUE.MSG.REF file. This is an INT file and does not have a PGM.FILE entry. Hence it cannot be queried from within T24. The content of the record in F.OFS.UNIQUE.MSG.REF is as follows

Transaction ID/OFS Unique Message Reference/1,DUPLICATE.TRAP:1:1=TRUE

When an OFS message uses an existing OFS Unique Message Reference, the record content of the record pertaining to the OFS Unique Reference ID is fetched from the F.OFS.UNIQUE.MSG.REF file and displayed.



Some OFS messages may contain special characters as part of the data.

An address might contain a comma

Eg: 146, Sterling Road

An OFS request to create a version would use comma as part of the record id

Eg: A version MEMBER of CUSTOMER would have an id CUSTOMER, MEMBER

Since comma is used as a delimiter in an OFS request, including a comma in the data portion would convey a completely different meaning.

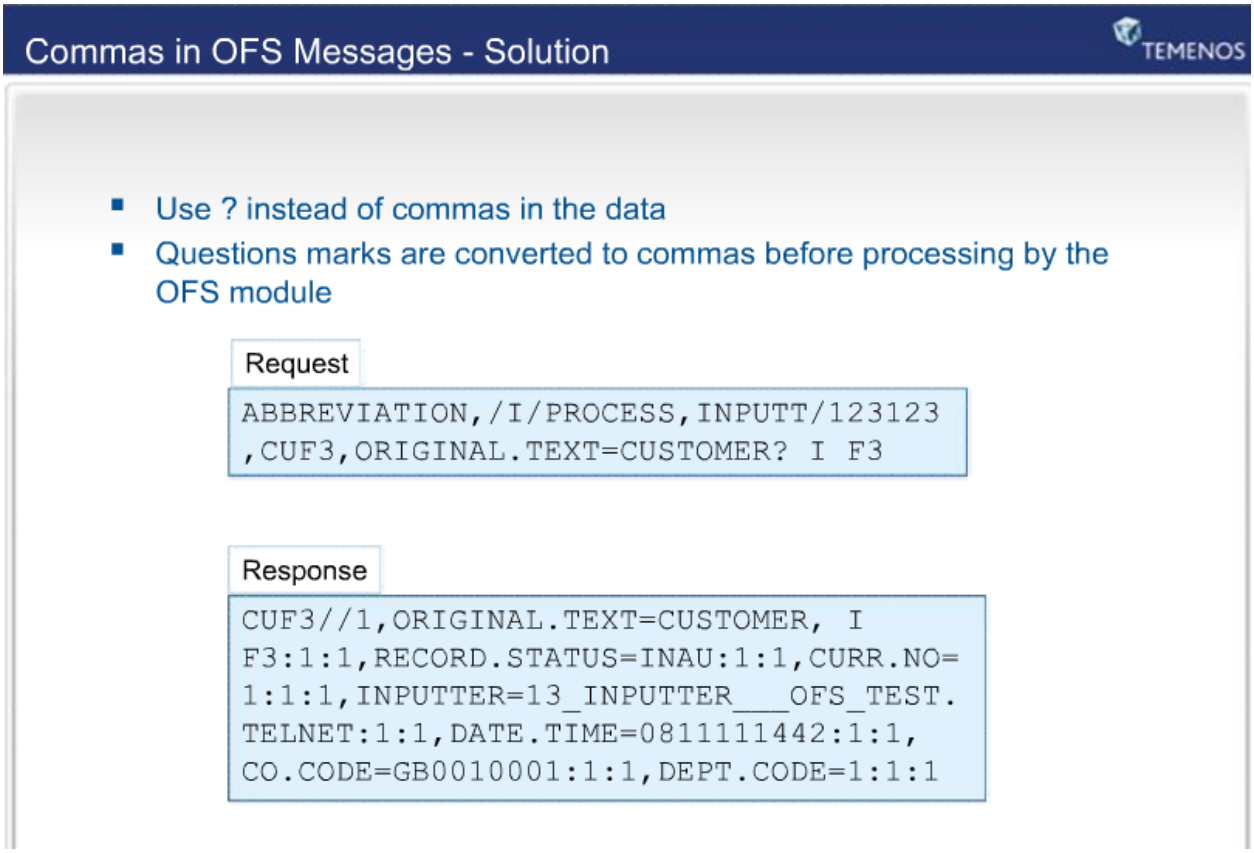
Feature 5 – Special Characters in OFS Messages

TEMENOS

- Some OFS messages may contain special characters as part of the data
  - An address might contain a comma
    - Eg: 146, Sterling Road
  - An OFS request to create a version would use comma as part of the record id
    - Eg: A version MEMBER of CUSTOMER would have an id CUSTOMER,MEMBER
- Comma in the data portion might convey a completely different meaning

Since comma is used as a delimiter in an OFS request, including a comma in the data portion would convey a completely different meaning. How do we get around this ? Well, use question marks instead of commas in the data. Questions marks are converted to commas before processing by the OFS module.

Look at the example shown. The value assigned to the field ORIGINAL.TEXT has a question mark in place of a comma. This has been substituted by a comma, as you can see from the response.



If a ‘,’ (comma) character is required within the field content then it should be replaced by the ‘?’ (Question mark) character.

If a single quote (‘) character is required within the field content then it should be replaced by the ‘@’ character.

If a double quote (") character is required within the field content then it should be replaced by the ‘|’ (pipe line) character.

If space is required within the field content then it should be replaced by the ‘~’ (tilt symbol) character.

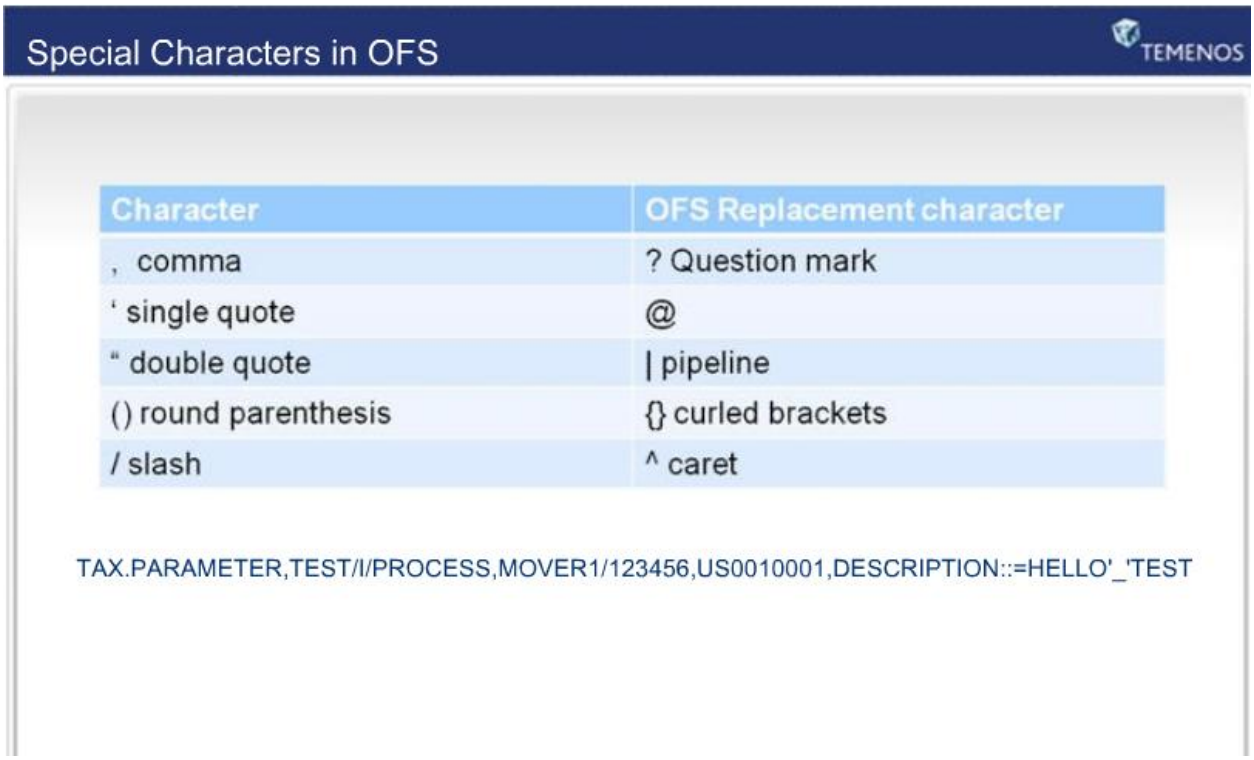
If ‘()’ is required within the field content then it should be replaced by the ‘{}’ character.

If ‘ / ’ is required within the field content then it should be replaced by the ‘^’ character.

To input an "Underscore __ " in a OFS String (Eg Kripesh_K@yahoo.com) in one of your

Fields (For Eg., Email.ID) of an Application you can do so by passing the value like this....

TAX.PARAMETER,TEST/I/PROCESS,MOVER1/123456,US0010001,DESCRIPTION::=HELLO'_'TEST



Modes in Detail

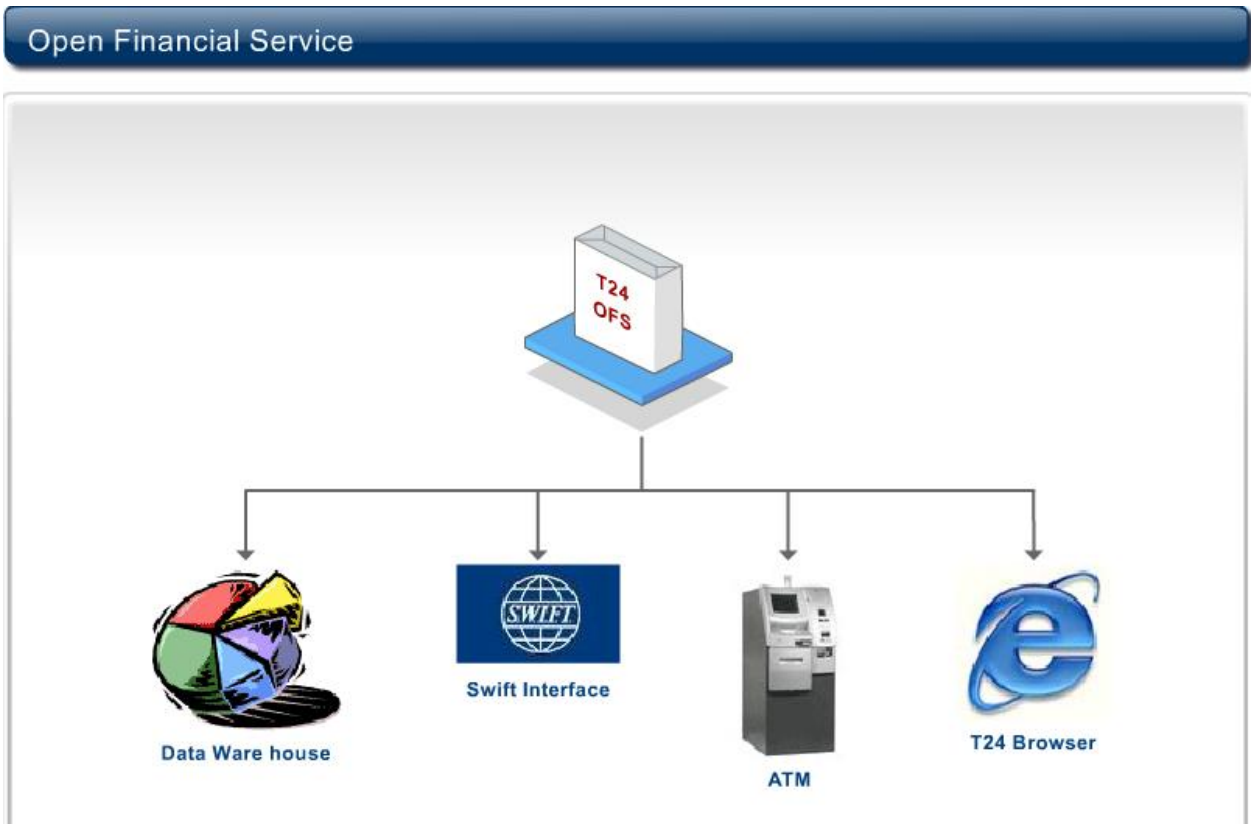
Who sends messages to OFS?

Third party systems may send messages online.

Third party systems may send more than one message in an offline mode i.e. in a batch.

A T24 application itself may spawn an OFS message.

Browser sends messages.



OFS can be used in three different ways with four different modes. Shall we see what they are?

1) The first way we can use OFS is in Batch processing. Batch processing, as the name implies, is used for handling multiple messages which may be processed offline. This is useful for handling offline requests from third party systems. The mode used is obviously BATCH.

2) The second way that we use OFS is Inter-application processing. Inter-application processing is used for calling one application from another. For example, this is used in user definable routines such as version routines. The mode used is GLOBUS.

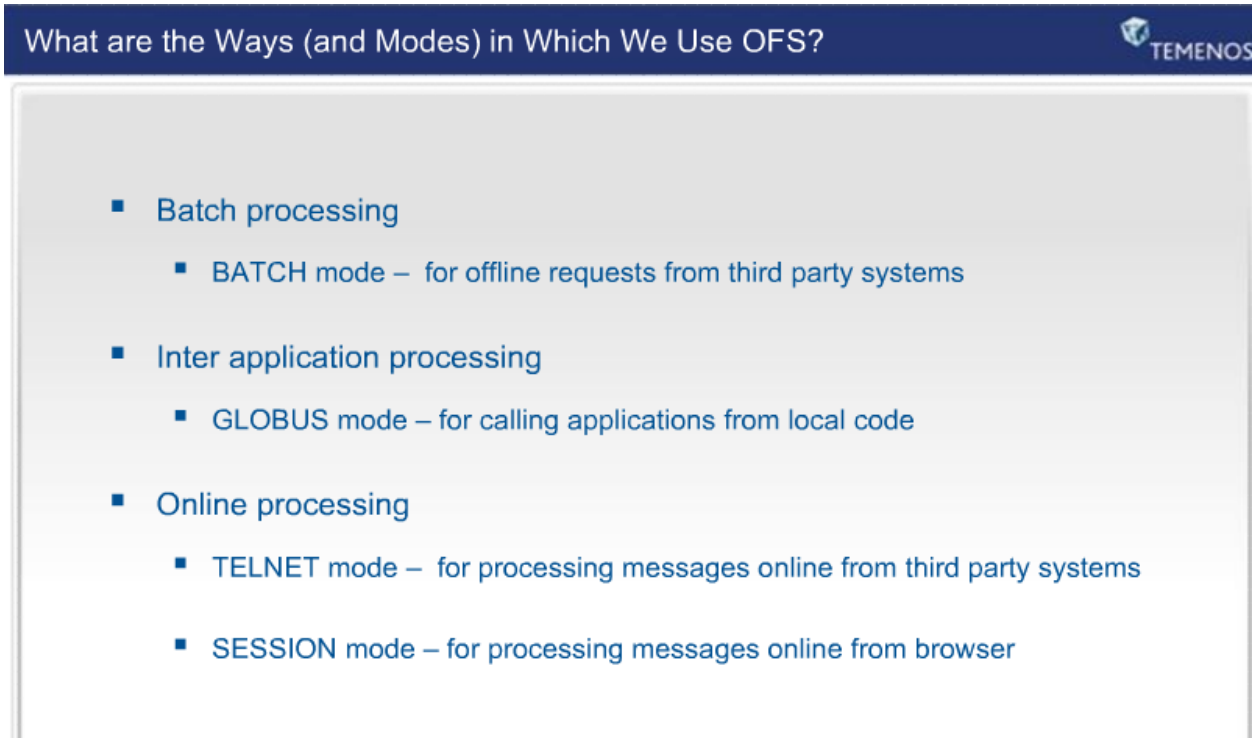
3) The last way in which we can use OFS is online. Online processing signifies that is a request is sent, processed immediately and the response sent back to the requestor.

There are two types or modes of online processing:

a) An external system or user connects directly to OFS or to OFS through TCS. The mode used in this case is TELNET.

b) A user connects to the T24 system using Browser. In this case, a special mode called SESSION is used.

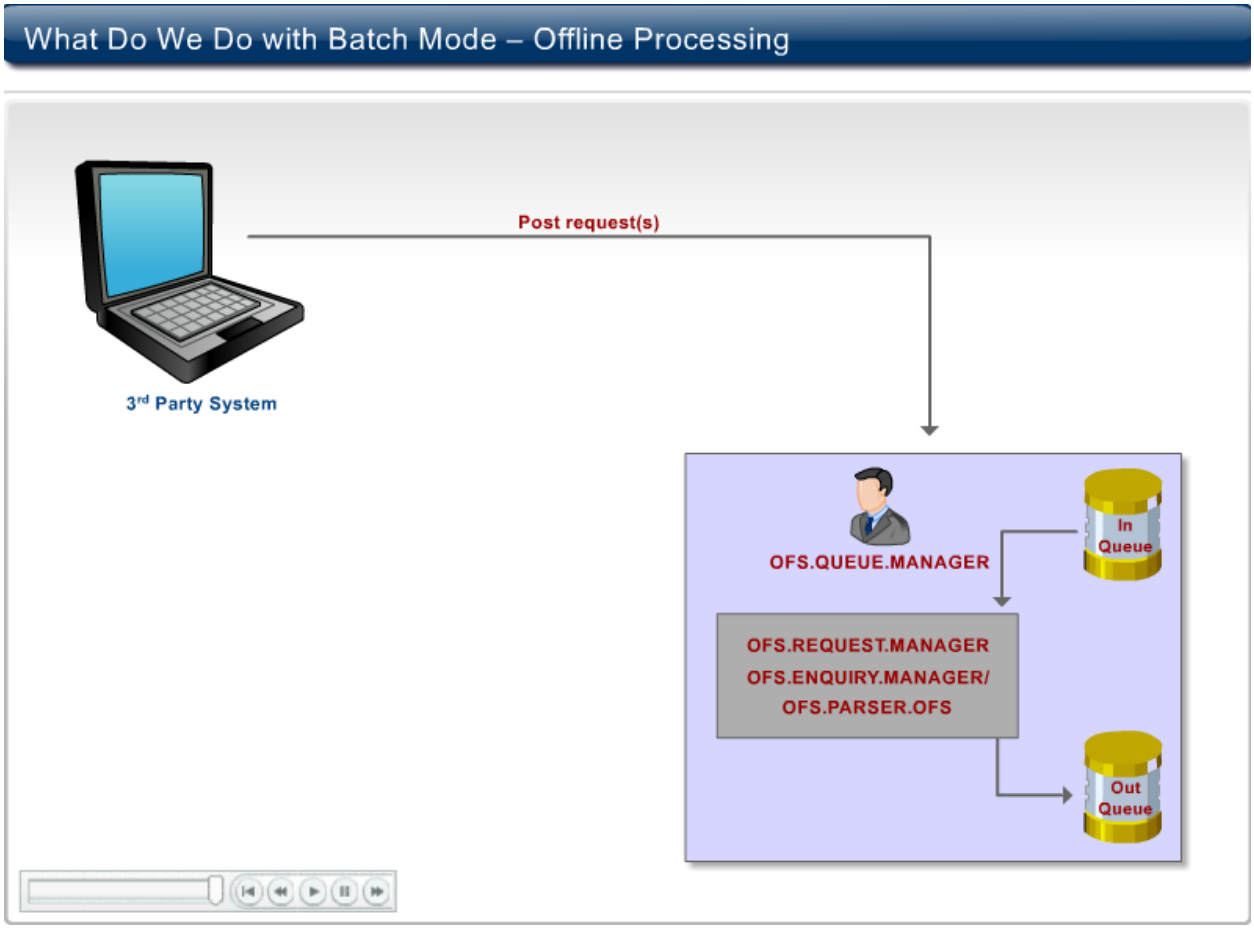
You will study each of these modes in detail.



Let us try to understand what happens in the OFS batch mode. In the OFS Batch mode, a T24 routine called OFS.QUEUE.MANAGER runs as a phantom process. This process picks up messages from a queue, and calls another T24 routine called the OFS.REQUEST.MANAGER for processing the request. OFS.REQUEST.MANAGER passes the message to a parser routine called OFS.PARSER.OFS which validates it. Validated messages are processed by OFS.REQUEST.MANAGER.

The Request manager will call the OFS.ENQUIRY.MANAGER, if the request is an enquiry request, to run an enquiry inside TEMENOS T24. The response is Then sent back.

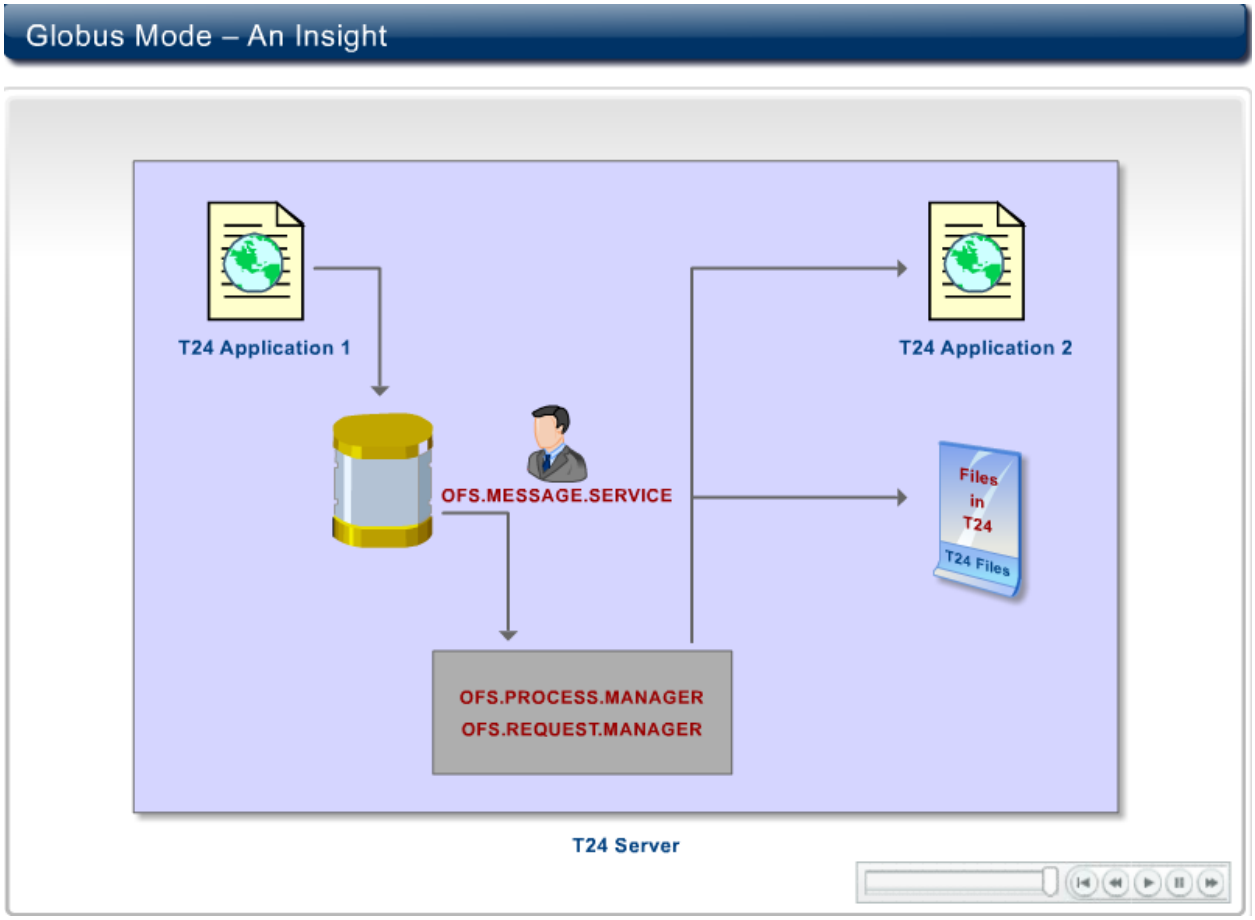
OFS batch mode can look into a directory containing various messages as files or handle a single file containing messages. All this is controlled by setting up an OFS.SOURCE record and EB.PHANTOM record, which you will learn as you proceed.



How does the Globus Mode work?

Well, all applications internally use a routine named OFS.POST.MESSAGE to post requests into the queue. Yes, you actually have to CALL this routine in your code. This routine writes data into a queue named F.OFS.MESSAGE.QUEUE.

What happens to the messages in the queue then? A service named OFS.MESSAGE.SERVICE picks up the request from the queue and sends the request to OFS.PROCESS.MANAGER to process the requests. OFS.PROCESS.MANAGER then calls OFS.REQUEST.MANAGER, the request is processed and the relevant files and applications in T24 are updated.



How did the Telnet mode work?

When the telnet connection was established, this launched a program called EB.AUTO.INIT.PROCESS. Yes, this program was called from the dot profile. This program then did some checks to establish that an online connection had been requested and called a routine named OFS.START.ONLINE.COMMS.

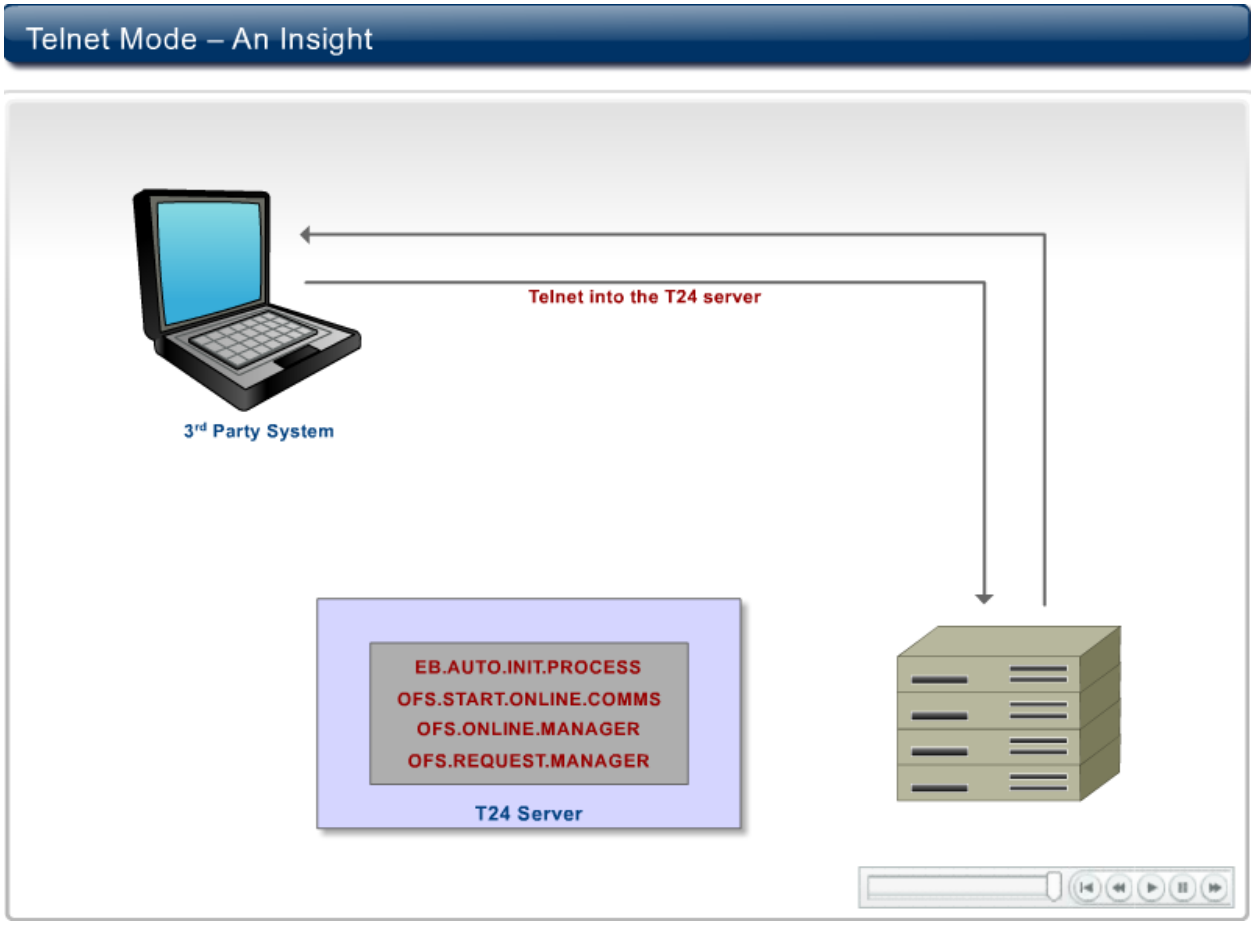
This in turn started a phantom process called the OFS.ONLINE.MANAGER. The online manager then listened for a message on the open connection. When a request was received, OFS.ONLINE.MANAGER internally called OFS.REQUEST.MANAGER to process the request.

Online manager was a single threaded process and did not have connection managing capabilities. For instance, it could not accept a second message, while the first has been processed. Therefore, you had to setup multiple connections to process multiple messages simultaneously.

You could however control the maximum number of open OFS connections. Each OFS connection would continue to be active until either shutdown from TEMENOS T24 or by the external application. The connection could also remain active during the offline mode to allow enquiry functions to take place.

The direct connection to OFS is obsolete with the advent of Temenos Connector Server.

Note: TC Server will not be covered as part of this session, but as part of the TC Server session.



Let us look at what happens when TCS receives a message.

TCS 'opens' a connection to T24. How does it do this? It spawns a program called tSS (that's the short form for t24 Server Session). This program by the way used to be called OFS.CONNECTION.MANAGER in older T24 releases. tSS receives the request and in turn passes it to OFS.BULK.MANAGER. This decides whether the message is part of a bulk message or a single message. The OFS.BULK.MANAGER then calls the OFS.PROCESS.MANAGER.

The OFS.PROCESS.MANAGER accepts the request and decides how to handle it appropriately. That is, it calls OFS.SESSION.MANAGER for standard OFS messages. Delivery messages are handled separately by calling a routine named OFS.DE.REQUEST. The OPM also checks if the OFS.SOURCE record exists and that the request is not an empty one.

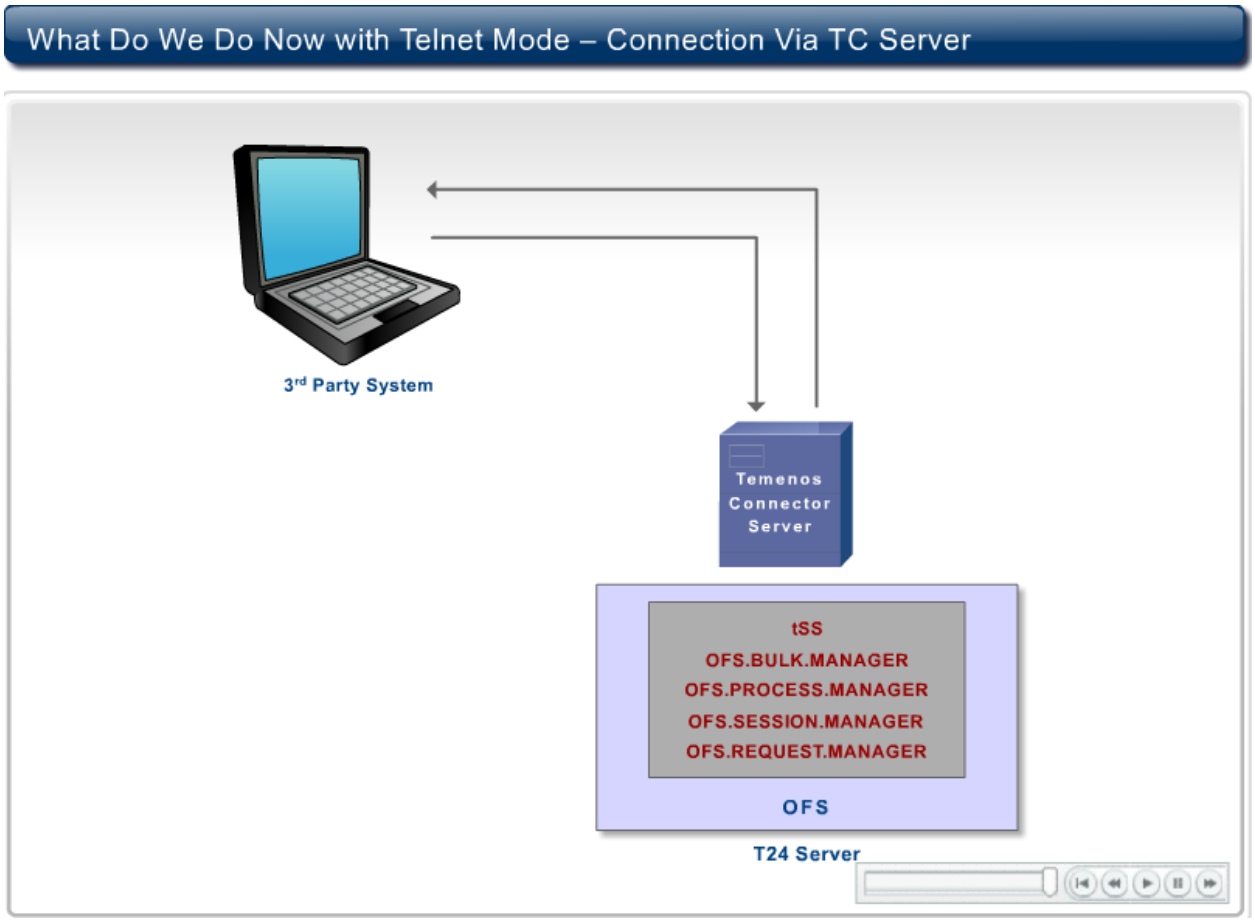
OFS.SESSION.MANAGER controls the login and token handling, as well as fronting.

OFS.REQUEST.MANAGER doing the repetitive tasks once only. It eventually calls.

OFS.REQUEST.MANAGER for processing the request.

OFS.REQUEST.MANAGER invokes the parser routine OFS.PARSER.OFS to validate the message.

Validated messages are processed by OFS.REQUEST.MANAGER.



Messages from browser are sent to OFS via the Temenos Connector Server (TCS). These messages are treated the same as messages from a third party system that you saw earlier.

TCS then spawns one or more tSS sessions. These tSS sessions are background processes that receive the request and in turn calls a new OFS routine named OFS.BULK.MANAGER. This decides whether the message is part of a bulk message or single message. The OFS.BULK.MANAGER then calls the OFS.PROCESS.MANAGER.

The OFS.PROCESS.MANAGER accepts the request and handles it appropriately by calling OFS.SESSION.MANAGER.

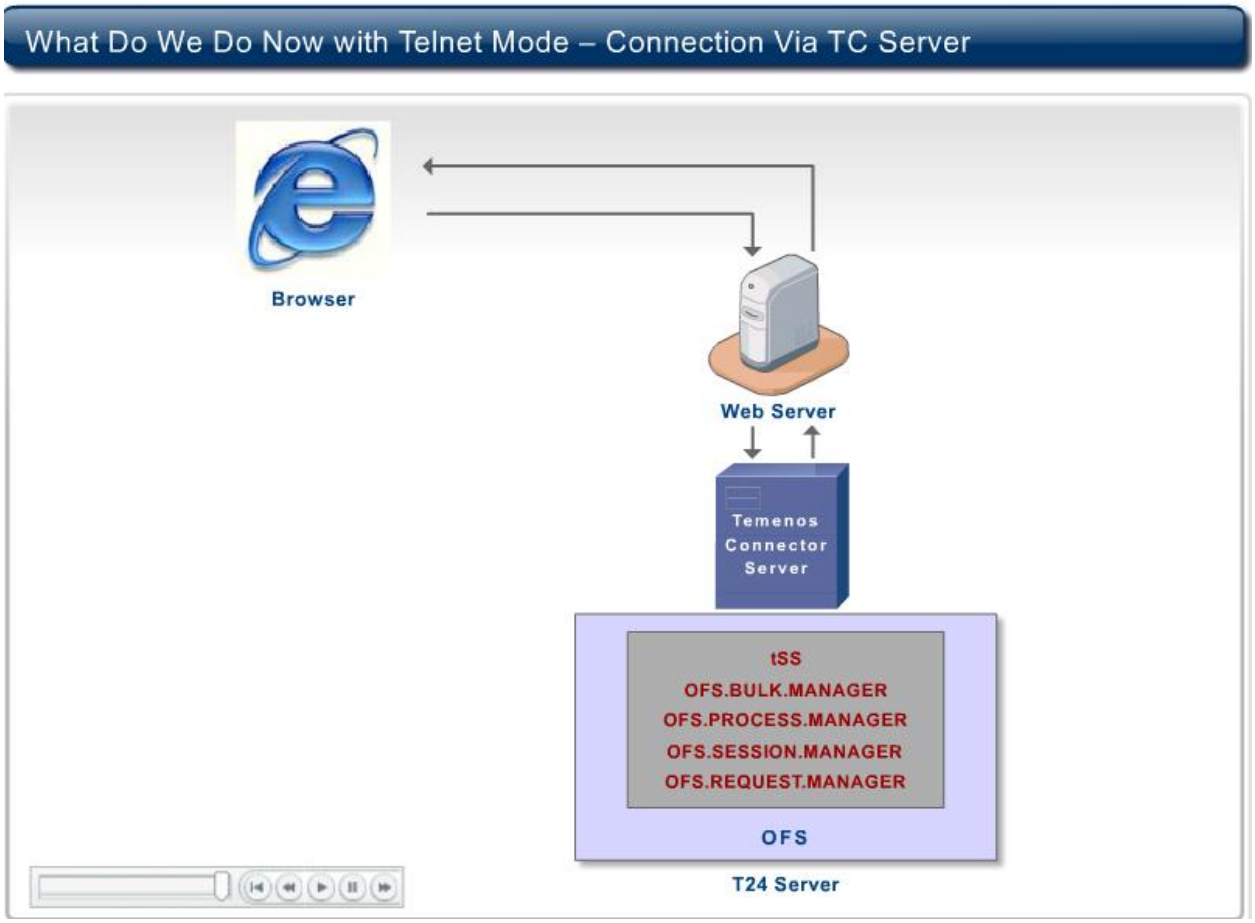
OFS.SESSION.MANAGER controls the login and token handling, as well as fronting.

OFS.REQUEST.MANAGER, doing the repetitive tasks once only. It eventually calls.

OFS.REQUEST.MANAGER for processing the request.

OFS.REQUEST.MANAGER invokes the parser routine OFS.PARSER.XML to validate the message.

Validated messages are processed by OFS.REQUEST.MANAGER.



Routines and Their Functions

TEMENOS

Routine	Function	Mode
OFS.QUEUE.MANAGER	Phantom that picks request from the IN QUEUE	Batch
OFS.REQUEST.MANAGER	Processes the message	ALL
OFS.PARSER.OFS	Validation of the message (For browser requests, it is OFS.PARSER.XML)	ALL
OFS.ENQUIRY.MANAGER	REQUEST MANAGER calls ENQUIRY MANAGER to run an enquiry if the request is an enquiry request	ALL
EB.PHANTOM.PH	This is the actual process that runs in the background	Batch

Routines and Their Functions

TEMENOS

Routine	Function	Mode
OFS.PROCESS.MANAGER	Called from BULK MANAGER for messages that are standard messages(Not BULK messages). Deals with handling of the message.	Online
OFS.ONLINE.MANAGER	Listens to a request and Calls OFS.REQUEST.MANAGER	TELNET (Obsolete)
OFS.BULK.MANAGER	Called when bulk OFS requests are sent. Incorporates Transaction Block. Any message that come in via online mode will pass through BULK MANAGER	Online
OFS.SESSION.MANAGER	Called by OFS.PROCESS.MANAGER when the request is from the Browser. Also handles token management.	Online

Routines and Their Functions

TEMENOS

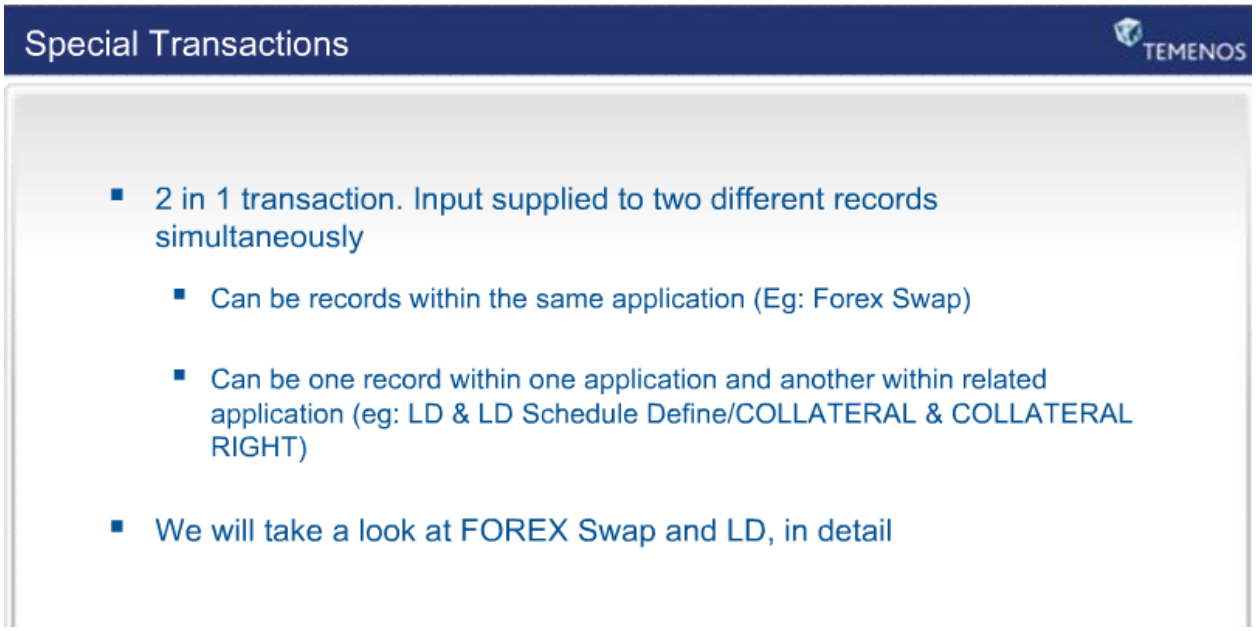
Routine	Function	Mode
EB.PHANTOM.RUN	Called by EB.PHANTOM. This routine in turn calls EB.PHANTOM.PH	Batch
OFS.POST.MESSAGE	Posts requests in the IN QUEUE	Globus
OFS.MESSAGE.SERVICE	Picks up the requests from the IN QUEUE	Globus
EB.AUTO.INIT.PROCESS	Program that is launched and does some initial checks for online requests	TELNET (Obsolete)
OFS.START.ONLINE.COMMS	Called by EB.AUTO.INIT.PROCESS and in turn calls OFS.ONLINE.MANAGER	TELNET (Obsolete)
OFS.DE.REQUEST	Called by OFS.PROCESS.MANAGER for delivery messages	ONLINE

Special OFS messages – Forex and LD

We will now look at two rather odd OFS messages. Wondering why we term them odd?.

Well, so far any OFS message that you saw had data to update a single record only. The special OFS transactions are messages which actually supply input to two different records simultaneously.

These can be records within the same application (Eg: Forex Swap deals). These can be one record within one application (e.g. LD.LOAN.AND.DEPOSITS) and another within a related application (e.g. LD.SCHEDULE.DEFINE).Another example is COLLATERAL.RIGHT and COLLATERAL.



Swap deals involve the exchange of currencies at the spot rate with an agreement to reverse the transaction with an identical amount of currency at a later date at a specified rate.

Let's try to understand this with an example.

Bank Alpha strikes a swap deal with Bank of New York. Under this deal, Bank Alpha buys 3,680,000 USD from Bank of New York for 2,000,000 GBP at a rate of 1.84. It agrees that it will buy back the 2,000,000GBP 10 days later from Bank of New York at a rate of 1.86 (i.e. it will pay 3,720,000 USD).

Look at the sample shown. It contains the details needed for the first leg of the deal such as the counter party (eg: 100472) , the currency bought (eg: USD), spot rate (eg: 1.84), amount bought (eg: 3,680,000) , value date (eg: 10 Jan 2008), the currency sold in return (eg: GBP). On committing this, T24 would automatically bring up the input page for the second leg of the deal.

Swap Deal Example – 1st Leg

Key fields with sample data	
Fields	**Sample data**
Deal type	SW
Counterparty	100472
Currency bought	USD
Amount Bought	3680000
Value Date Buy	20080110
Currency sold	GBP
Spot rate	1.84
Transaction type	SW

Notes correct to See the details required for the second leg of the deal. The ones you need to specify are the Forward rate (eg: 1.86) and the value date buy (21 Jan 2008). The amount sold and forward rate fields are mutually exclusive. Enter values in one of them and T24 would fill out the other.

Swap Deal Example – 2nd Leg

Key fields with sample data

Fields	Sample data
Deal type	SW
Counterparty	100472
Currency bought	GBP
Amount Bought (no input)	2000000
Value Date Buy	20080121
Currency sold	USD
Forward rate	1.86
Transaction type	SW

You should keep the following points while preparing the OFS message for a Forex swap.

Foreign exchange swaps require that information be provided for both legs of a swap within a single message.

The information for the first leg is separated from information for the second leg by an underscore (_).This applies to the multi value number, sub value number, and field data parts of the message.

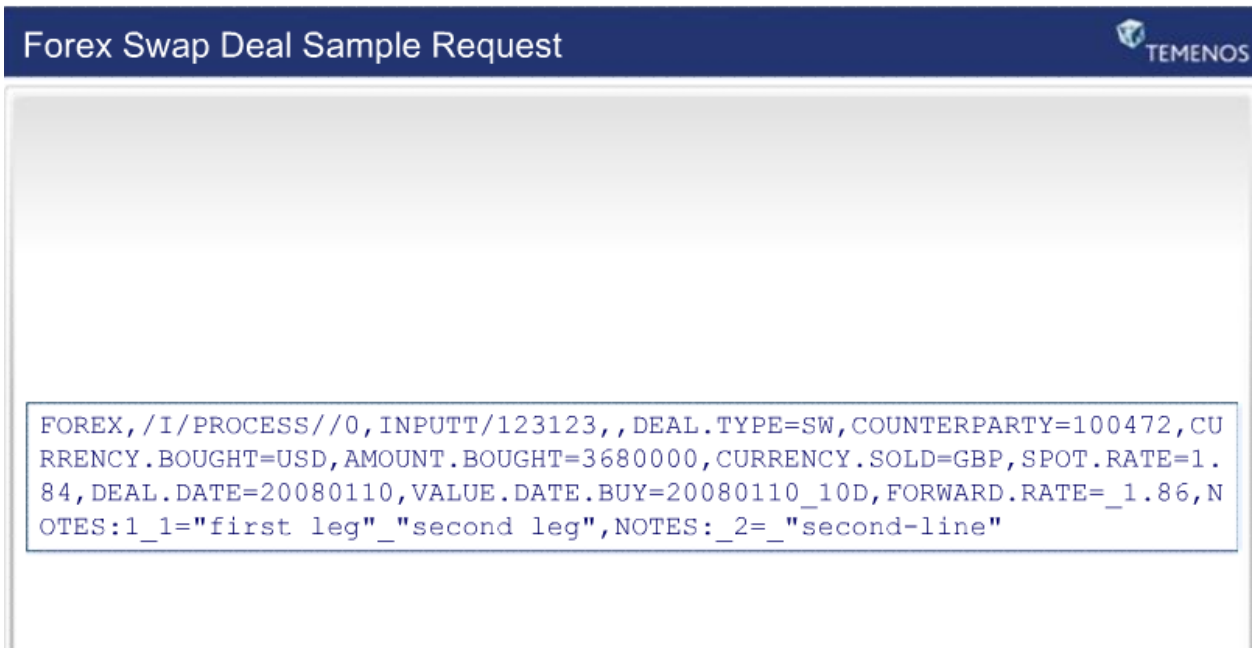
OFS & Forex Swaps – Things That Matter

- Information for both legs supplied *within a single message*
- Information for the first leg is separated from information for the second leg by an underscore (_)
- This applies to the multi value number, sub value number, and field data parts of the message

You can see the corresponding OFS message for the swap deal discussed earlier. The data appearing after the underscore indicates that it is meant for the second leg of the deal. For instance

VALUE.DATE.BUY=2008010_10D indicates that the value date for the second deal is 10 days from the value date of the first.

FORWARD.RATE does not apply for the first leg of the deal, and therefore you can see that the value has been specified as _1.86. Similarly if you observe the NOTES field you will notice that since it is a multi-value field, the multi-value positions next to the field have two entries separated by underscore (eg: NOTES:1_1) ,indicating the multi-value position in both parts of the deal.

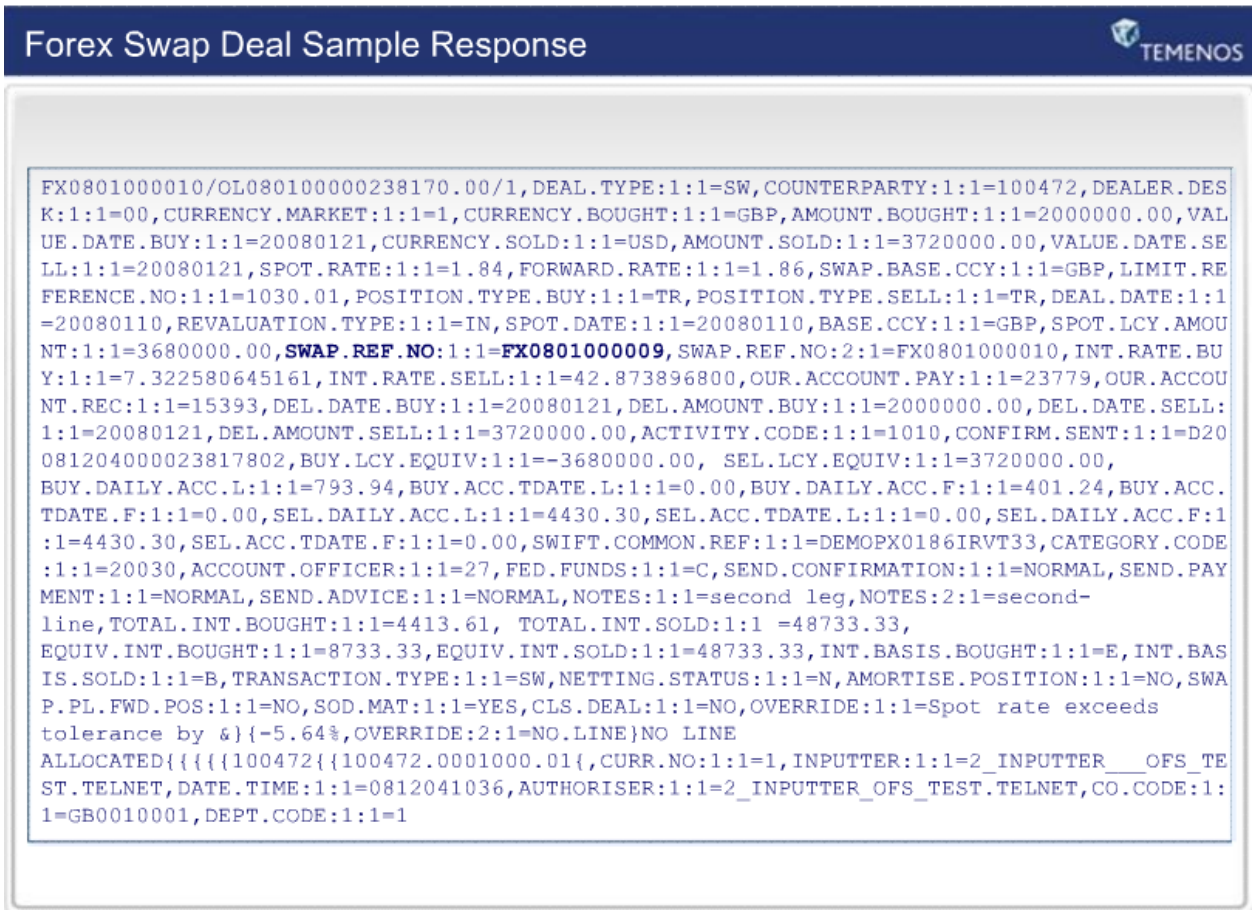


Shown here is the response message for the swap deal.

In the example, the reply contains only the second leg. This is because, technically it is the last transaction (last record update) performed by the OFS though there was only one request.

In the response message, notice that the field SWAP.REF.NO contains the reference (transaction ID) of the first leg, thus indicating the first leg transaction was also successful.

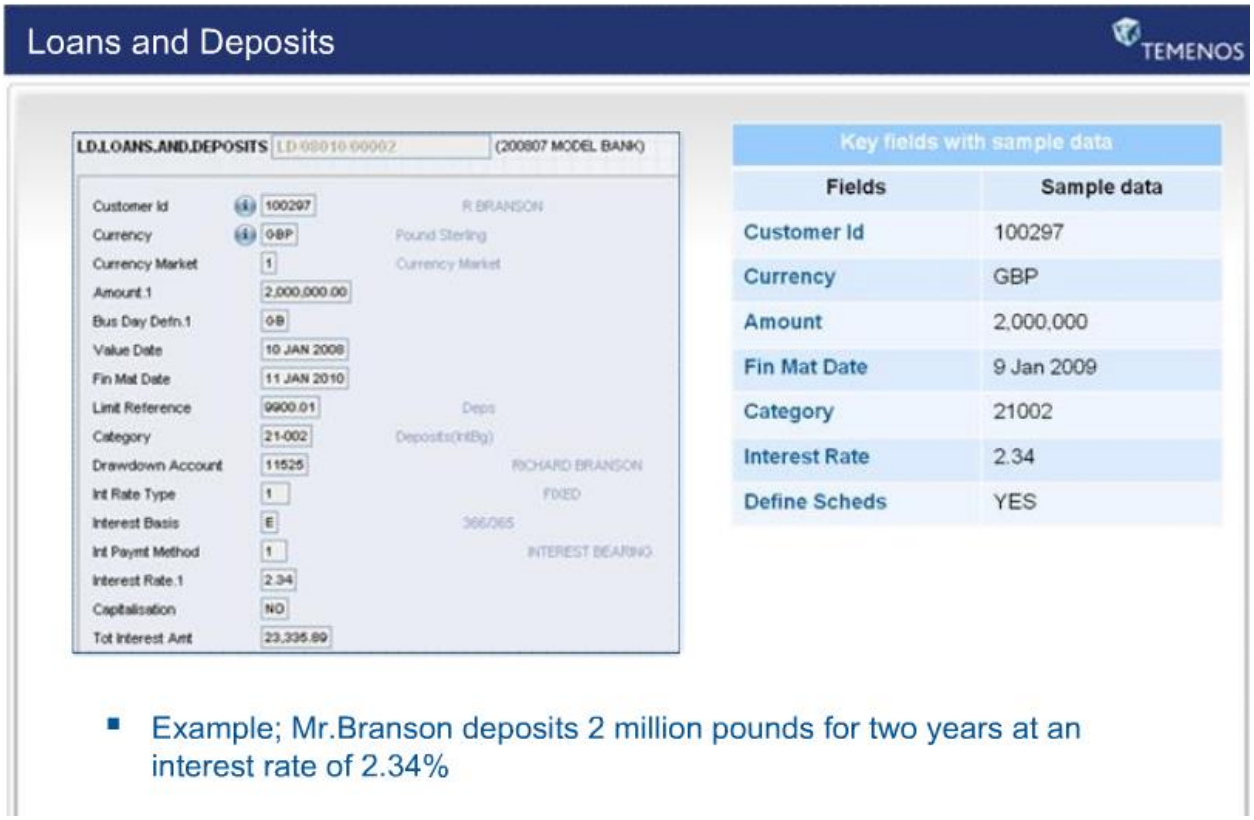
Forex Swap Deal Sample Response



```
FX0801000010/OL080100000238170.00/1,DEAL.TYPE:1:1=SW,COUNTERPARTY:1:1=100472,DEALER.DES
K:1:1=00,CURRENCY.MARKET:1:1=1,CURRENCY.BOUGHT:1:1=GBP,AMOUNT.BOUGHT:1:1=2000000.00,VAL
UE.DATE.BUY:1:1=20080121,CURRENCY.SOLD:1:1=USD,AMOUNT.SOLD:1:1=3720000.00,VALUE.DATE.SE
LL:1:1=20080121,SPOT.RATE:1:1=1.84,FORWARD.RATE:1:1=1.86,SWAP.BASE.CCY:1:1=GBP,LIMIT.RE
FERENCE.NO:1:1=1030.01,POSITION.TYPE.BUY:1:1=TR,POSITION.TYPE.SELL:1:1=TR,DEAL.DATE:1:1
=20080110,REVALUATION.TYPE:1:1=IN,SPOT.DATE:1:1=20080110,BASE.CCY:1:1=GBP,SPOT.LCY.AMOU
NT:1:1=3680000.00,SWAP.REF.NO:1:1=FX0801000009,SWAP.REF.NO:2:1=FX0801000010,INT.RATE.BU
Y:1:1=7.322580645161,INT.RATE.SELL:1:1=42.873896800,OUR.ACCOUNT.PAY:1:1=23779,OUR.ACCOU
NT.REC:1:1=15393,DEL.DATE.BUY:1:1=20080121,DEL.AMOUNT.BUY:1:1=2000000.00,DEL.DATE.SELL:
1:1=20080121,DEL.AMOUNT.SELL:1:1=3720000.00,ACTIVITY.CODE:1:1=1010,CONFIRM.SENT:1:1=D20
081204000023817802,BUY.LCY.EQUIV:1:1=-3680000.00, SEL.LCY.EQUIV:1:1=3720000.00,
BUY.DAILY.ACC.L:1:1=793.94,BUY.ACC.TDATE.L:1:1=0.00,BUY.DAILY.ACC.F:1:1=401.24,BUY.ACC.
TDATE.F:1:1=0.00,SEL.DAILY.ACC.L:1:1=4430.30,SEL.ACC.TDATE.L:1:1=0.00,SEL.DAILY.ACC.F:1
:1=4430.30,SEL.ACC.TDATE.F:1:1=0.00,SWIFT.COMMON.REF:1:1=DEMOPX0186IRVT33,CATEGORY.CODE
:1:1=20030,ACCOUNT.OFFICER:1:1=27,FED.FUNDS:1:1=C,SEND.CONFIRMATION:1:1=NORMAL,SEND.PAY
MENT:1:1=NORMAL,SEND.ADVICE:1:1=NORMAL,NOTES:1:1=second leg,NOTES:2:1=second-
line,TOTAL.INT.BOUGHT:1:1=4413.61, TOTAL.INT.SOLD:1:1 =48733.33,
EQUIV.INT.BOUGHT:1:1=8733.33,EQUIV.INT.SOLD:1:1=48733.33,INT.BASIS.BOUGHT:1:1=E,INT.BAS
IS.SOLD:1:1=B,TRANSACTION.TYPE:1:1=SW,NETTING.STATUS:1:1=N,AMORTISE.POSITION:1:1=NO,SWA
P.PL.FWD.POS:1:1=NO,SOD.MAT:1:1=YES,CLS.DEAL:1:1=NO,OVERRIDE:1:1=Spot rate exceeds
tolerance by &}{-5.64%,OVERRIDE:2:1=NO.LINE}NO LINE
ALLOCATED{{{{{100472{{100472.0001000.01{,CURR.NO:1:1=1,INPUTTER:1:1=2_INPUTTER___OFS_TE
ST.TELNET,DATE.TIME:1:1=0812041036,AUTHORISER:1:1=2_INPUTTER_OFS_TEST.TELNET,CO.CODE:1:
1=GB0010001,DEPT.CODE:1:1=1
```

The next special transaction we will look at is Loans and Deposits. Let's understand this with an example. Mr.Branson deposits an amount of 2 million pounds in the bank. Been a valued customer, he specifies his own schedule for the interest payments that he would receive. The payment schedule is specified in a linked application called LD.SCHEDULE.DEFINE.

Normally the payment schedule (or the repayment schedule in the case of loans) is generated automatically by T24. The LD application automatically brings up the page to input a schedule if the field Define Scheds has been given a value of YES.



The LD application automatically brings up the page to input a schedule in LD.SCHEDULE.DEFINE if the field Define Scheds has been given a value of YES.

You can see a sample schedule above. Let's look at each field.

Forward Backward key : This field indicates the method which will be followed for the generation of schedules and the action that will be taken if the derived date is a non working day. This must be given a value of 4 in case of schedules which are not automatic, i.e. user defined schedules.

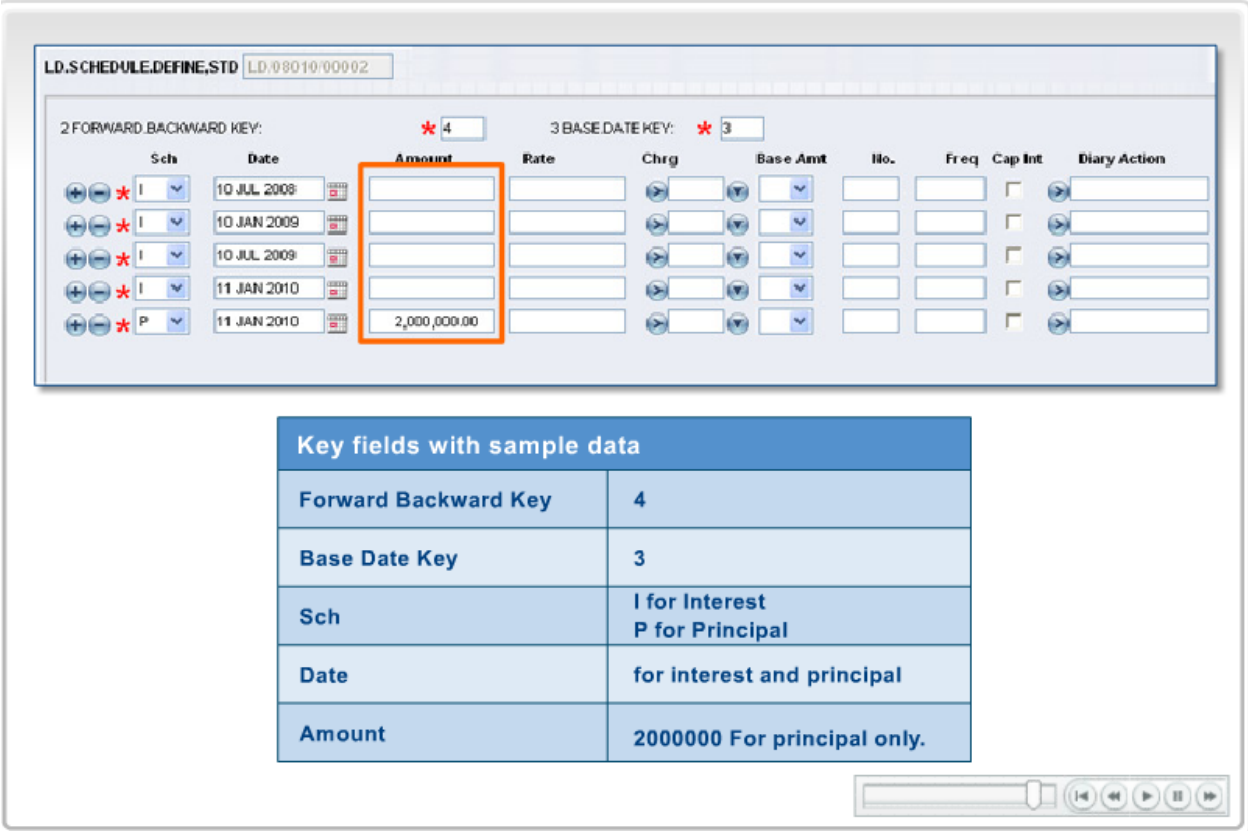
Base Date key : Must be given a value of 3 to indicate that the dates will be supplied by the user.

Schedule Type : Enter I for Interest and P for Principal.

Date : Indicates the date of the event, i.e. payment of interest or principal , repayment date etc.

Amount : This furnishes the financial value to be processed on the given date. We have given 2 million to indicate that the principal must be paid back to the customer on the given date.

Loans and Deposits Schedule

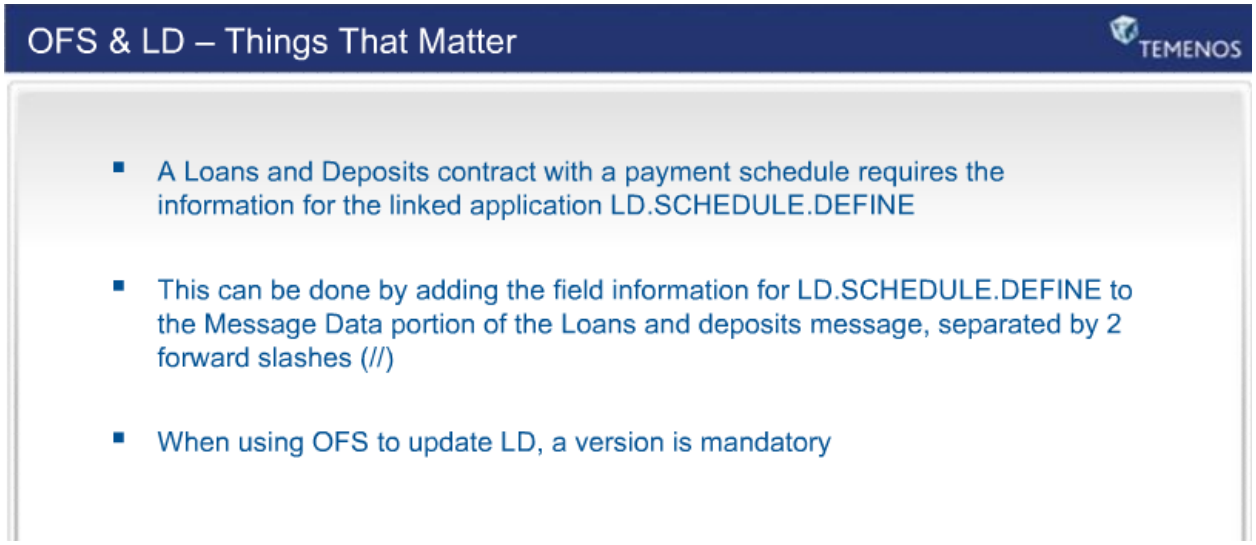


Key fields with sample data	
Forward Backward Key	4
Base Date Key	3
Sch	I for Interest P for Principal
Date	for interest and principal
Amount	2000000 For principal only.

A Loans and Deposits contract with a payment schedule requires the information for the linked application LD.SCHEDULE.DEFINE.

This can be done by adding the field information for LD.SCHEDULE.DEFINE to the Message Data portion of the Loans and deposits message, separated by 2 forward slashes (//).

When using OFS to update LD ,a version must be used. OFS returns an error “VERSIONS WITH NO AUTHORISERS NOT ALLOWED” in case you don’t.



This example shows the deposit transaction discussed previously, input with schedules through OFS. A version called LD.THRU.OFS has been used.

You can see that the input to the 2 applications is separated by 2 forward slashes (//), indicating to OFS that the first part of the message belongs to LD.LOANS.AND.DEPOSITS and the second part to LD.SCHEDULE.DEFINE.

Shown here is the response message for LD transaction input with schedules.

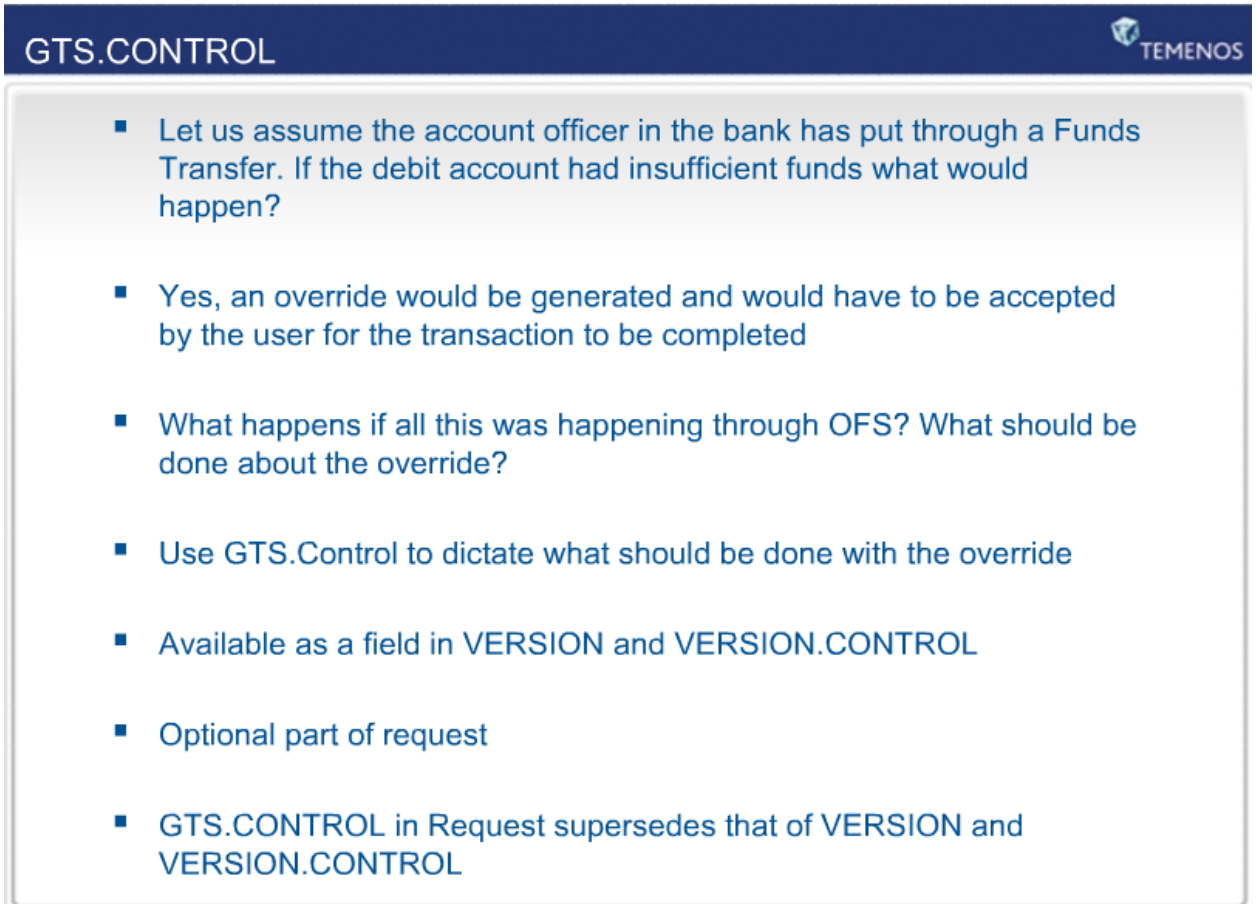
Here again, the data from the 2 applications are separated by 2 forward slashes (//), indicating that the part before the forward slashes (//) belongs to LD.LOANS.AND.DEPOSITS and the part after to LD.SCHEDULE.DEFINE.

Ofs related fields in version

Let us assume the account officer in the bank has put through a Funds Transfer. Normally, if the debit account had insufficient funds what would happen? Yes, an override would be generated and would have to be accepted by the user for the transaction to be completed.

Now, what happens if all this was happening through OFS? What should be done about the override?

GTS Control tells OFS what action to take if an OVERRIDE or ERROR is encountered. GTS Control may be specified in the Version or at the VERSION.CONTROL level. GTS control may also be specified within the OFS request itself. If it is in the message, this value supersedes that given in the version or version control.



What can we do if an override is encountered? Either approve it or place the message on hold so that it may approved manually later.

What can we do if an error is encountered? Either reject the message or place the message on hold. The errors mentioned here are data errors, where though the message is syntactically correct, errors in the data prevent it from been processed.

GTS Control permits five values which give you various combinations of these two sets of actions.

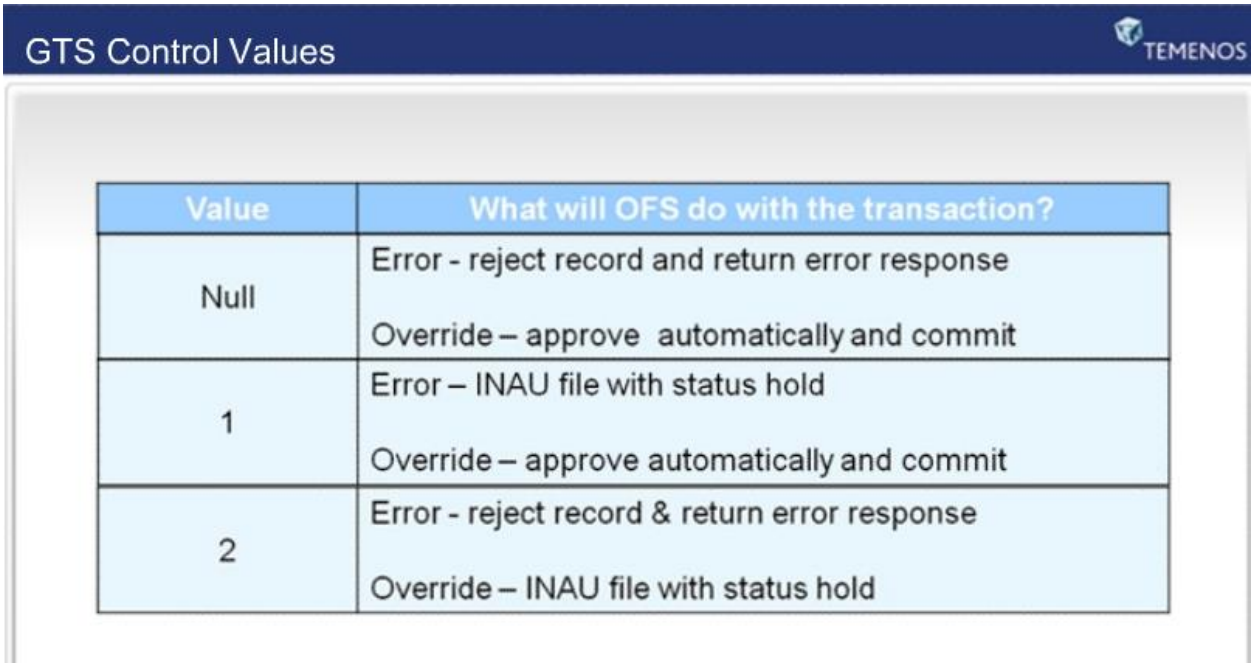
When no value is specified in this field, validation errors will result in the message (i.e. transaction) being rejected, and response returned with details of the error message. A message which results in an override however has the override accepted and transaction committed.

When a value of 1 is specified, a transaction record which results in a data validation error is put in the unauthorised file with status set to hold. A message which results in an override has the override accepted, and transaction committed.

When a value of 2 is specified in this field, data validation errors will result in the message being rejected, and response returned with details of the error message. A transaction which results in an override is written into the unauthorised file with status set to hold.

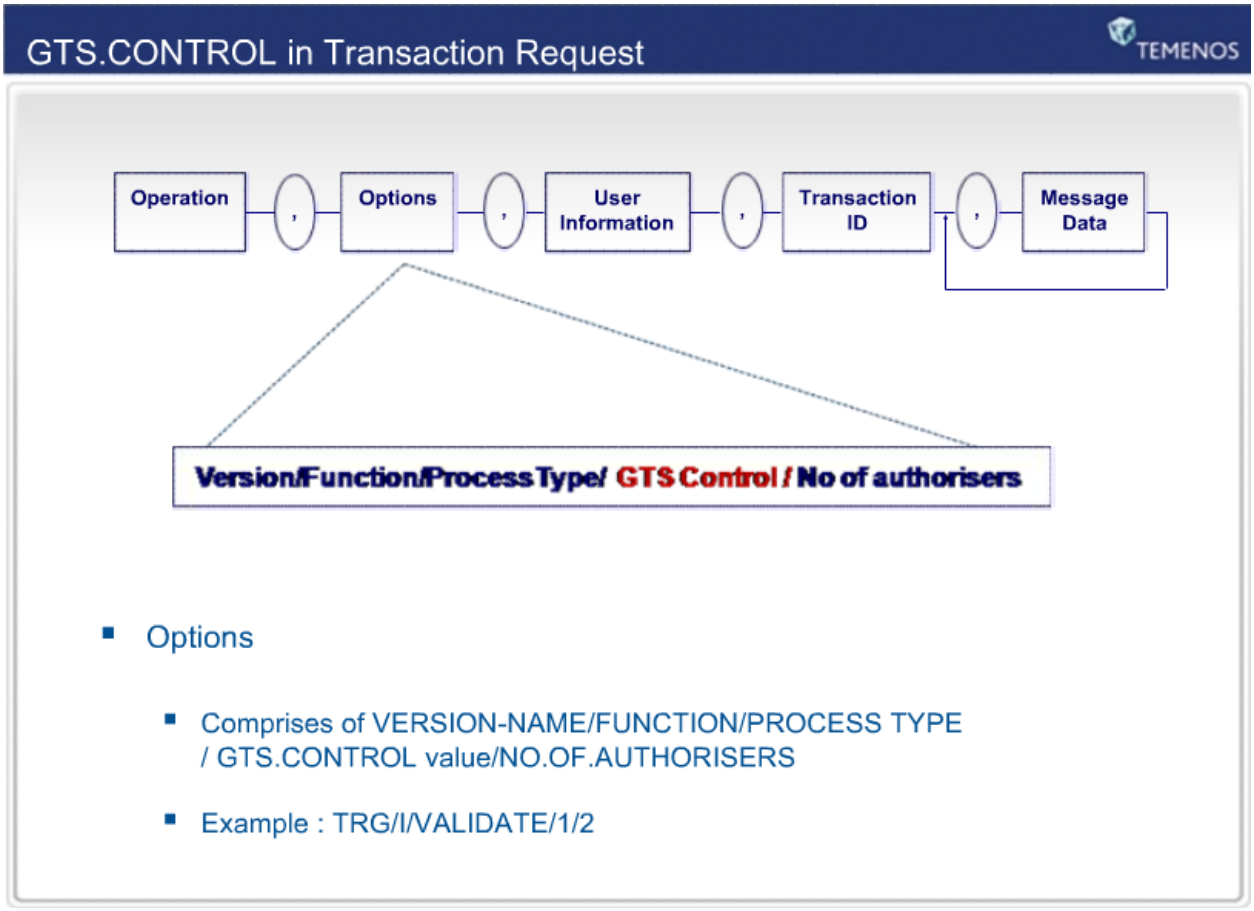
When a value of 3 is specified in this field, data validation errors will result in the message being rejected, and response returned with details of the error message. A transaction which results in an override is written into the unauthorised file with status set to hold.

The value 4 tells OFS to write all transactions into the unauthorised file with status set to hold.



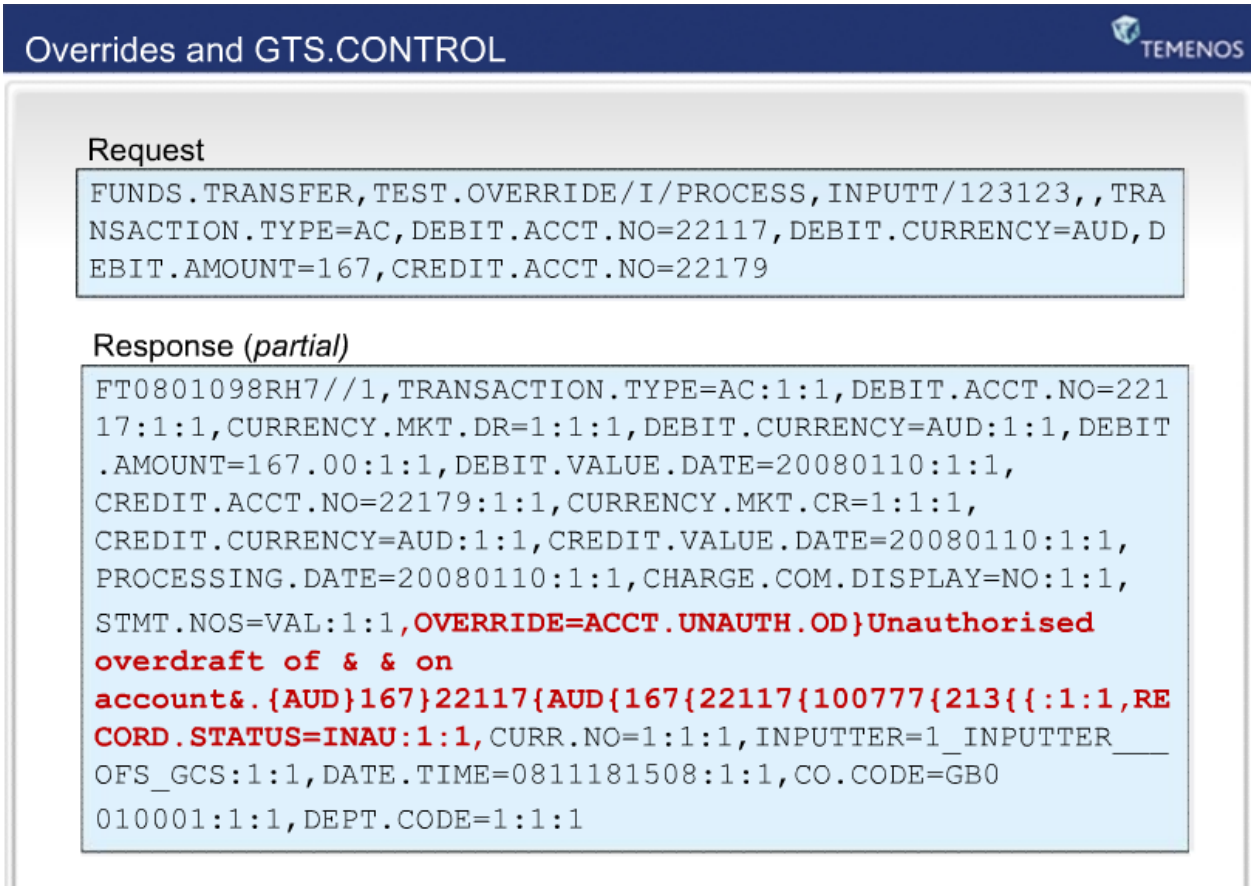
As you may recall GTS.CONTROL is the 4th sub-part in the Options portion of a OFS request.

This supersedes the value given (if any) in VERSION or VERSION.CONTROL.

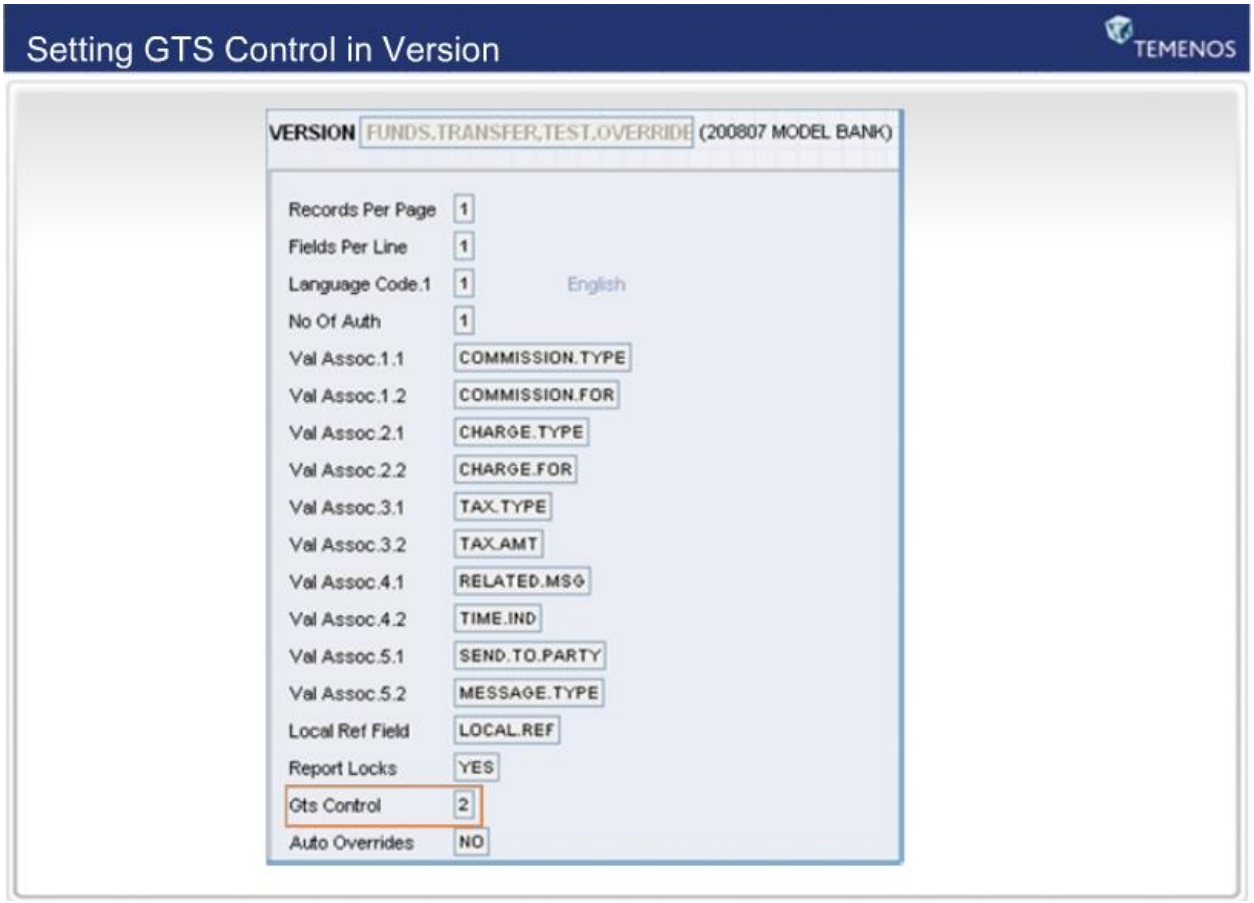


0You can see an OFS message to input a Funds Transfer and the response to that message. This message uses a version called TEST.OVERRIDE which has 1 authoriser and null in the GTS.CONTROL field.

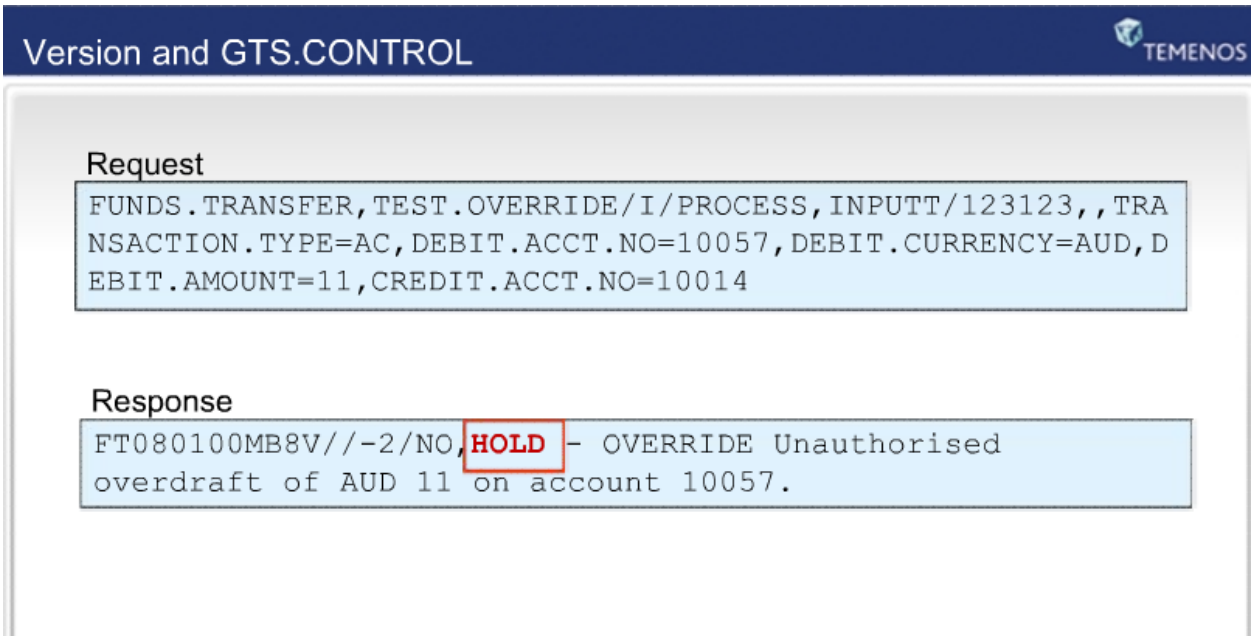
Note that the message is in the INAU state.



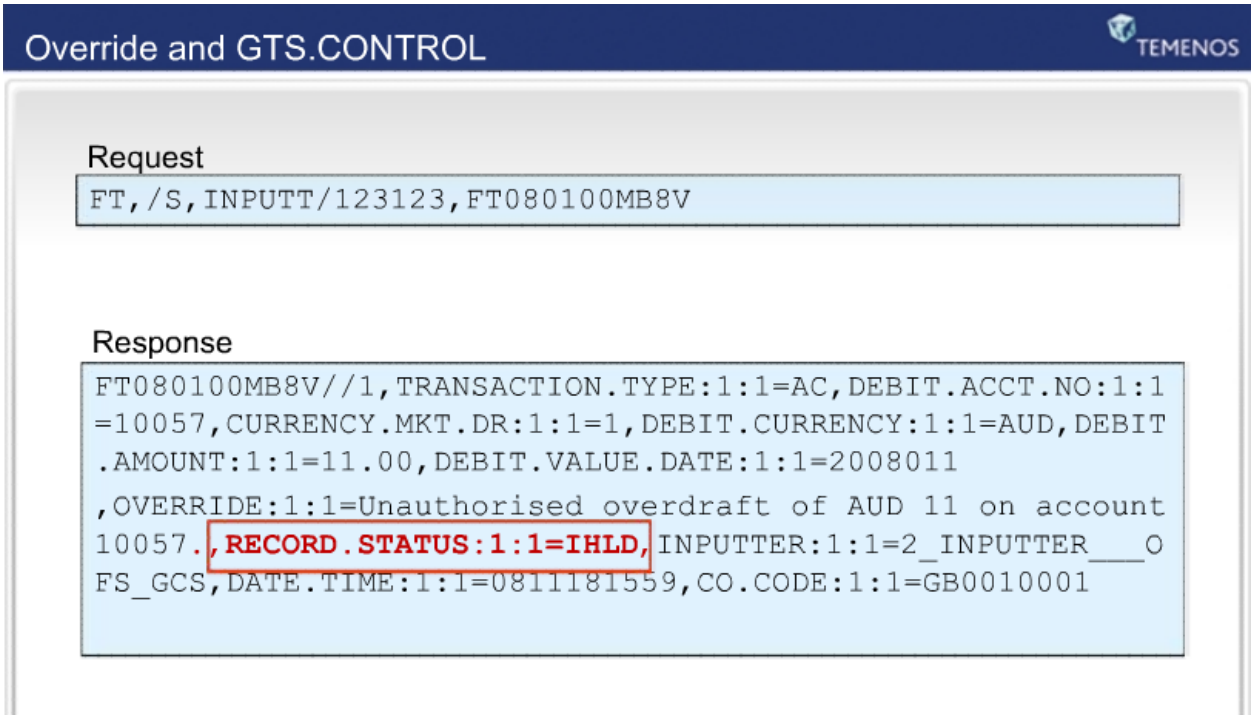
Let us set the GTS.CONTROL field in the version to 2.



Try inputting the FT again. What happened now? The record was put on Hold this time.



Let us check the record status using another OFS message. Notice that the status shows IHLD.



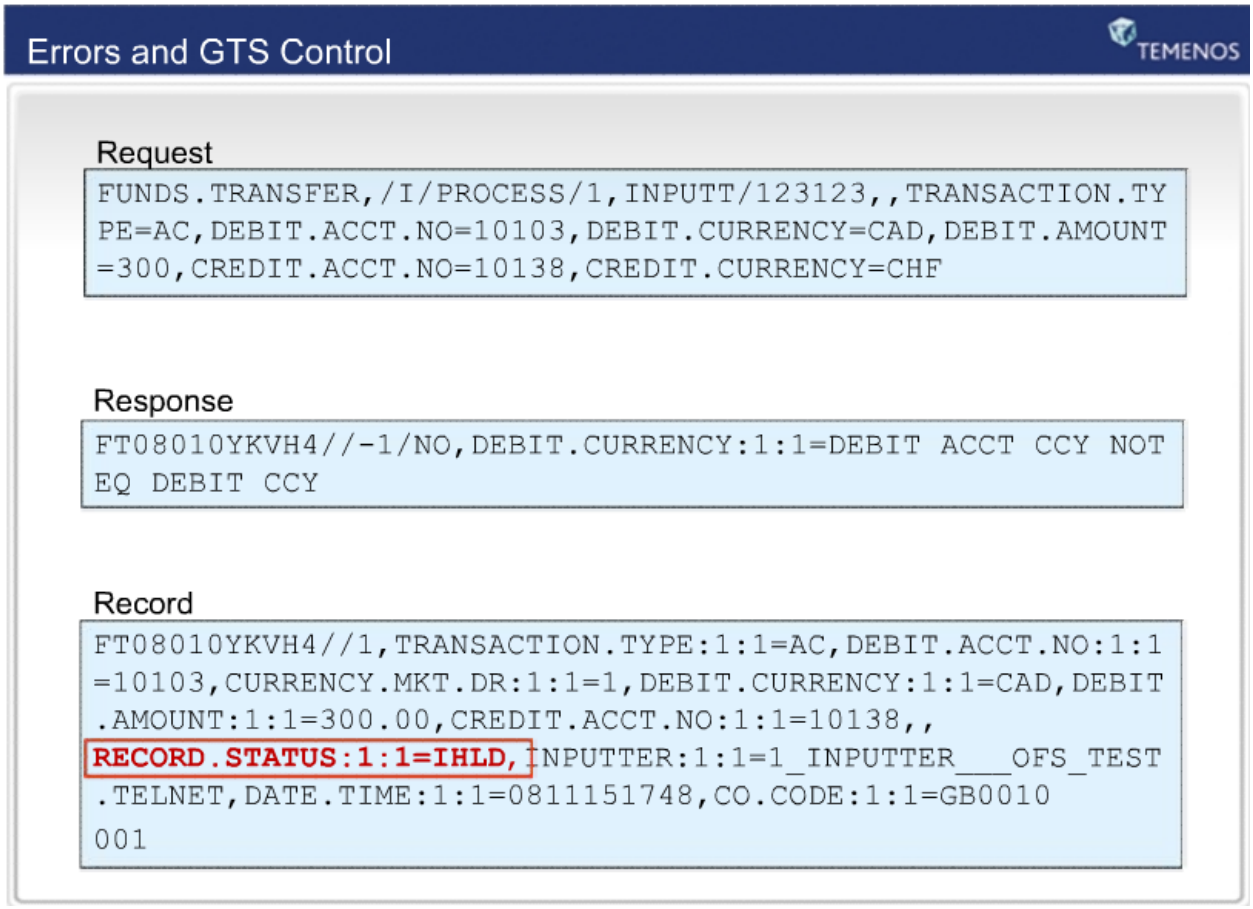
What happens if both the version and the message have GTS Control specified? As you recall, the value in the version, oops, no, the value in the message supersedes that of the version.

Here in this example the version has a GTS control of 2 , whereas the message has 1. The status as you can see in the response shows INAU, implying that the GTS control of the message has been used.

You just saw the effect of GTS Control on a transaction with overrides. Let us see the effect of GTS Control on a transaction that has errors.

Here you see an OFS message which is syntactically correct. However, an attempt is made to debit in CAD from an account that is in CHF.

Since the GTS control has been set as 1 in the message, the transaction is put on hold and the resultant record id given back in the response. You may view the record in the INAU file.

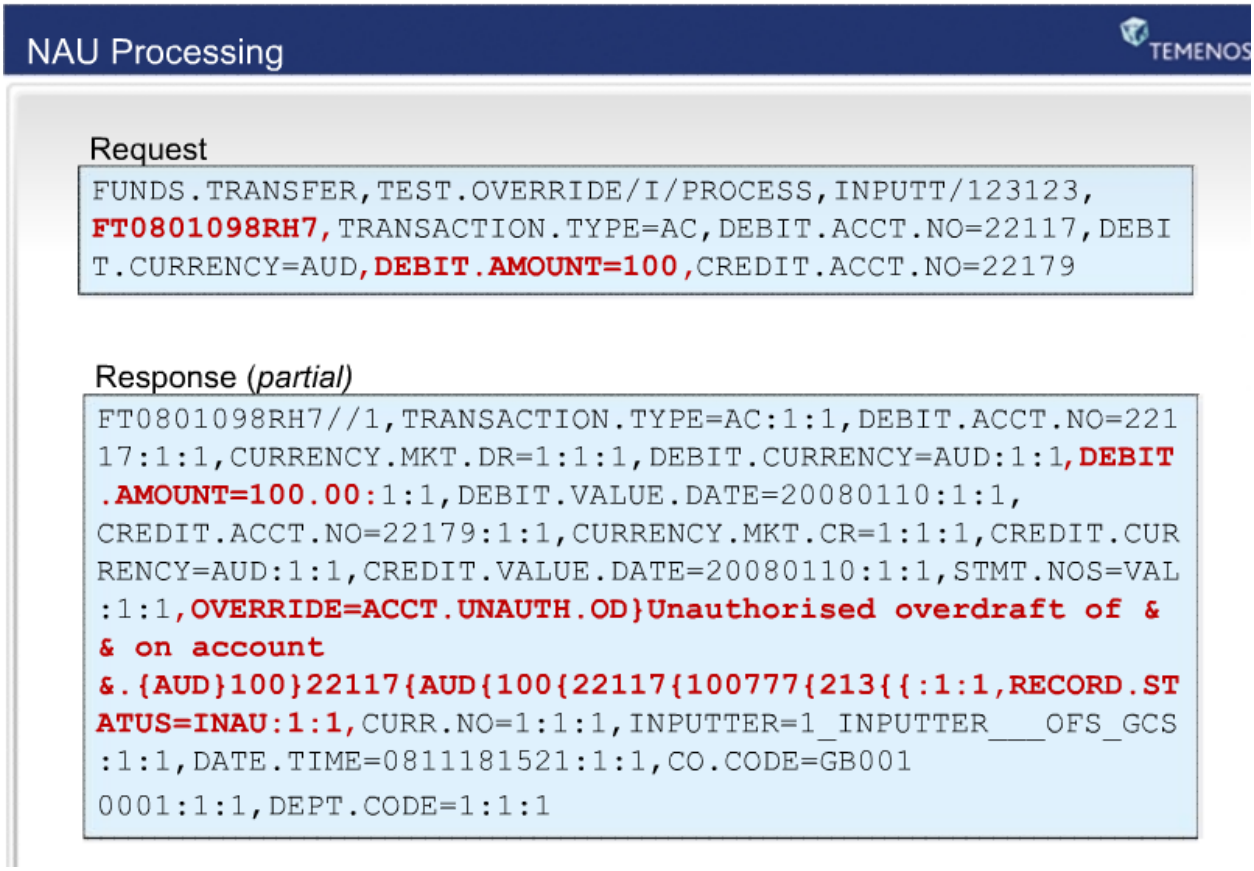


Now, you are going to learn about NAU Processing.

Here you see an OFS message for a Funds Transfer. This uses the id of the previous FT transaction. In effect, you are inputting the same transaction but with modified values. In this case, the only modified value is the amount.

What happened to the transaction? You can see that the record is in the INAU state. i.e. it overwrote the existing record in NAU file. Now, T24 didn't object or throw out any warnings . If you had done this through browser you would have been warned that you are "overtaking the work of another inputter".

So, how do you control this behaviour in OFS? Yes, use the NAU.PROCESSING field in the version. Let us try to do that.



Normally when a transaction is committed using OFS an unauthorised record is formed by the application. But, what if the record already exists in the INAU file? NAU Processing allows you to control what happens in this situation.

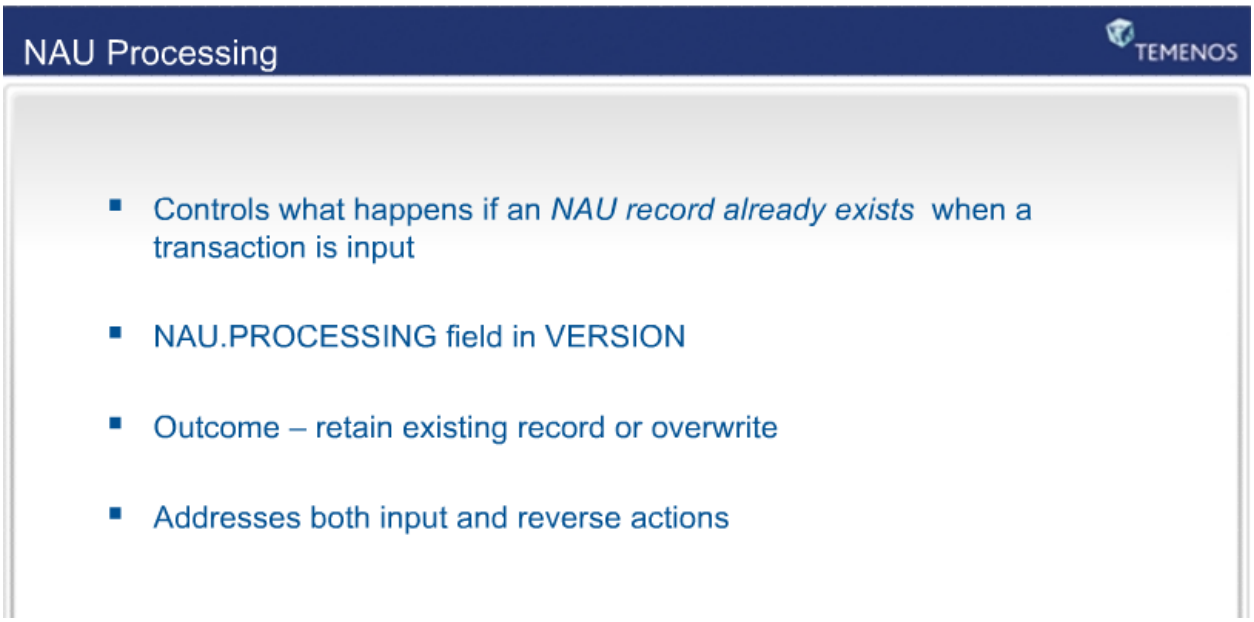
The control is applied through a field in the VERSION called NAU.PROCESSING. You cannot do this through the message as you did for GTS Control.

Obviously there are only two outcomes

a. Keep the existing NAU record, or

b. Overwrite it with the one in the OFS message

Now we know that NAU records may be created not only when we input a transaction , but also when we reverse an existing live record. NAU Processing addresses both.



NAU.Processing can take on values 0,1,2,3 or NULL. How does the system behave for each of these values?

0 - Reject messages when an NAU record exists

1 – This addresses an attempt to input a record which is already in the NAU state. In this case the NAU record is overwritten with the data from the OFS message. If the version in the OFS message is a 0 authoriser version, and the inputter in the record and user from OFS are the same, the record is authorised. If the users are different, the record is written to NAU but not authorised.

2 – This addresses an attempt to reverse a live record when the same record is in the NAU file also. The existing NAU record is deleted and live record is reversed.

3 – Takes on the cumulative behaviour of 1 & 2.

Null – The behaviour depends on factors like is it a modification or reversal, no of userS and has zero authorisation been used.

NAU Processing - Values

TEMENOS

Value	Behaviour
0	Applies to input & reversals . Effect - Reject messages when an NAU record exists
1	Applies to Input only. Effect - Overwrites NAU record with the values from the OFS message
2	Applies to Reversals only. Effect - Deletes NAU + reverses live
3	Applies to input & reversals . Effect - Behaviour of 1 & 2
NULL	The behaviour depends on factors like is it a modification or reversal, no of users, and has zero authorisation been used.

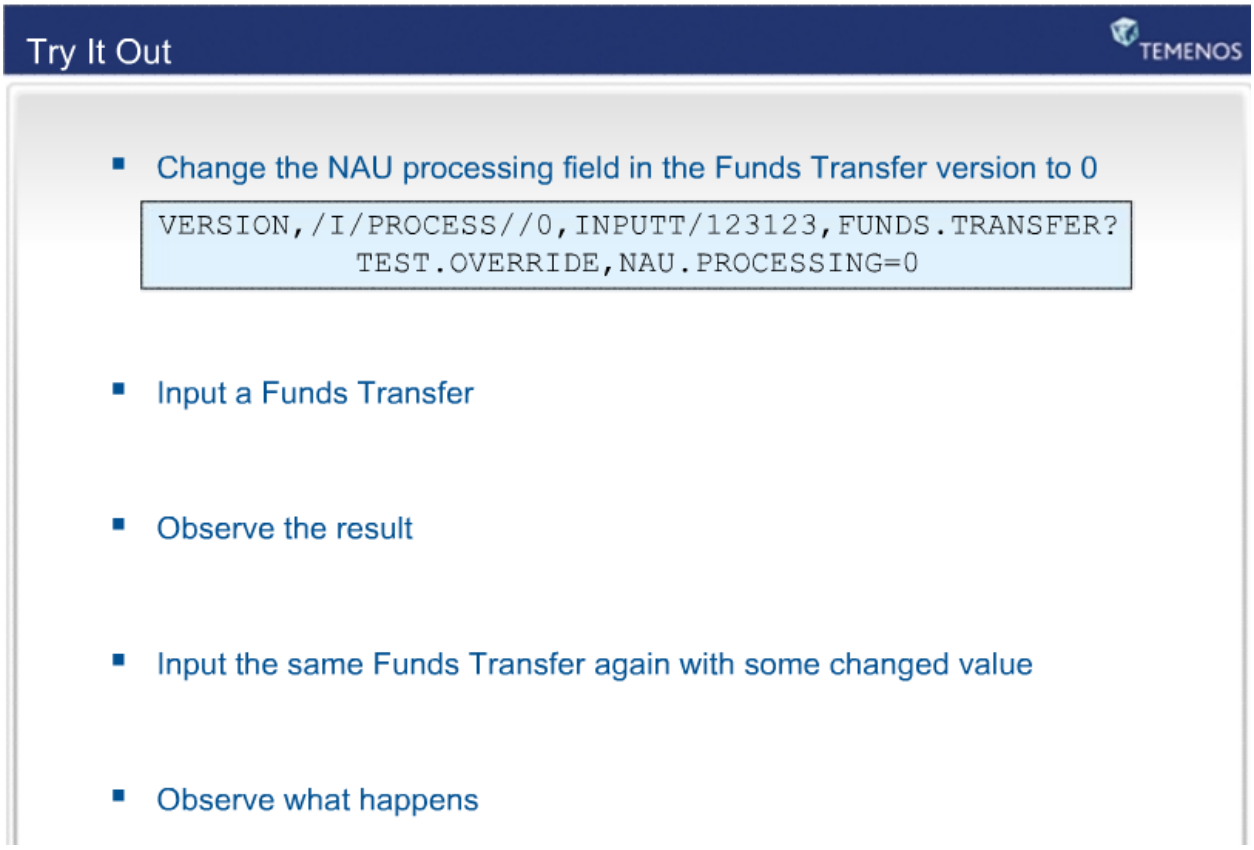
Change the NAU processing field in the Funds Transfer version to 0. You can do this by using another OFS message as shown in the screen.

Input a Funds Transfer with some changed amount. Ideally choose a debit account that has a balance so that an override message is NOT raised.

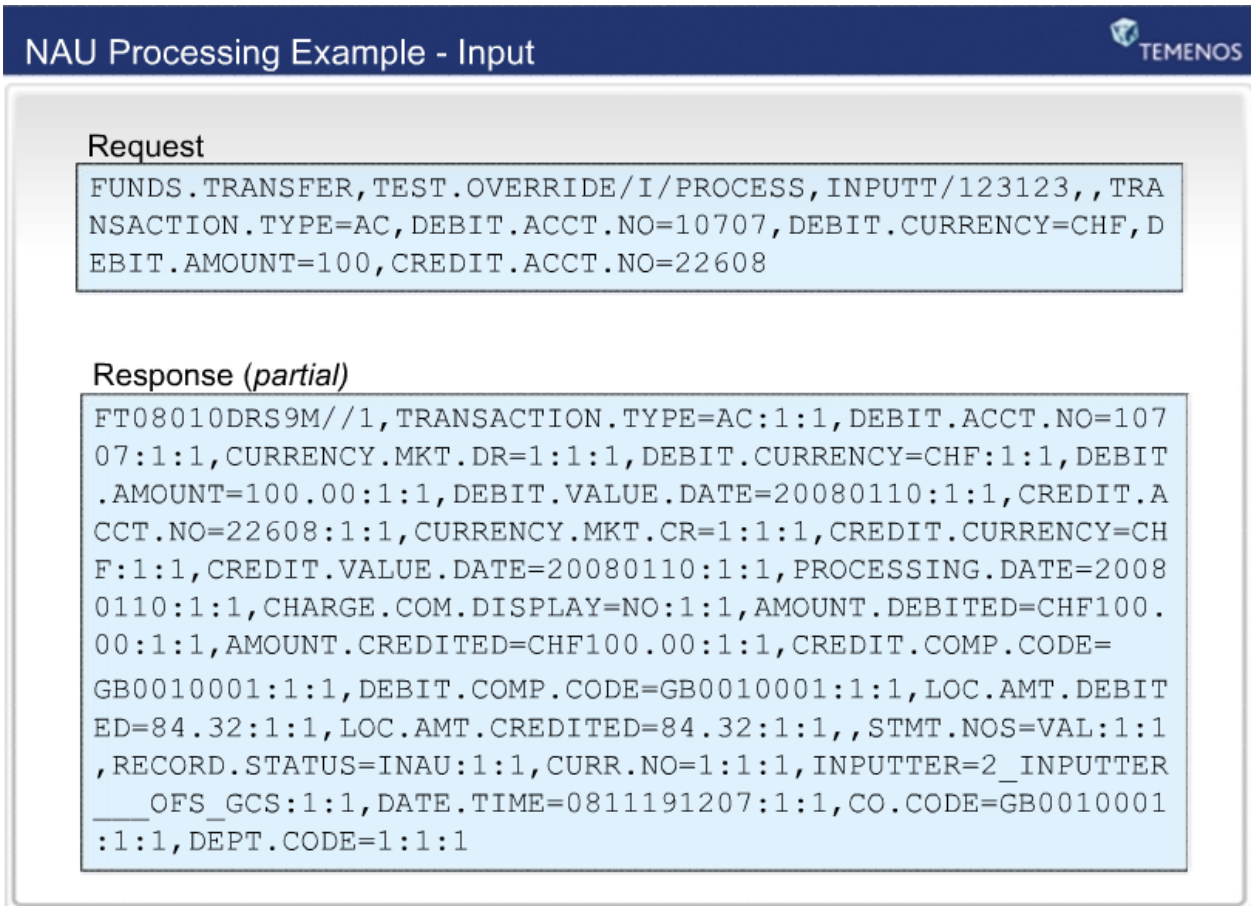
Observe the result.

Input the same Funds Transfer again with some changed value.

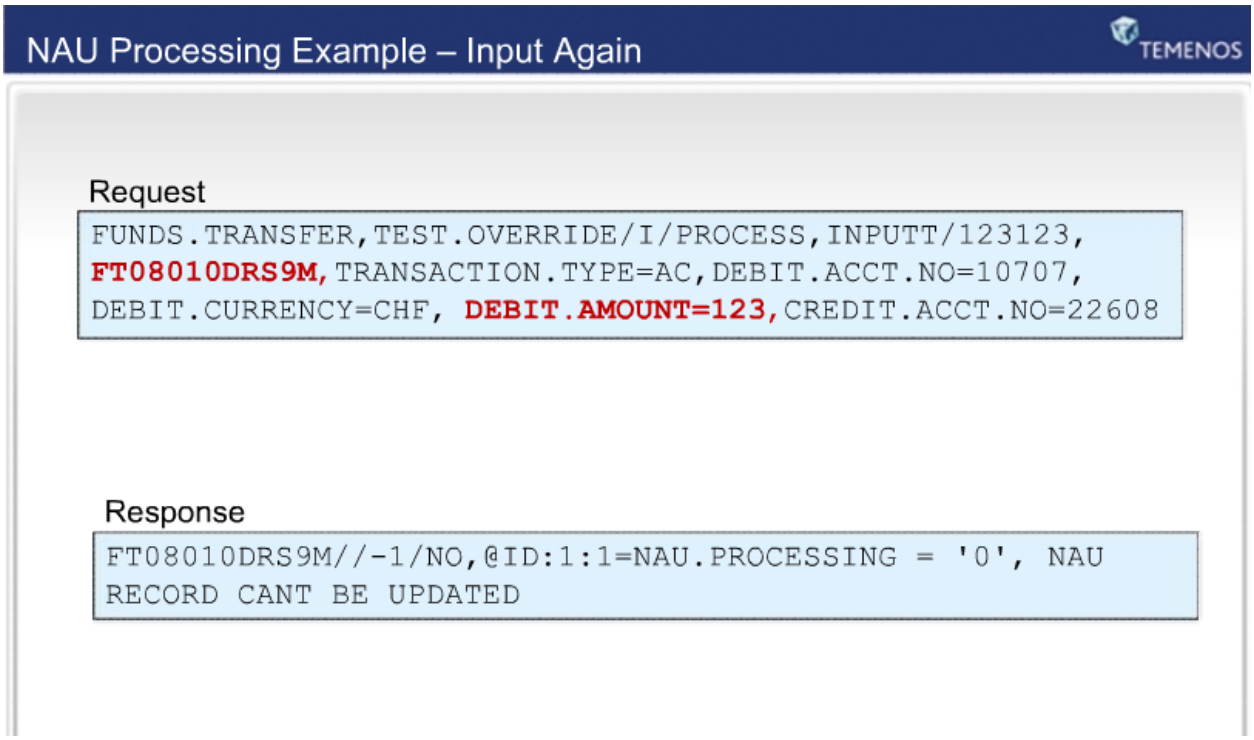
Observe what happens.



Look at the sample message and its response. This shows a funds transfer from an account that has sufficient funds. Therefore, no override has been generated.



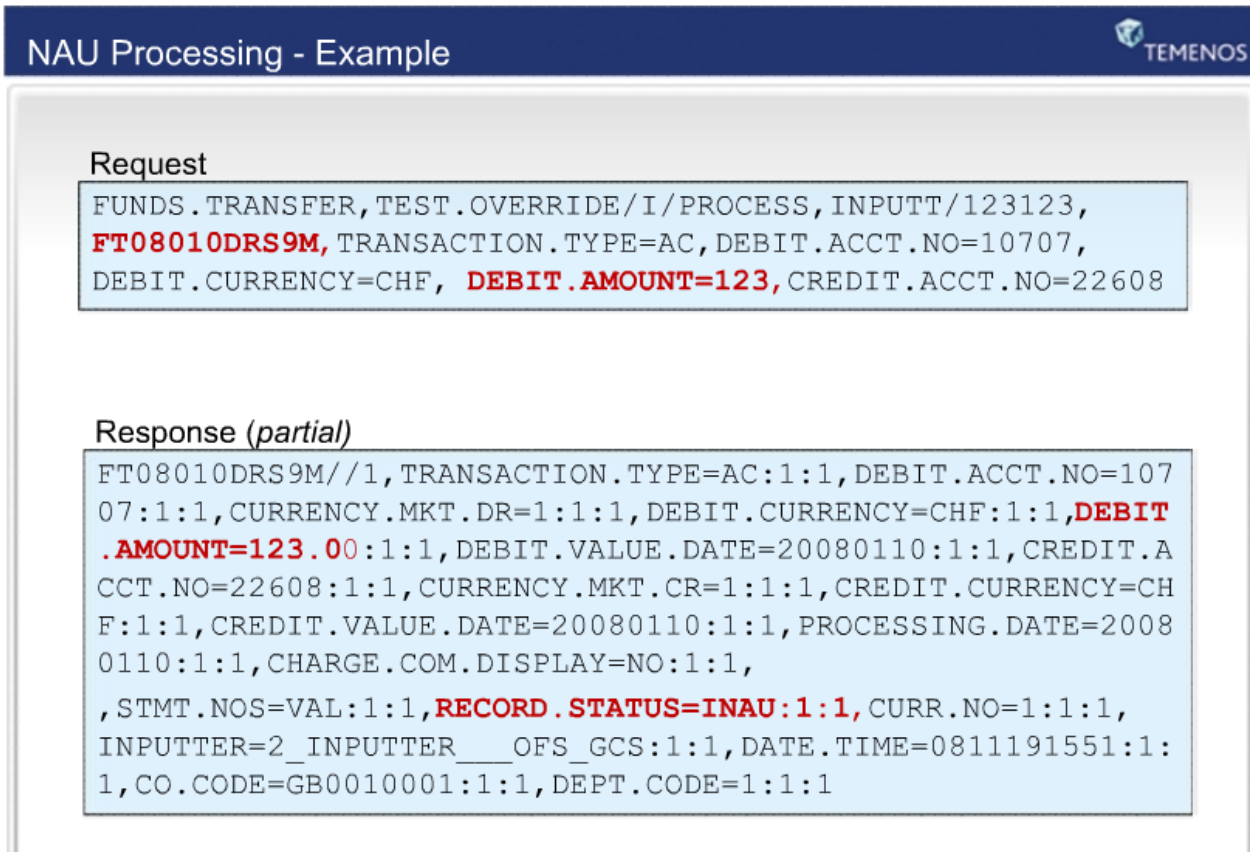
Now, you see the same transaction re-input with a different amount. The changes are highlighted. You can see that the request was rejected.



What happens if we set the NAU processing field to 1?

This example shows the same transaction input with a different debit amount.

The record is taken into INAU and you can see that the debit amount in the response is the updated one.



This table lists the behaviour if NAU.PROCESSING is NULL and an attempt is made to input a record, which is already in NAU state.

More on NAU processing

TEMENOS

- This table lists the behaviour if NAU.PROCESSING is NULL and an attempt is made to input a record which is already in NAU state

Does NAU record Exist?	Function	No. of Authorisers	IS there only one user on the NAU record and is it the same as the OFS user ?	Processing	Corresponding NAU.PROCESSING Flag
Yes	I	1 or more	No	Overwrites fields from OFS record onto NAU record and writes record to NAU file.	1
Yes	I	0	No	Overwrites fields from OFS record onto NAU record and writes record to NAU file	1
Yes	I	0	Yes	Overwrite fields from OFS record onto NAU record and writes record to the authorized file.	1

This table lists the behaviour if NAU.PROCESSING is NULL and an attempt is made to reverse a record, which is already in the NAU file.

More on NAU Processing

TEMENOS

- This table lists the behaviour if NAU.PROCESSING is NULL and an attempt is made to reverse a record which is already in NAU file

Does NAU record Exist?	Function	Live Rec	No. of Authorisers	IS there only one user on the NAU record and is it the same as the OFS user ?	Processing	Corresponding NAU.PROCESSING Flag
Yes	R	Yes	1 or more	N/A	Sets live rec to RNAU status. Deletes NAU record.	2
Yes	R	No	1 or more	N/A	Deletes NAU record	2
Yes	R	No	0	No	Deletes NAU record	2
Yes	R	Yes	0	No	Sets live record to RNAU status. Deletes NAU record	2
Yes	R	No	0	Yes	Deletes the NAU record.	2
Yes	R	Yes	0	Yes	Deletes NAU record and does Reversal on Live file. Reversal with 0 authoriser will move the record in live to history	2

Now, you try to write an OFS message to input a funds transfer. Use the GTS control to ensure that transactions with errors or overrides are put on hold.

Also, you try to write another OFS message to change the amount. Use the NAU Processing field to ensure that existing NAU records are not overwritten.

Try It Out

TEMENOS

- Write an OFS message to input a funds transfer. Use the GTS control to ensure that transactions with errors or overrides are put on hold
- Write another OFS message to change the amount. Use the NAU Processing field to ensure that existing NAU records are not overwritten

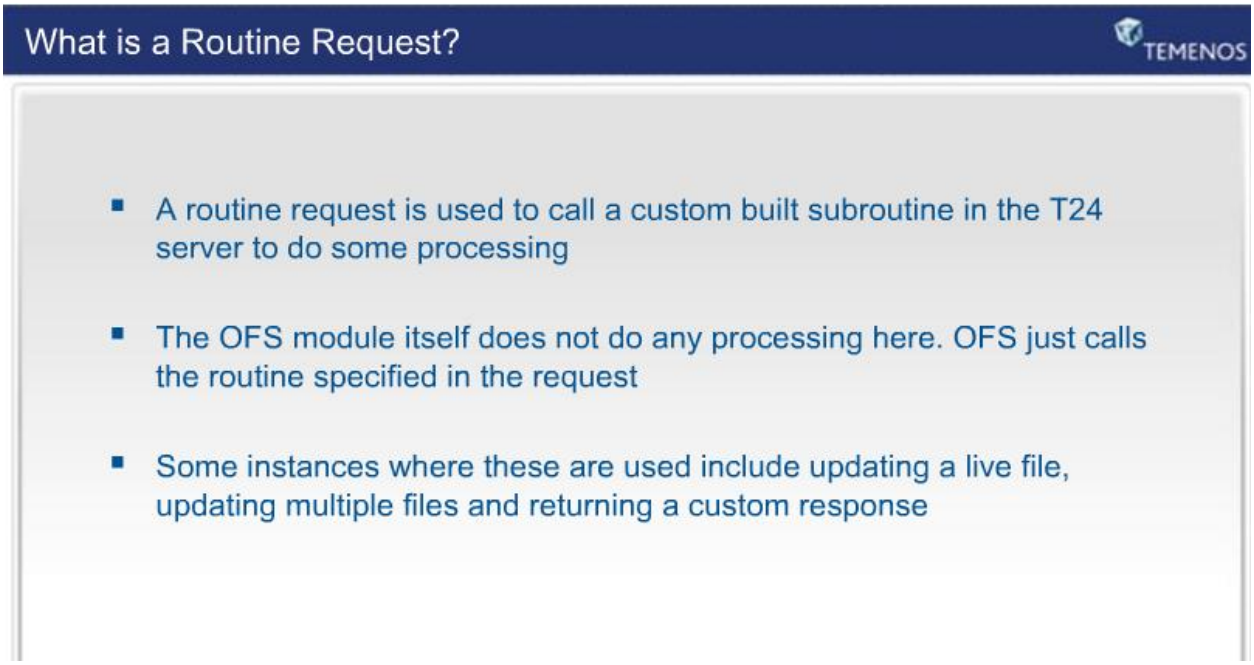
Calling routines

Now we move on to a very interesting type of OFS Message – The Routine request.

The routine request as you may have guessed from the name, invokes a custom built routine within the T24 server to do some processing.

The OFS module itself does not do any processing here. OFS just calls the routine specified in the request.

T24 routines are useful especially when updates to multiple files (involving more than one T24 application) have to be effected. Routine requests are very flexible and can even be used to return some data, i.e. a custom response from T24, like a query.



The OFS Routine request takes the Custom Routine name, User information and Custom data only. The options and transaction id that we saw used in transaction requests are not applicable here.

Custom Routine

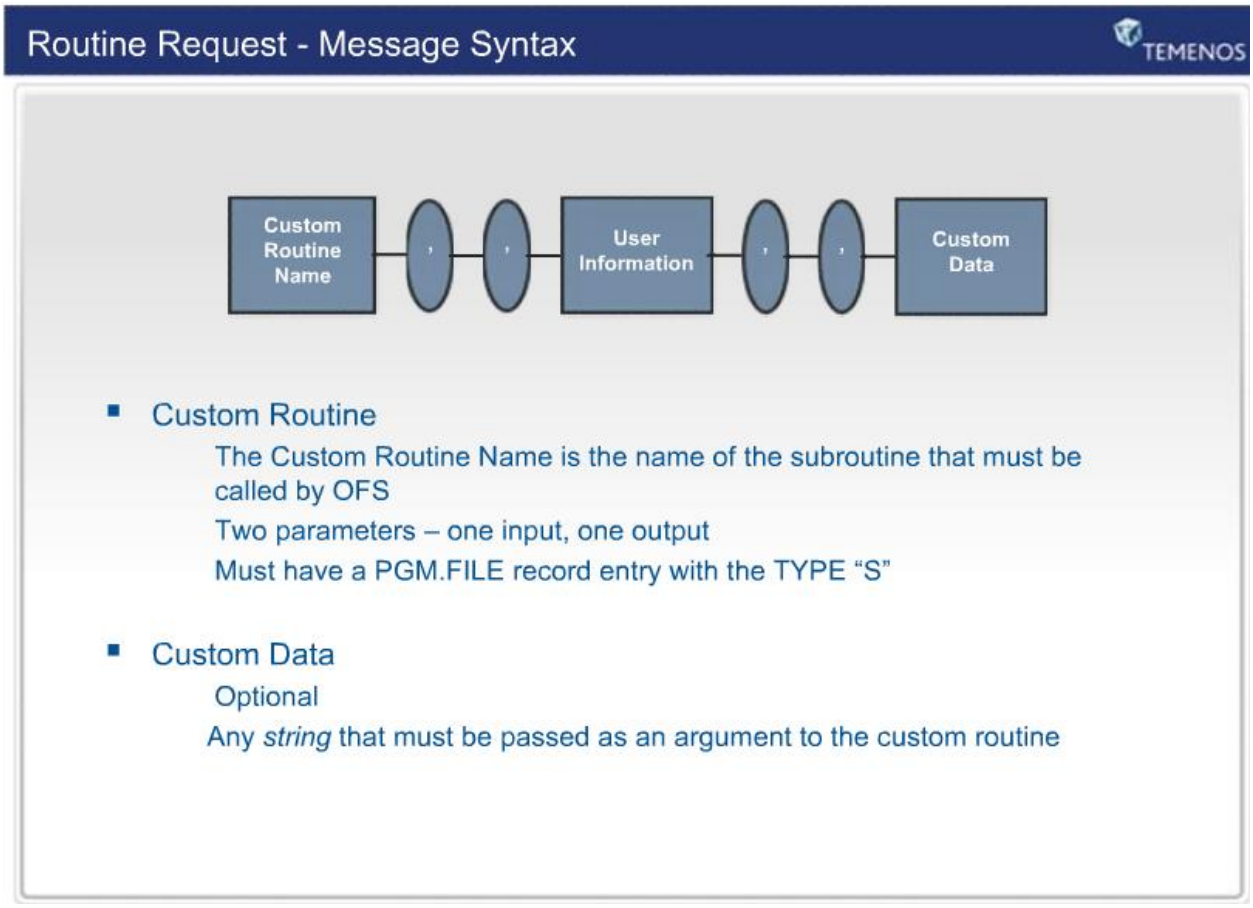
The Custom Routine Name is the name of the subroutine that must be called by OFS.

The routine must have two parameters, one for input and one for output.

The routine name must have a PGM.FILE record entry with the TYPE field value set to “S”.

Custom Data

This is optional. It is used for passing any string as an argument to the custom routine.



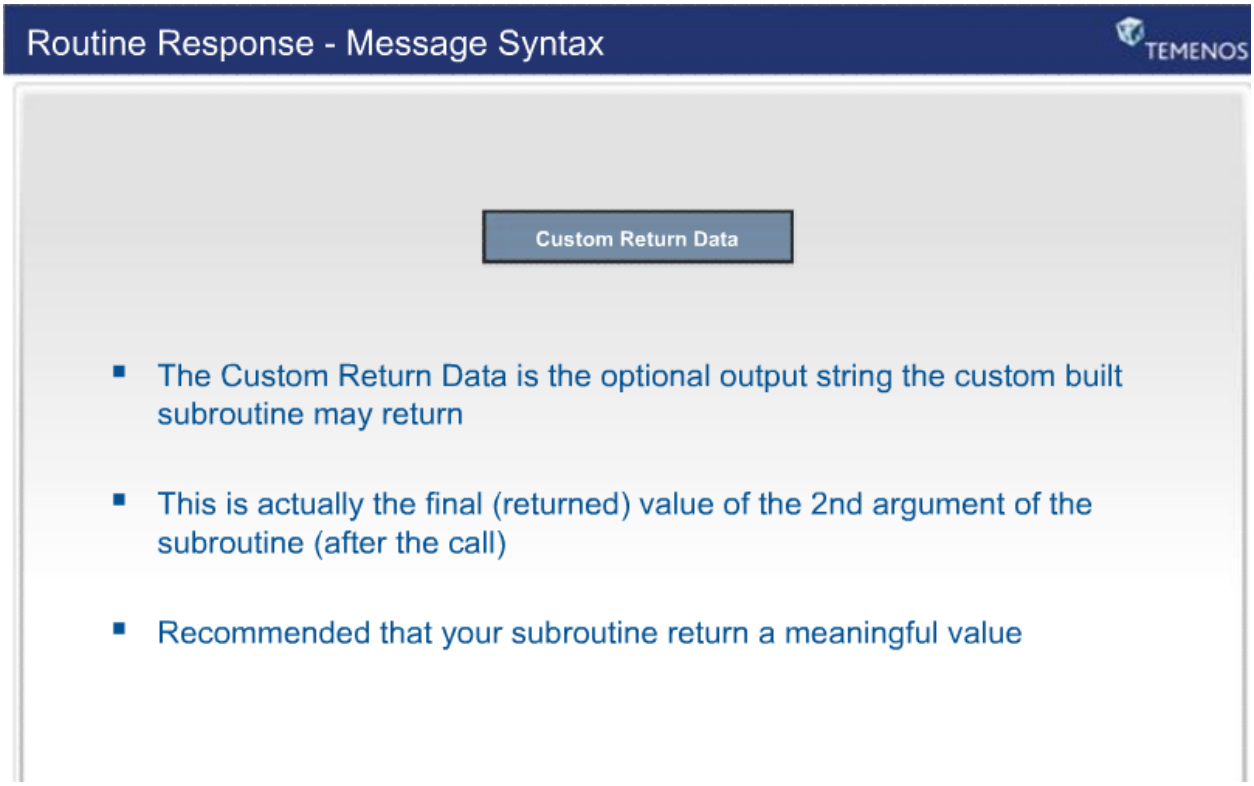
Actually, there is no standard syntax structure laid down by the OFS for routine type responses.

The diagram here shows a sort of syntax structure of OFS Routine type response messages, if you want to call it that way.

The Custom Return Data is the optional output string the custom built subroutine may return.

This is actually the final (returned) value of the 2nd argument of the subroutine (after the call).

Though this is optional it is highly recommended that your subroutine return a meaningful value to avoid any problems that may arise otherwise.



This page shows the source code of a simple custom-built subroutine that you will be using as an example routine type OFS message.

This subroutine has two arguments – one for input and second for output. This is a must, failing which a "SUBROUTINE_PARM_ERROR" will be encountered.

"SUBROUTINE_PARM_ERROR" is given when a jBASE subroutine is called with incorrect number of arguments. For information on "SUBROUTINE_PARM_ERROR" you may refer to jBASE® manuals.

Routine Type Example - The Code

```
SUBROUTINE T24.ECHO.ROUTINE(Y.REQUEST, Y.RESPONSE)
!*****************************************************************************
! Subroutine to return the string passed thru the Y.REQUEST argument)
! Arguments:
! Y.REQUEST  - contains the incoming string
! Y.RESPONSE - contains the echo'ed back string
!*****************************************************************************
!
  IF Y.REQUEST NE '' THEN                    ; * Check for emptiness
    Y.RESPONSE = "Received from OFS : " : Y.REQUEST ; * Talk back !
  END ELSE
    Y.RESPONSE = "Did not receive anything from OFS" ; * Tell 'em !
  END
!
  RETURN
!
END
```

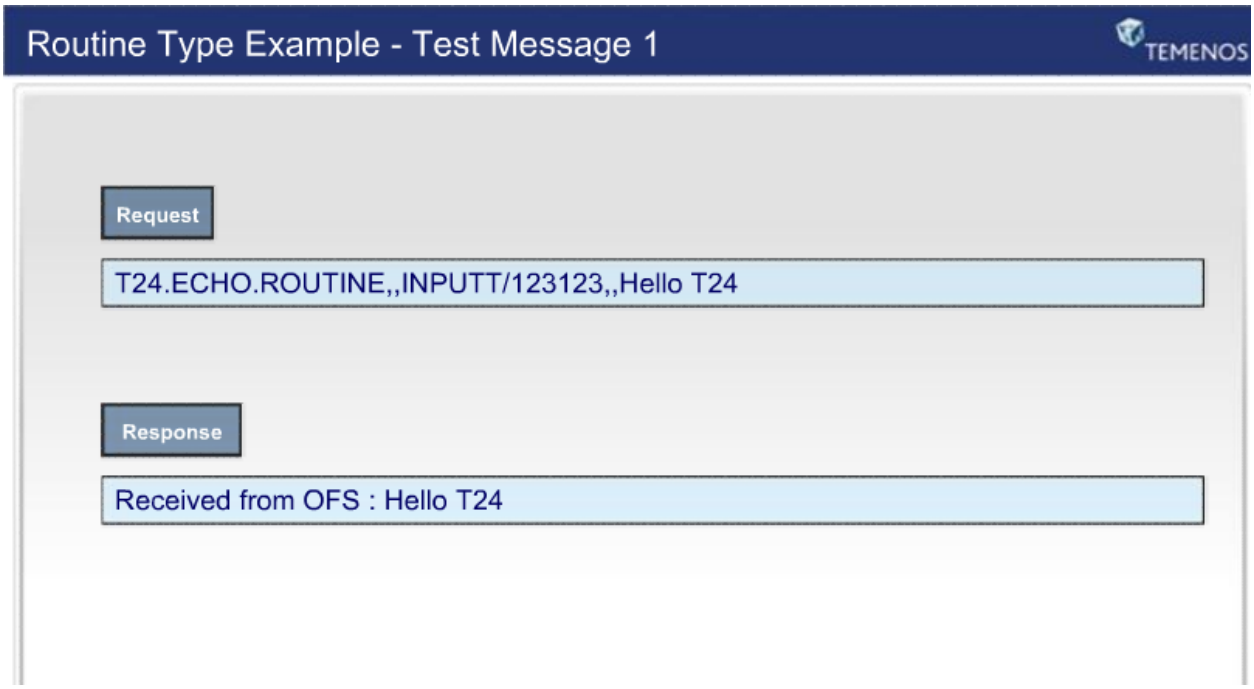
The illustration shows the PGM.FILE record for the example subroutine we have seen. The name must correspond to the routine name and type must be S.

Routine Type Example - PGM.FILE Entry

The example here shows the routine type request and response messages.

In this example, the custom subroutine created – T24.ECHO.ROUTINE is supplied in the Custom Routine Name portion of the message. Also the Custom Data portion is supplied as an argument to the subroutine which in this case is the string “Hello T24”.

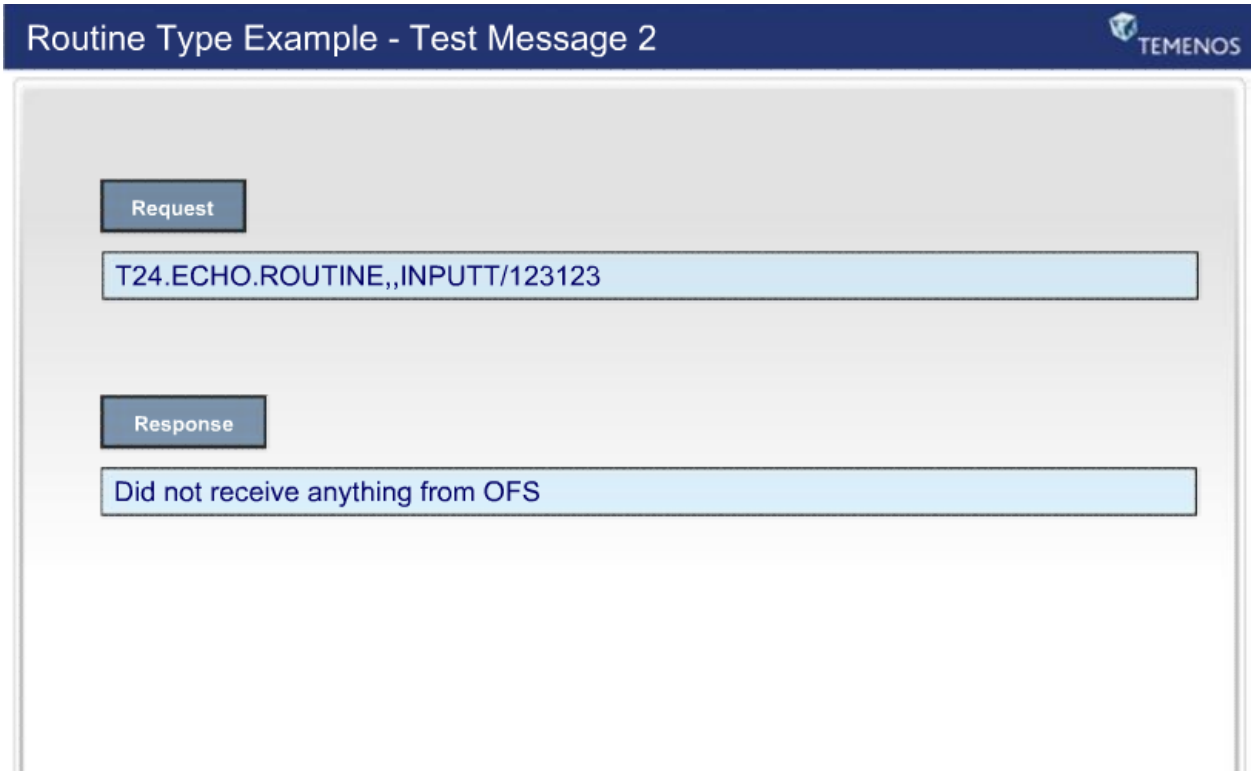
In the response the string returned by OFS, is the string assigned to the output argument (variable) – Y.RESPONSE, in our custom subroutine – T24.ECHO.ROUTINE. Going by source code, this shows that the Custom Data portion supplied in the request message is passed to the subroutine through the incoming argument (variable) – Y.REQUEST.



The example here shows the routine type request without any arguments.

In this example, the Custom Data portion is not supplied in the message, indicating that the subroutine does not receive any value in the incoming argument (variable) – Y.REQUEST.

Since no string is passed to the custom subroutine – T24.ECHO.ROUTINE, the above given string is received. This is again consistent with our code.

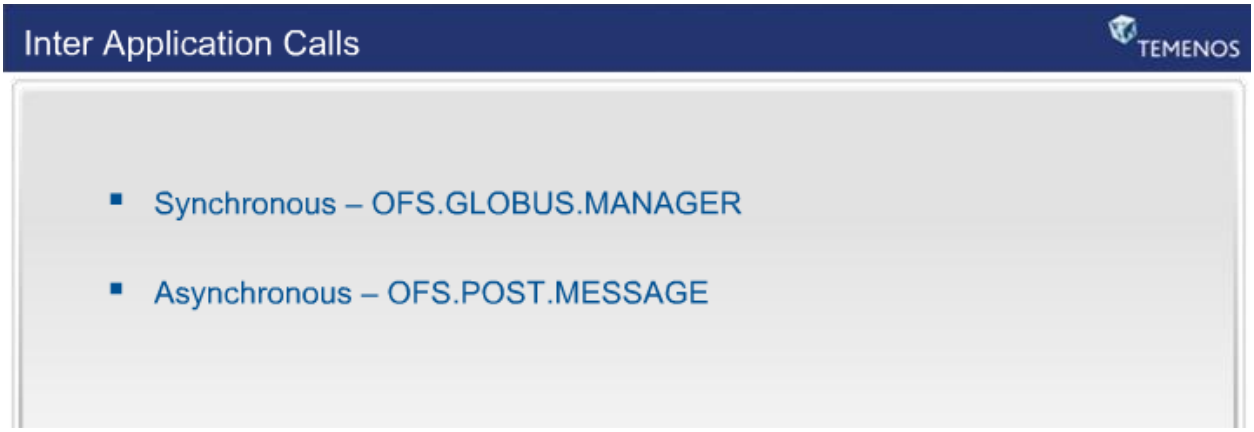


Inter application calls:

It is often necessary to update another application through code. For e.g. when altering a limit we may need to alter sub-limits; we may need to perform some additional updates for certain delivery messages; while executing a standing order we need to add an entry to the receipts file. There are two ways of doing this:

Synchronously – You call a T24 API named OFS.GLOBUS.MANAGER and pass it the OFS message to update the other application. This is a synchronous process, so therefore the routine or application that called OFS.GLOBUS.MANAGER has to wait till it receives a response.

Asynchronously – You call a T24 API named OFS.POST.MESSAGE and pass it the OFS message. This API is asynchronous, which means that your code or application cannot expect an instantaneous response. On a positive note your code or application need not wait for the OFS message to be executed either.



Syntax:

OFS.GLOBUS.MANAGER(OFS.SOURCE.ID, OFS.MESSAGE).

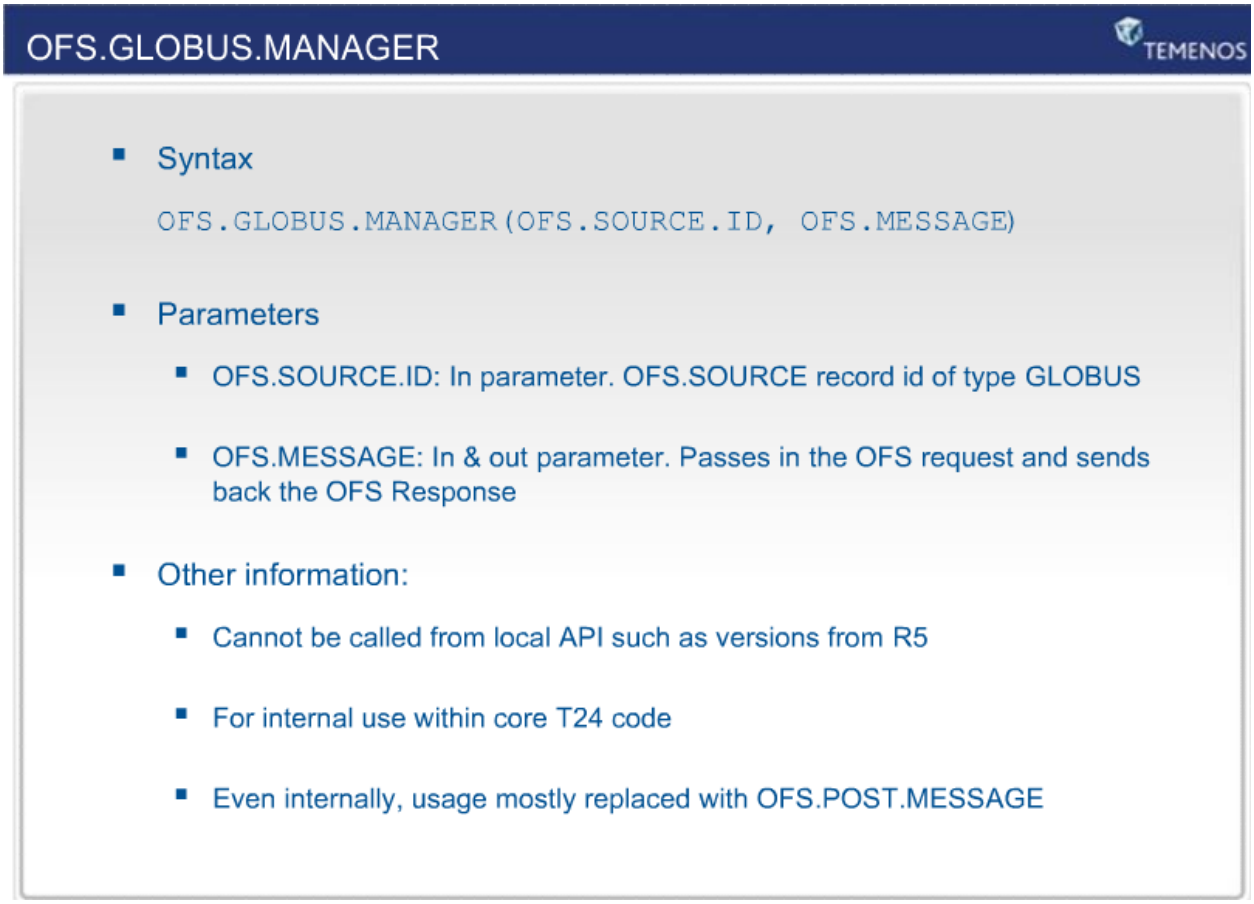
Parameters:

OFS.SOURCE.ID is an in parameter. This is used to pass the OFS.SOURCE record id. This record must be of of type GLOBUS.

OFS.MESSAGE is an In and out parameter. It passes in the OFS request and passes back the OFS Response.

Other information:

This cannot be called from local APIs such as versions from R5 onwards. This API is still used internally within some core T24 code. However even internally, it has been mostly replaced with OFS.POST.MESSAGE or the same functionality achieved through other means.



Examples of OFS.GLOBUS.MANAGER used within T24 core are:

LC.INWARD:

To perform additional updates for inward message types 700, 701, 705, 707.

AZ.UNAUTH.PROCESSING: Deletes unauthorised AZ , TT & FT records during COB.

AA.PROCESS.CONTRACT:

Handles OFS processing for AA.

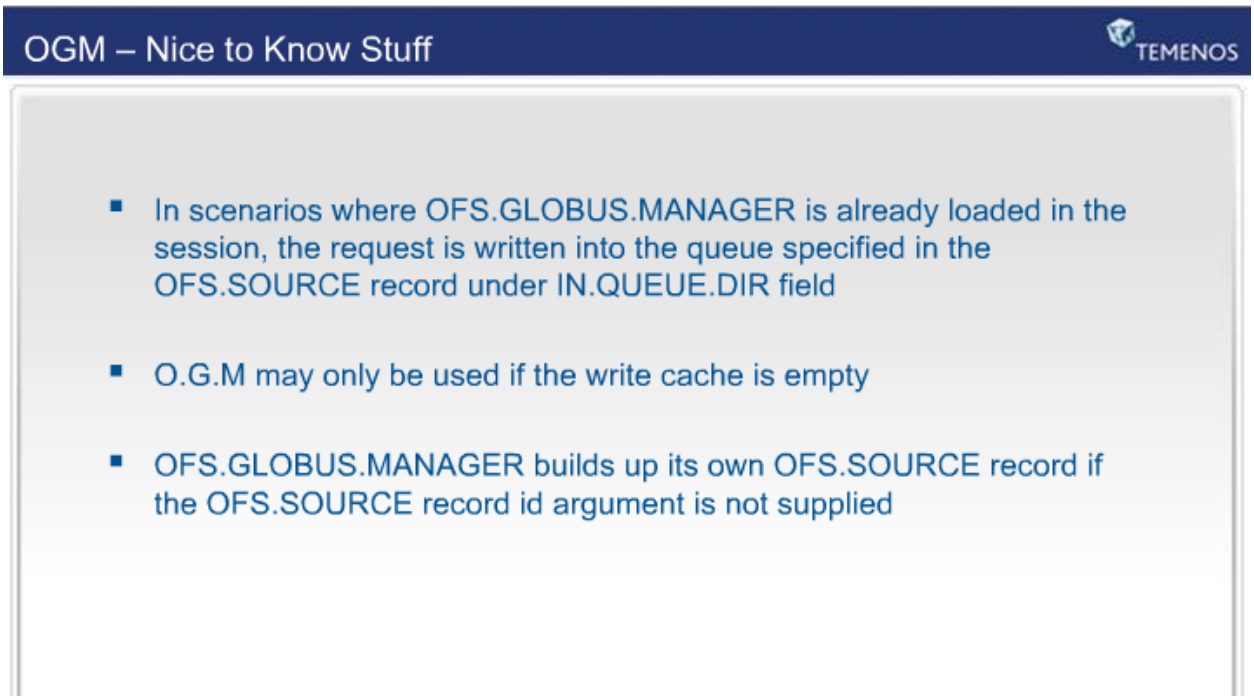
Examples of OFS.GLOBUS.MANAGER in Use

ROUTINE/APPLICATION	Purpose
LC.INWARD	To perform additional updates for inward message types 700, 701, 705, 707
AZ.UNAUTH.PROCESSING	Deletes unauthorised AZ, TT & FT records during COB
AA.PROCESS.CONTRACT	Handles OFS processing for AA

In scenarios where OFS.GLOBUS.MANAGER is already loaded in the session, the request is written into the queue specified in the OFS.SOURCE record under IN.QUEUE.DIR field.

O.G.M may only be used if the write cache is empty.

O.G.M builds up its own OFS.SOURCE record if the OFS.SOURCE record id argument is not supplied.



Syntax:

OFS.POST.MESSAGE(OFS.MESSAGE, OFS.MSG.ID, OFS.SOURCE.ID, OPTIONS).

Parameters:

OFS.MESSAGE is an in parameter which contains the OFS message to be processed.

OFS.MSG.ID is an out parameter which passes back a auto generated message reference.

OFS.SOURCE.ID is an in parameter which contains a OFS source record id of type Globus.

Options is reserved for future use.

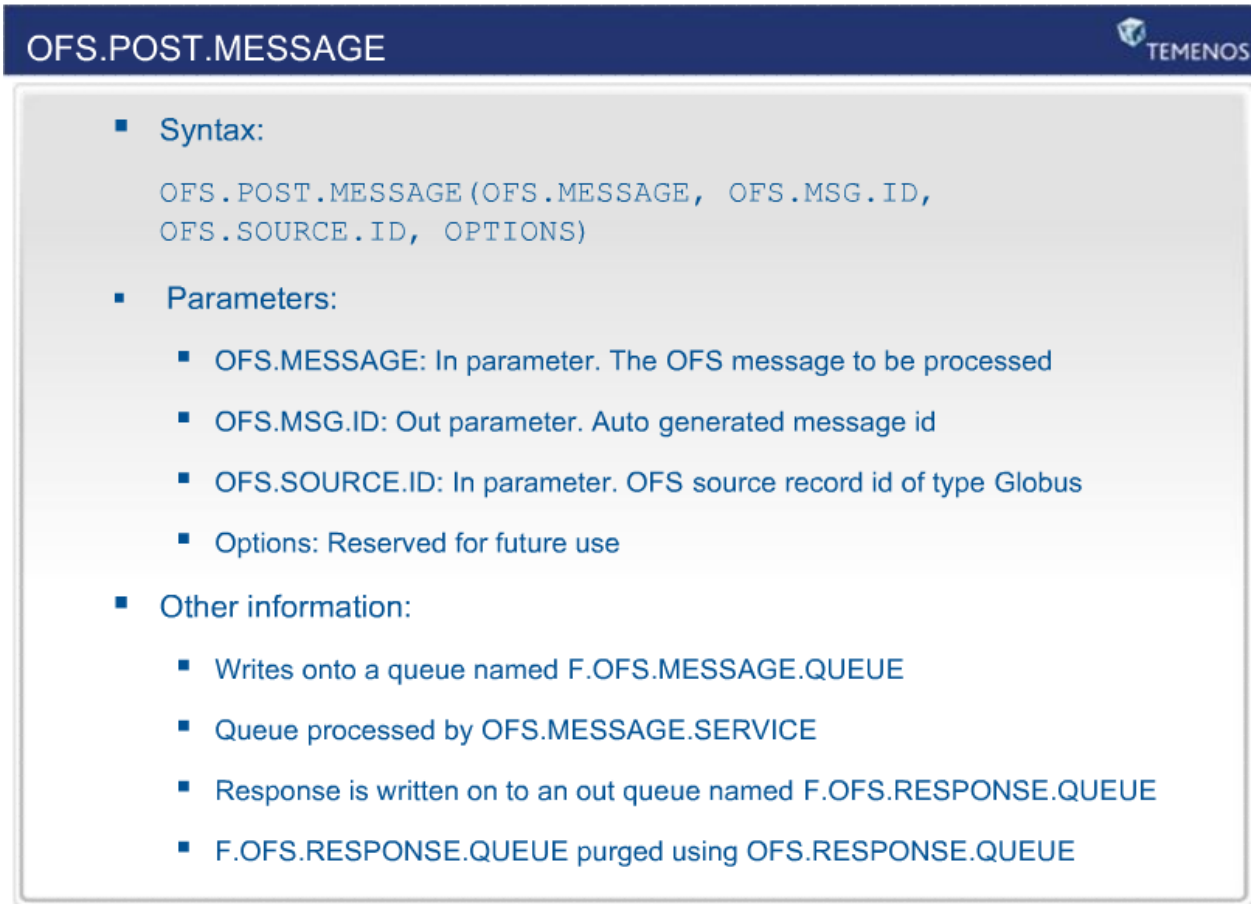
Other information:

OPM writes the message onto a queue named F.OFS.MESSAGE.QUEUE. The id of the record is returned by the second parameter in OPM the OFS.MSG.ID.

The queue is processed by a service named OFS.MESSAGE.SERVICE.

The service picks up the message, passes it to OFS.REQUEST.MANAGER for executing and eventually writes the response to an out queue (well actually a file) named F.OFS.RESPONSE.QUEUE.

The responses in F.OFS.RESPONSE.QUEUE are not cleared automatically. They may be purged by using another service named OFS.RESPONSE.QUEUE.



The message id is auto generated.

It follows the format YYYYMMDDUUUUUSSSSS.NN where,

YYYY = year

MM = month

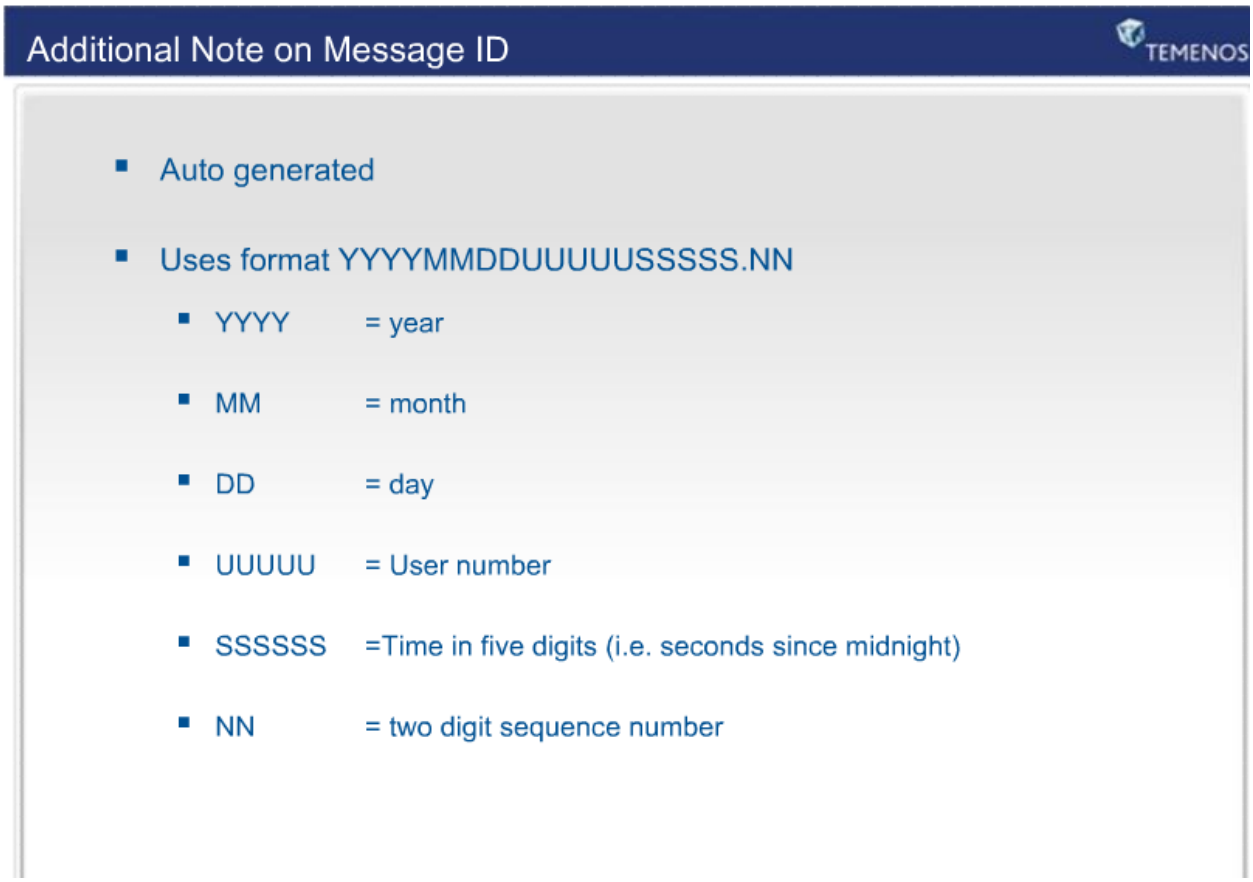
DD = day

UUUUU = User number

SSSSSS =Time in five digits (i.e seconds since midnight)

NN = two digit sequence number

The last 3 portions of the message id are generated by a T24 routine called ALLOCATE.UNIQUE.TIME whereas the date is returned by a jBase library called DATE.



F.OFS.RESPONSE.QUEUE contains a response that is in three parts, well fields actually:

Field 1:

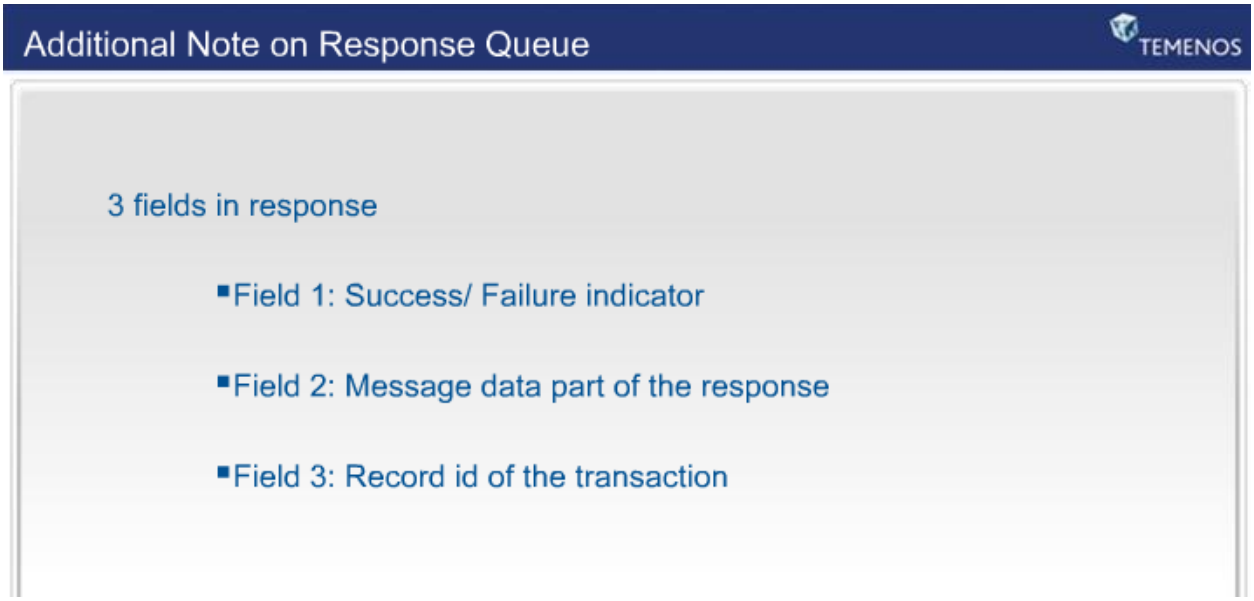
This contains the success of failure indicator . This if you recall could be -1,1,2 or 3.

Field 2:

This field contains the message data part of the response, i.e. the field names and values with which they have been populated.

Field 3:

Contains the record id of the transaction that was committed.



Some example of OFS.POST.MESSAGE been used within T24 are:

STO.GEN.EXP.RECS:

Updates the application AC.EXPECTED.RECS during authorisation of an STO.

FX.CREATE.RECORDS:

This routine is called when authorising FX.ORDER. Used for bulk FX.

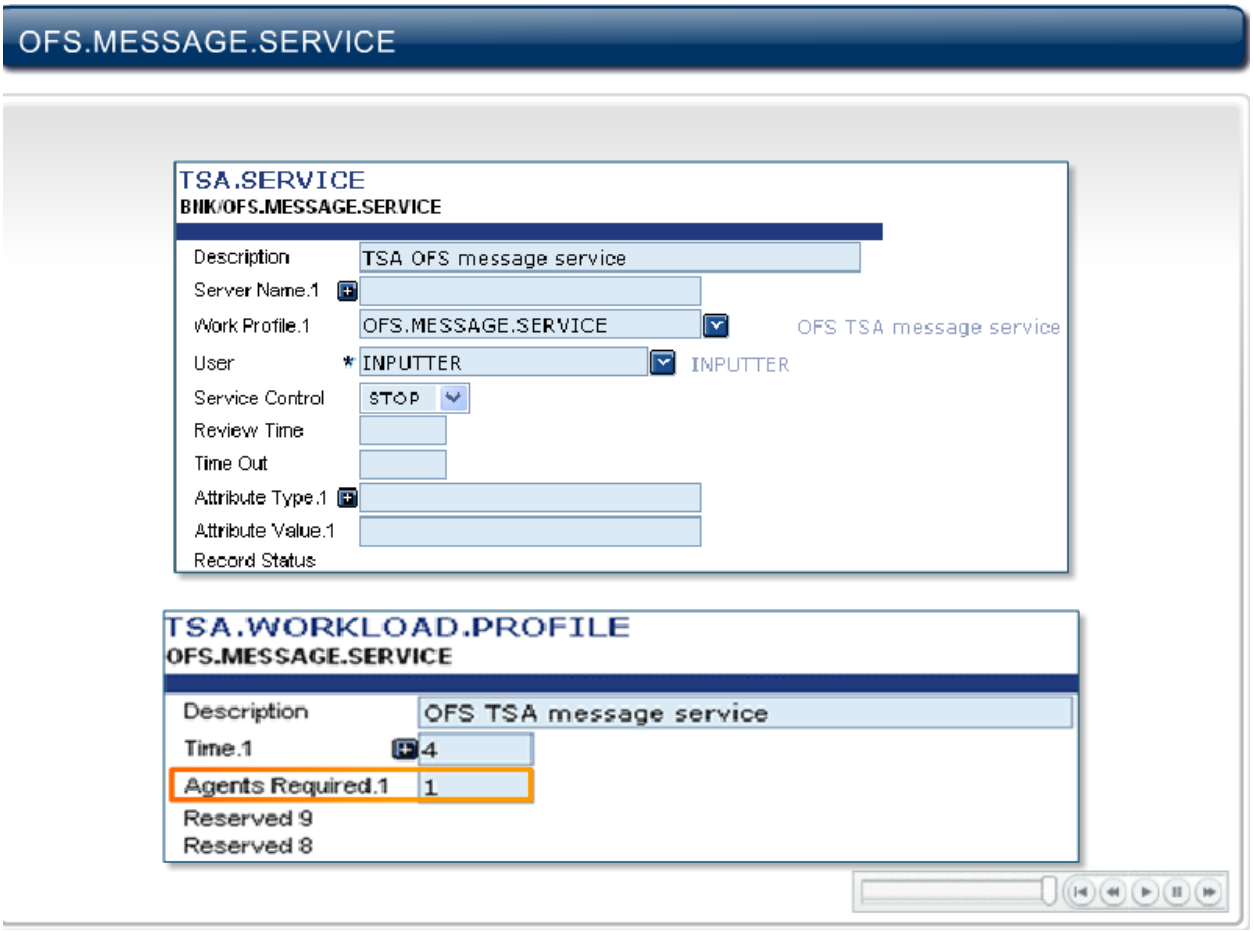
CR.OPPORTUNITY.PROCESSING:

This routine is called during COB. OPM builds PW related records for CR.

Example of OFS.POST.MESSAGE in Use

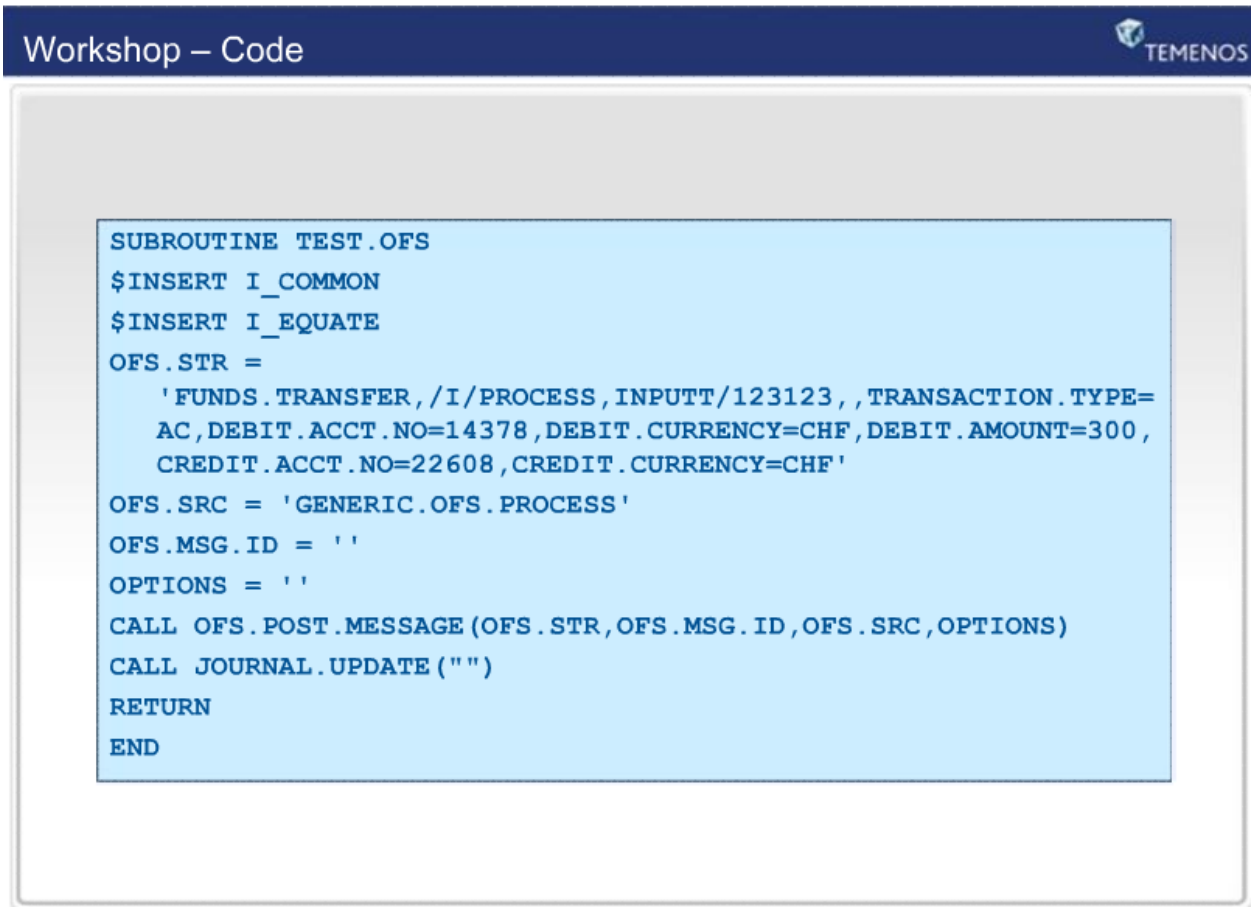
ROUTINE/APPLICATION	Purpose
STO.GEN.EXP.RECS	Updates the application AC.EXPECTED.RECS during authorisation of an STO
FX.CREATE.RECORDS	Called when authorising FX.ORDER. Used for bulk FX
CR.OPPORTUNITY.PROCESSING	Builds PW related records for CR during COB

OFS.MESSAGE.SERVICE is a standard T24 service. It's TSA.SERVICE entry is shown here. It has it's own workload profile, where the no of agents are usually set to 1.

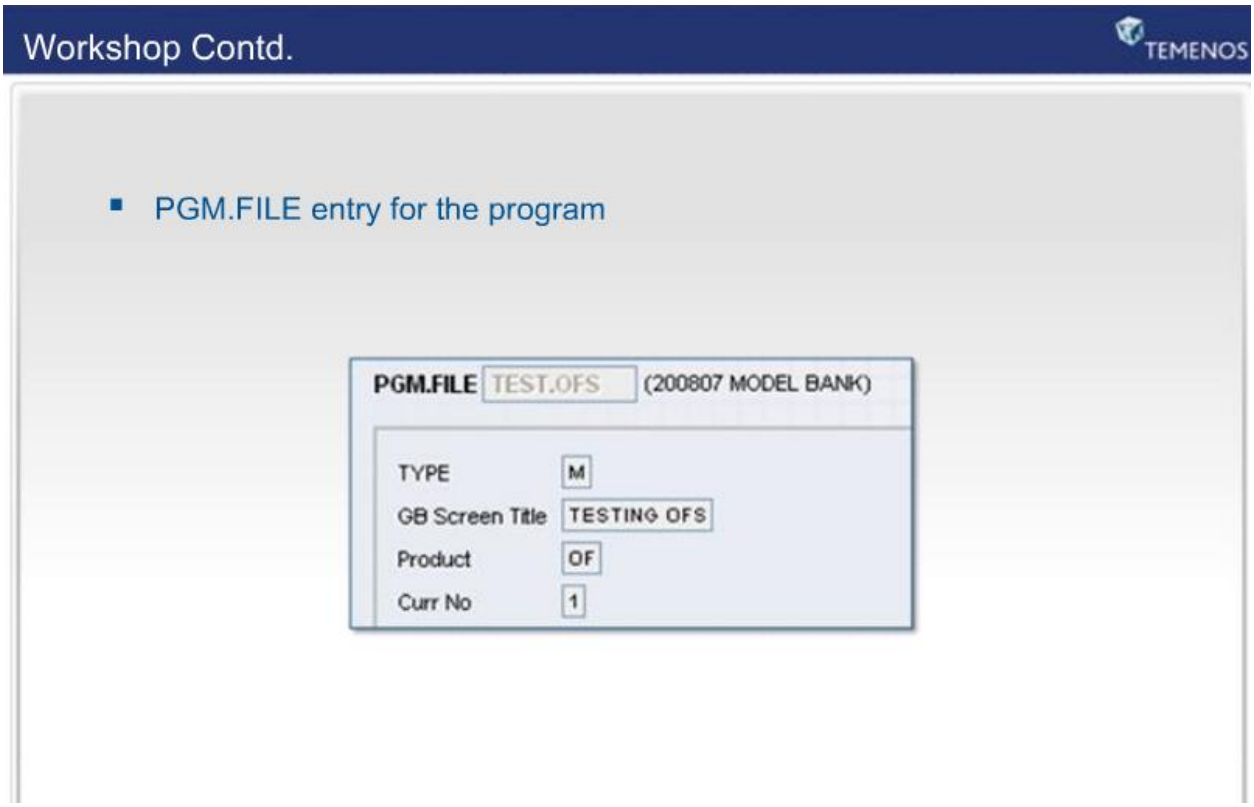


Given here is a mainline routine to test OFS.POST.MESSAGE. Please note that such kind of mainline routines may not be practically useful. However it serves to illustrate OFS.POST.MESSAGE. Since this is a mainline routine, we are forcing a transaction block by calling JOURNAL.UPDATE.

Note: Check if you have the same Account numbers in your area before trying this code.

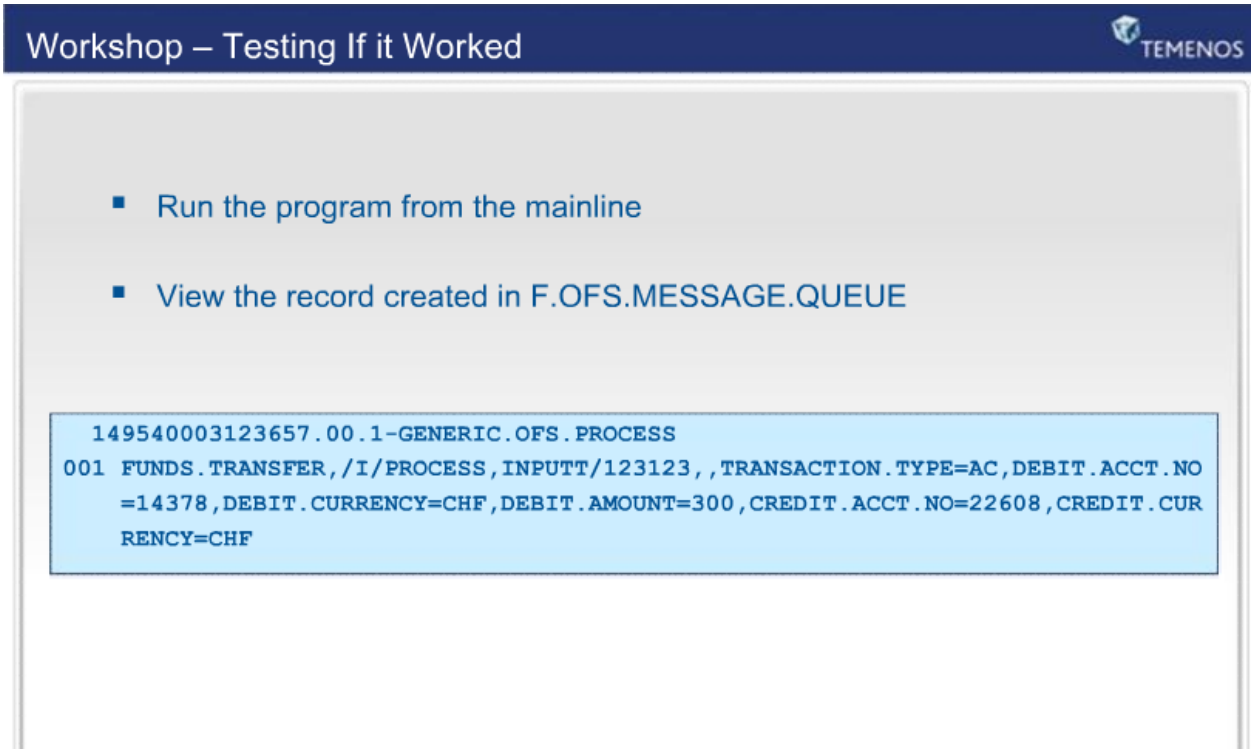


This page shows the PGM.FILE entry for the program.



Let us execute the program from the main line. What would have happened when you ran the program?

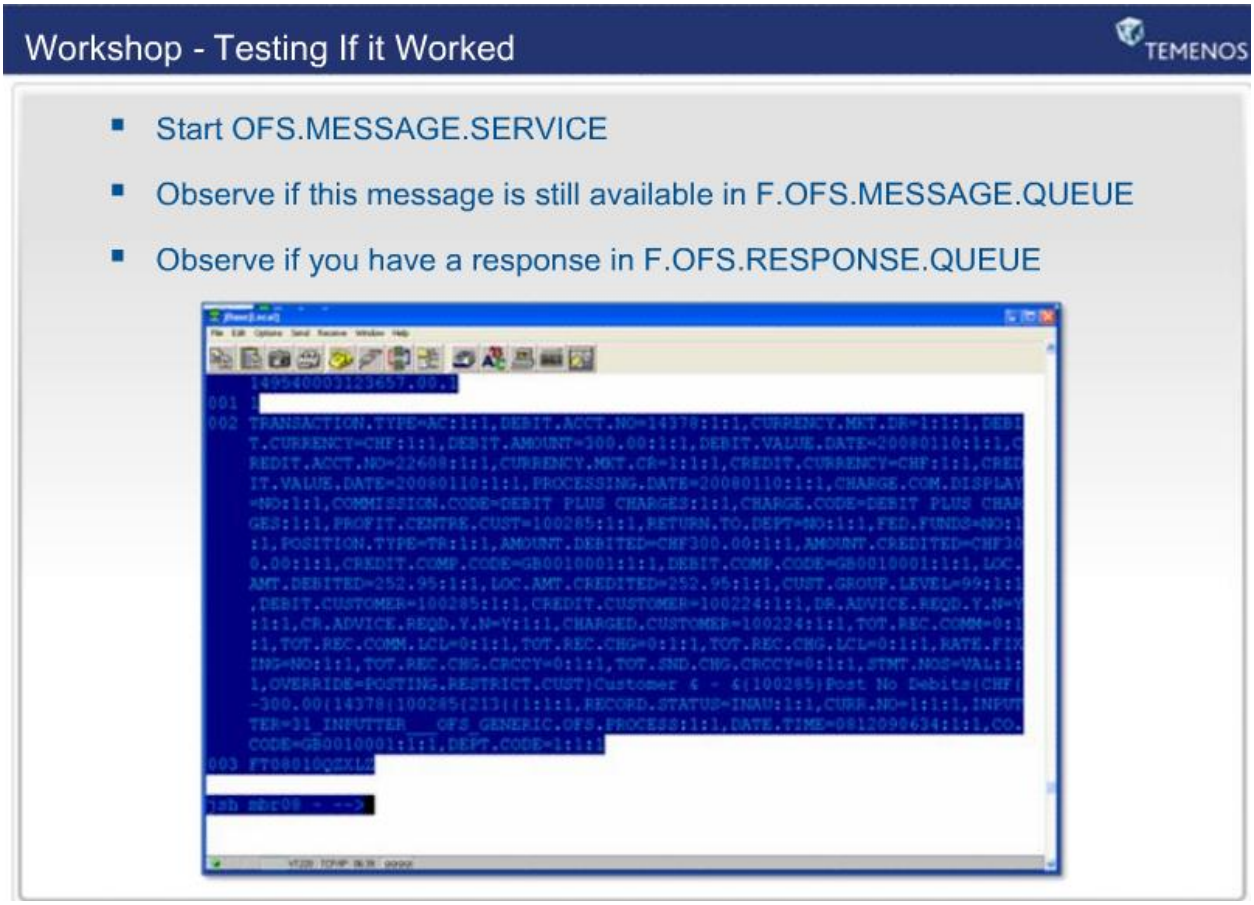
The message would have been written onto the queue F.OFS.MESSAGE.QUEUE. You can check the queue by typing CT F.OFS.MESSAGE.QUEUE at the jshell prompt.



What next?

You need to get your message processed by starting the service called OFS.MESSAGE.SERVICE . If the service has picked up the message, it will no longer be in the queue.

Check if the queue still contains your message. Check if you have a response in F.OFS.RESPONSE.QUEUE.



Purpose:

OFS.BUILD.RECORD is a routine that builds up a OFS message

Syntax:

OFS.BUILD.RECORD (APP.NAME, OFSFUNCT, PROCESS, OFSVERSION, GTSMODE, NO.OF.AUTH, TRANSACTION.ID, RECORD, OFSRECORD)

Parameters:

APP.NAME - Application Name

OFSFUNCT - The function (This could be I, A, R or D)

PROCESS - should always be PROCESS or BR.PROCESS

OFSVERSION - holds the version name. Where a version is not used specify application name followed by comma.

GTSMODE - GTS Control value

NO.OF.AUTH - the no of authorisers

TRANSACITON.ID - transaction id. This is mandatory if function is A, R or D

RECORD - data in dynamic array format

OFSRECORD - out parameter which contains the OFS message

Other information:

No provision for user information.

This routine is almost identical to the previous except for the use of OFS.BUILD.RECORD. In OFS.BUILD.RECORD, we do not pass the user info as a parameter to this routine.

Then Which User info does it take?

BATCH MODE - GTS User Id in EB.PHANTOM.

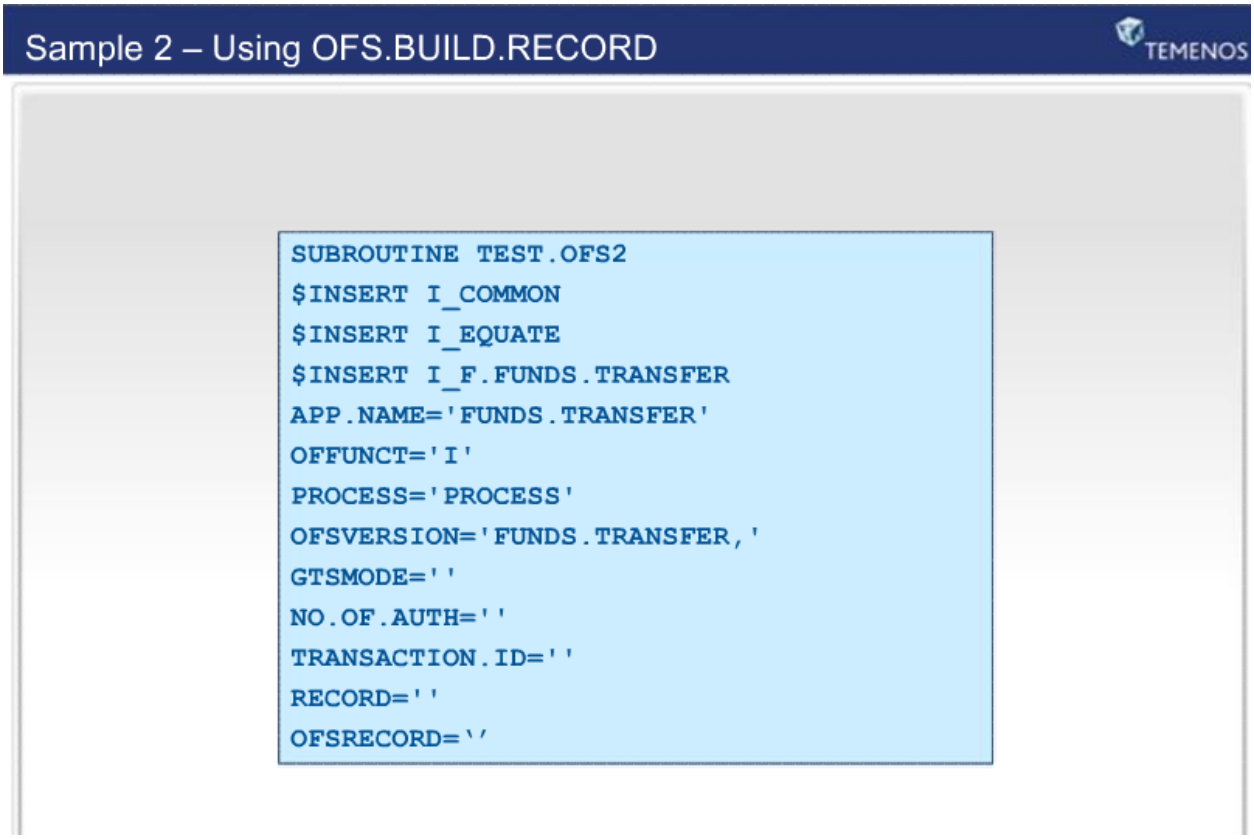
GLOBUS MODE - Generic User field in OFS.SOURCE , if OFS.GLOBUS.MANAGER is used.

User field from TSA.SERVICE , if OFS.POST.MESSAGE is used.

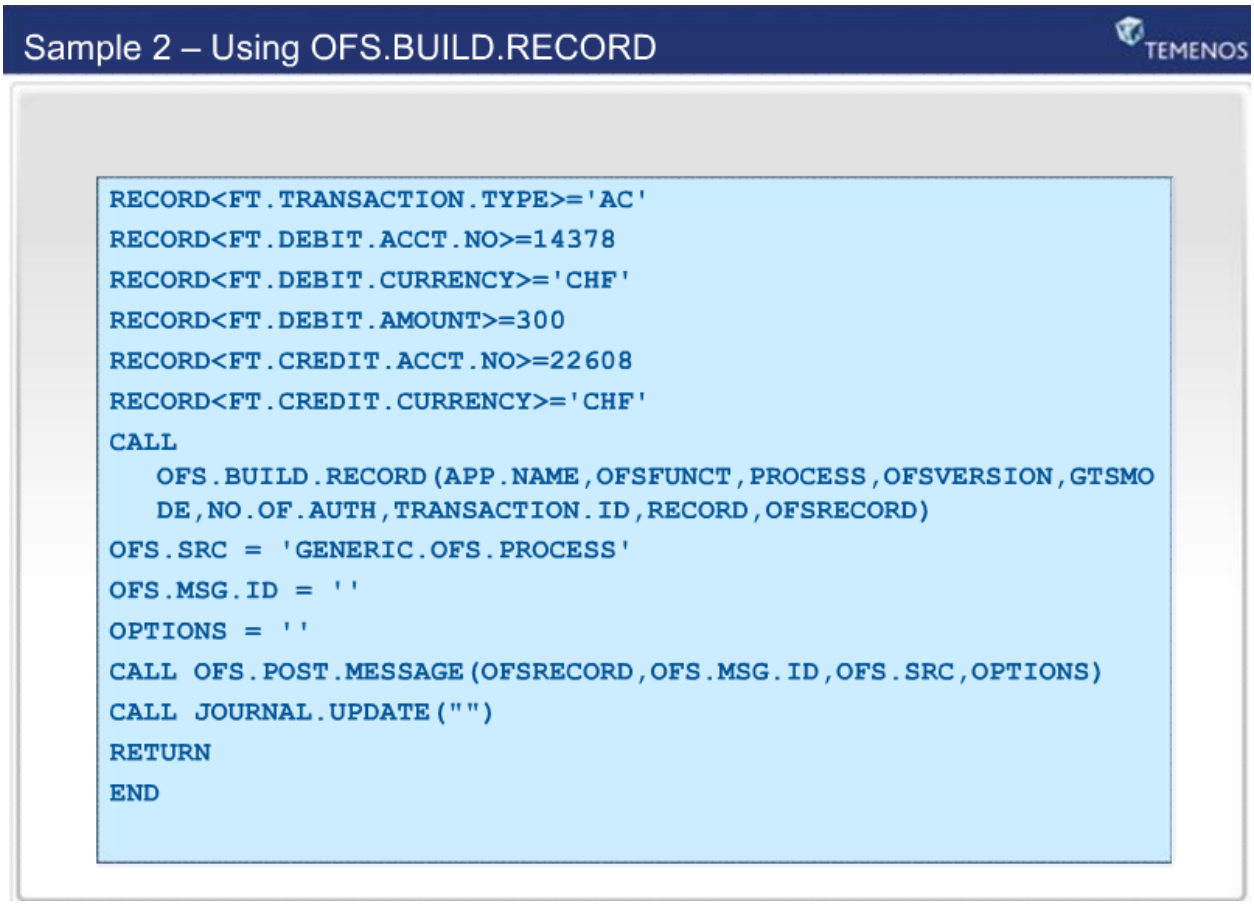
TELNET MODE - Throws an error when OFS.BUILD.RECORD is used.

SESSION MODE - T24 login User.

The main purpose of this routine is to be used in the GLOBUS POST MESSAGE mode.



Test this routine as before.

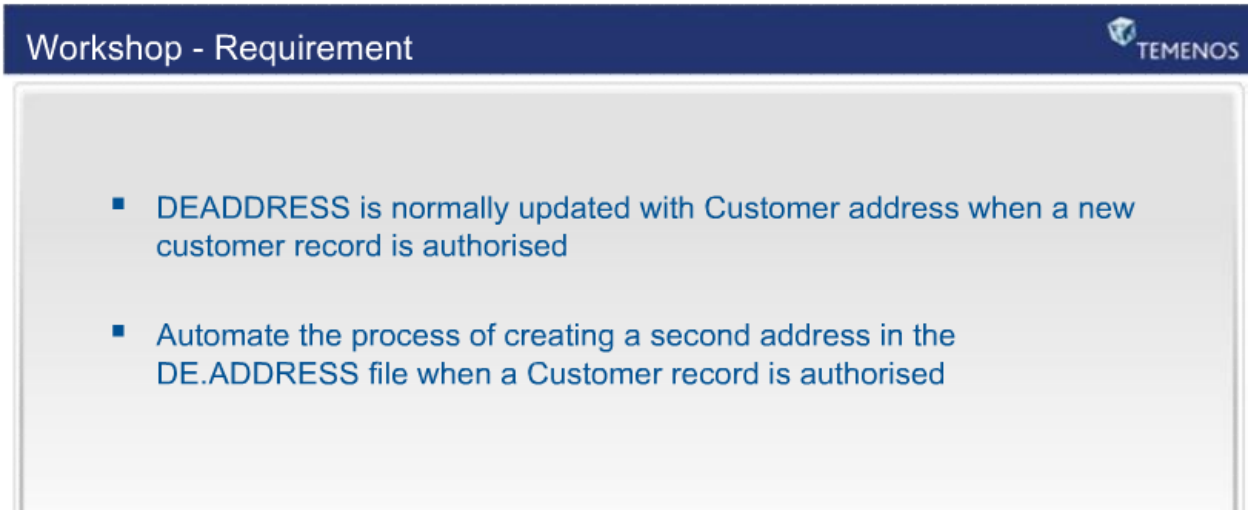


Let us try to understand and use OFS.POST.MESSAGE in a simple practical scenario:

As you may know, DE.ADDRESS is normally updated with the Customer address when a new customer record is authorised.

But what if we wanted the facility to input a second address in the DE.ADDRESS file when a Customer record is authorised?. This means that you need to input the second address into another field and have your own way of writing that into DE.ADDRESS since the core T24 code has the provision for adding only a single address per customer.

What could be the algorithm for this?.



Algorithm:

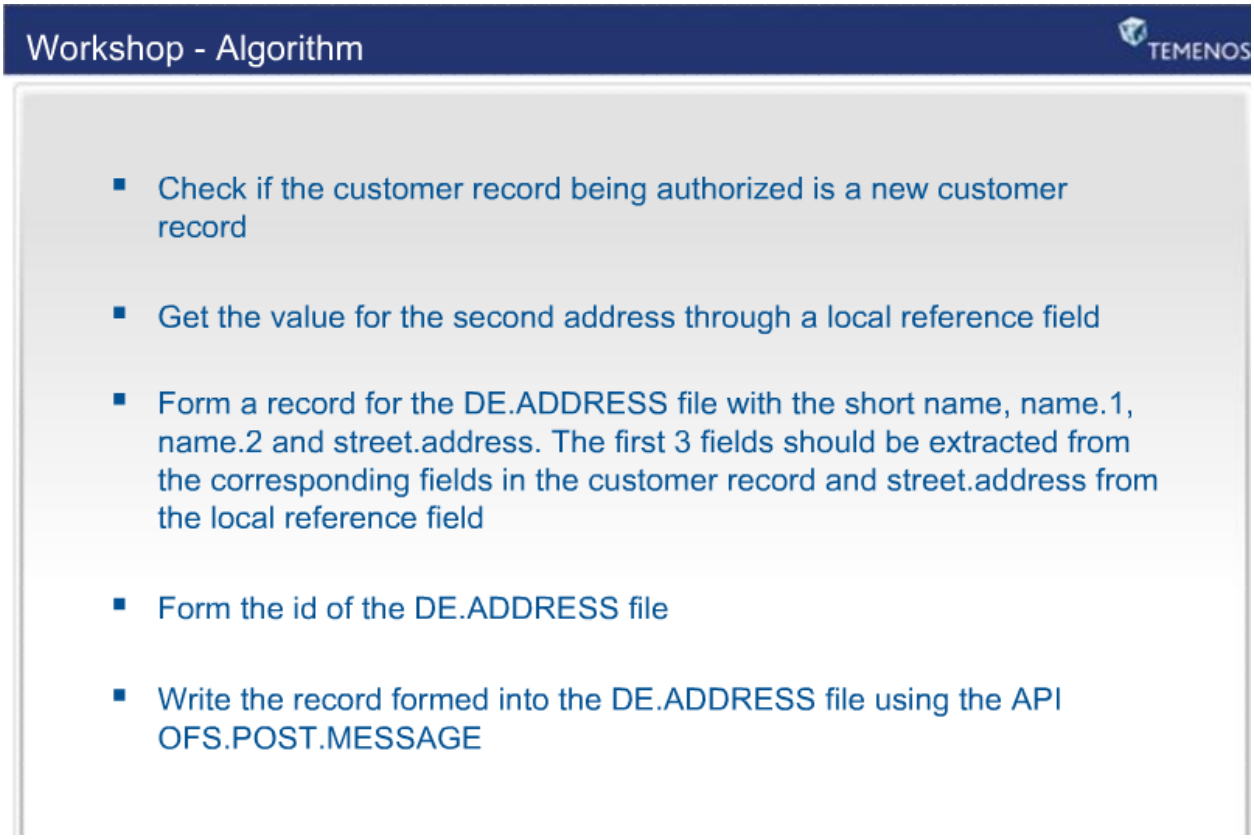
Check if the customer record being authorized is a new customer record.

Get the value for the second address through a local reference field.

Form a record for the DE.ADDRESS file with the short name, name.1, name.2 and street.address. The first 3 fields should be extracted from the corresponding fields in the customer record and street.address from the local reference field.

Form the id of the DE.ADDRESS file.

Write the record formed into the DE.ADDRESS file using the API OFS.POST.MESSAGE.



DE.ADDRESS records have ids usually created in the following format:

company-id.C-customer-id.PRINT.1

We need to create an id in the format:

Companycode.C-CustomerNo.PRINT.2

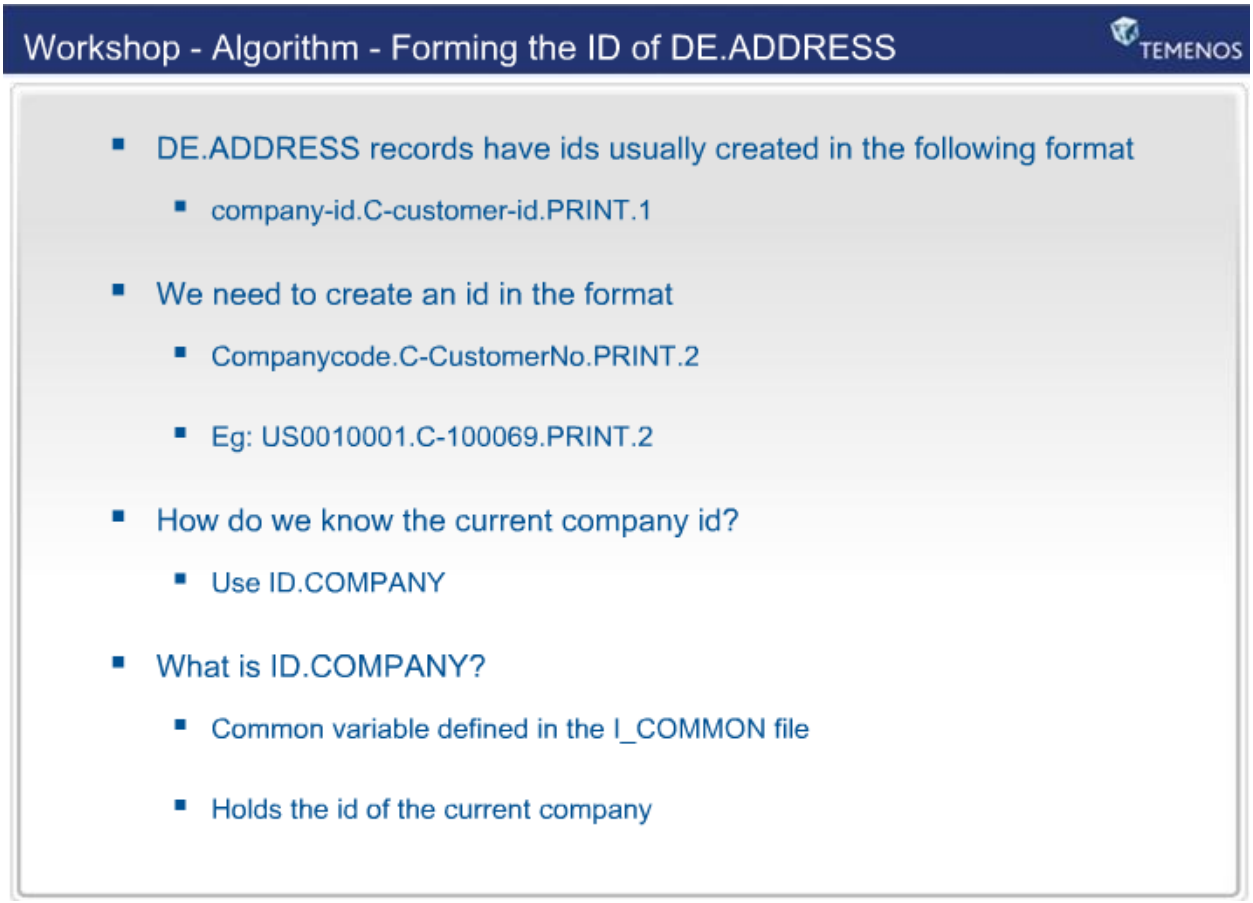
Eg: US0010001.C-100069.PRINT.2

But how do we know the current company id?. We can use ID.COMPANY.

What is ID.COMPANY?.

It is a Common variable defined in the I_COMMON file.

It holds the id of the current company.



The code for the routine is shown here.

Workshop – Code

```
 SUBROUTINE V.TRG.AUTH.RTN
    $INSERT I_COMMON
    $INSERT I_EQUATE
    $INSERT I_F.CUSTOMER
    $INSERT I_F.DE.ADDRESS

    GOSUB INIT
    GOSUB OPENFILES
    GOSUB PROCESS
    RETURN

INIT:
    FN.CUS = 'F.CUSTOMER'
    F.CUS = ''
    Y.ADDRESS = ''
    RETURN

OPENFILES:
    CALL OPF(FN.CUS,F.CUS)
    RETURN
```

The code for the routine is continued in this page. Take care on the local reference field. For instance, here we assume that it is the seventh local ref field and hence we use the statement R.NEW(EB.CUS.LOCAL.REF)<1,7>. You need to key in the correct position as per your area.

Workshop – Code Contd.

```
PROCESS:
   IF R.OLD(1) = '' THEN    ;* If it is a new record being authorised

   * Form the id of the DE.ADDRESS record

   Y.DE.ADDRESS.ID = ID.COMPANY:'.C-':ID.NEW:'.PRINT.2'
   Y.ADDRESS = R.NEW(EB.CUS.LOCAL.REF)<1,7>

   Y.OFS.REC="DE.ADDRESS,,INPUTT/654321,":Y.DE.ADDRESS.ID:",SHORT.NAME
   =":R.NEW(EB.CUS.SHORT.NAME):",NAME.1=":R.NEW(EB.CUS.NAME.1):",NAME.
   2=":R.NEW(EB.CUS.NAME.2):", STREET.ADDR=":Y.ADDRESS

   OFS.MSG.ID=""
   OFS.SOURCE.ID="TRG.OFS.GLOBUS"
   OPTIONS=""

   CALL OFS.POST.MESSAGE (Y.OFS.REC, OFS.MSG.ID, OFS.SOURCE.ID,
   OPTIONS)

   END

RETURN
END
```

How can we test this? These are the steps:

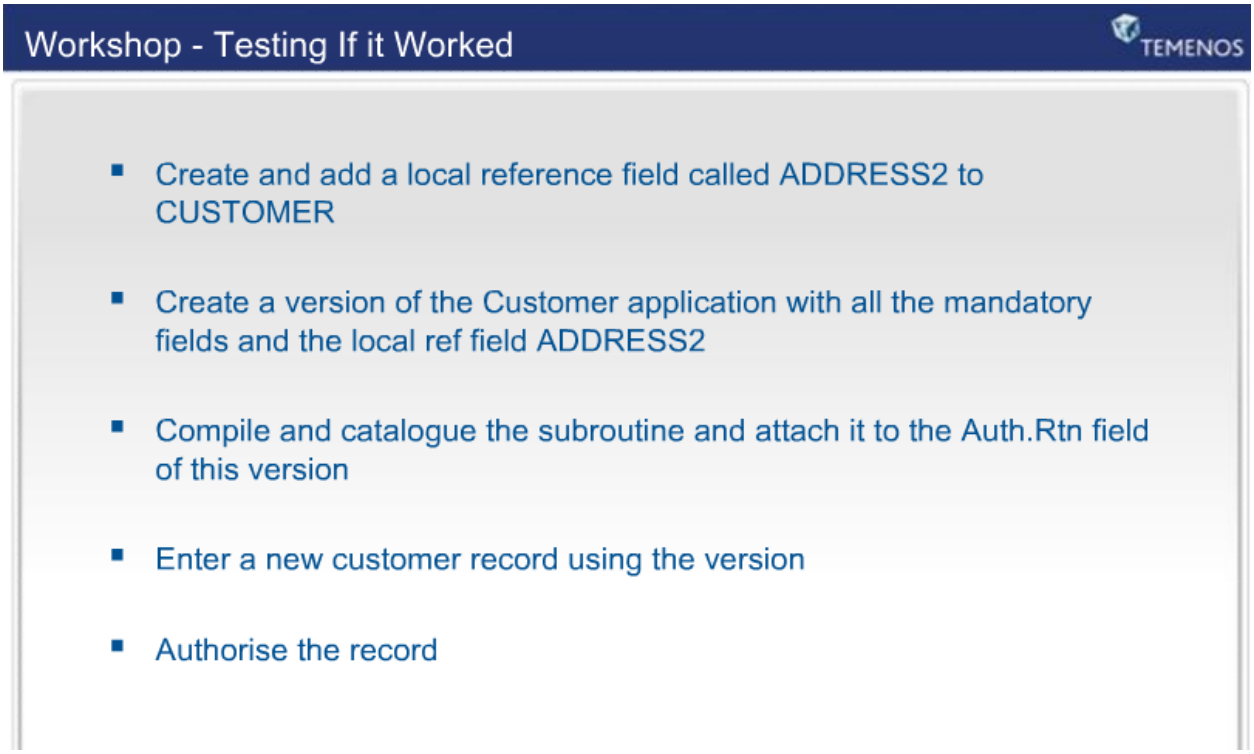
a. Create and add a local reference field called ADDRESS2 to CUSTOMER

b. Create a version of the Customer application with all the mandatory fields and the local ref field ADDRESS2

c. Compile and catalogue the subroutine and attach it to the Auth.Rtn field of this version

d. Enter a new customer record using the version

e. Authorise the record



Check if OFS.MESSAGE.QUEUE has been updated.

Start the following services:

- TSM
- OFS.MESSAGE.SERVICE

Check if the message has been removed from F.OFS.MESSAGE.QUEUE.

Check if you have the response in F.OFS.RESPONSE.QUEUE.

Workshop - Testing If it Worked Contd.

- Check if OFS.MESSAGE.QUEUE has been updated
- Start the following services:
  - TSM
  - OFS.MESSAGE.SERVICE
- Check if the message has been removed from F.OFS.MESSAGE.QUEUE
- Check if you have the response in F.OFS.RESPONSE.QUEUE

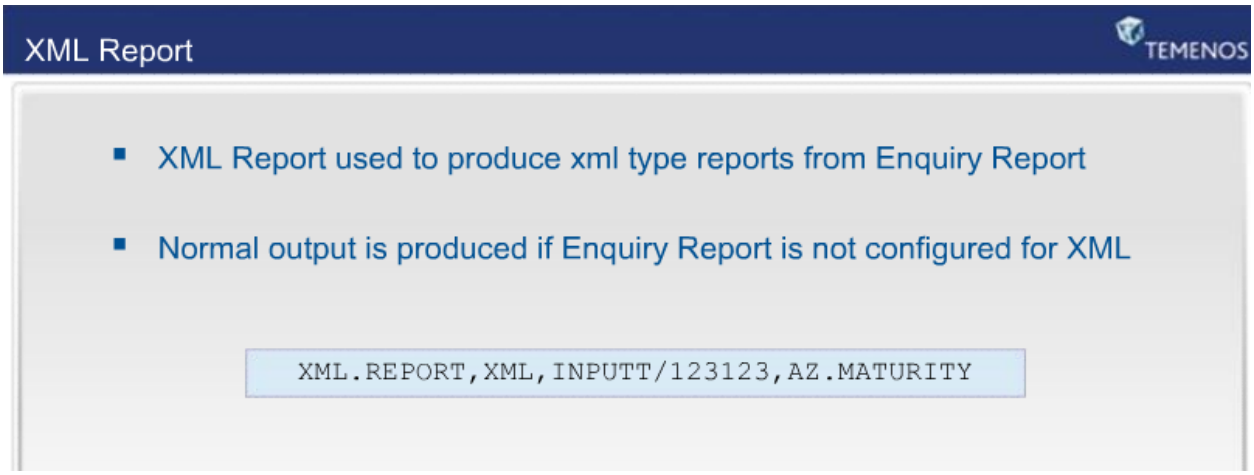
Important Points

- The two ways of making inter-application calls are by using:
  - OFS.GLOBUS.MANAGER
  - OFS.POST.MESSAGE
- OGM is synchronous
- OGM cannot be used from local routines
- OPM is asynchronous
- OPM writes into a queue F.OFS.MESSAGE.QUEUE
- The messages in the queue are process by a service called OFS.MESSAGE.SERVICE
- This service internally calls OFS.PROCESS.MANAGER

Xml reporting facility

The XML Report type request is used to produce reports in xml format. It uses Enquiry Report as the base for building the report. The particular Enquiry report and supporting records in other applications (such as DE.FORM.TYPE) must be configured to produce xml output. A normal report is produced if Enquiry Report is not configured for XML.

Did you notice the message format?. It starts with XML.REPORT, has only XML in the options part, function is missing, and ends with an Enquiry Report id. Rather similar to Enquiries isn't it?.



This diagram describes the syntax of a XML Report type request.

Operation:

This will always be XML.REPORT

Options: This may XML or ID.

XML – Specify this to view the xml report directly.

ID – Specify this to the get id of output file in &HOLD& as the response.

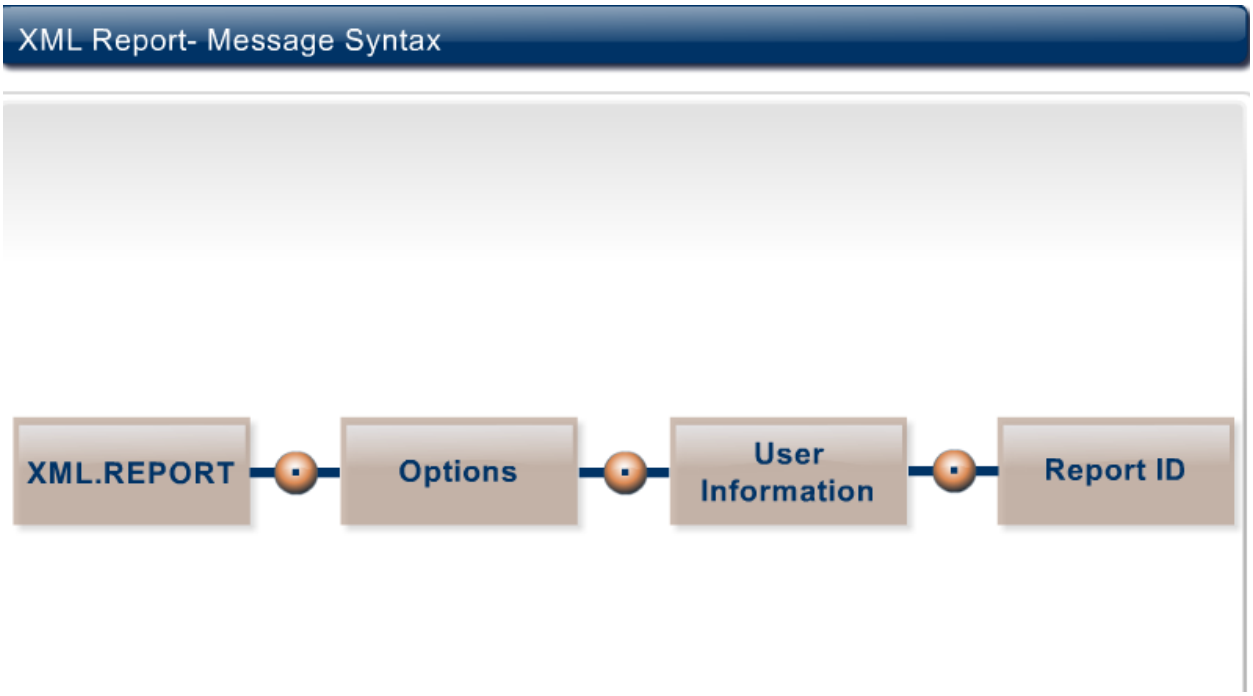
User Information:

This consists of the Sign On Name/Password/Company code.

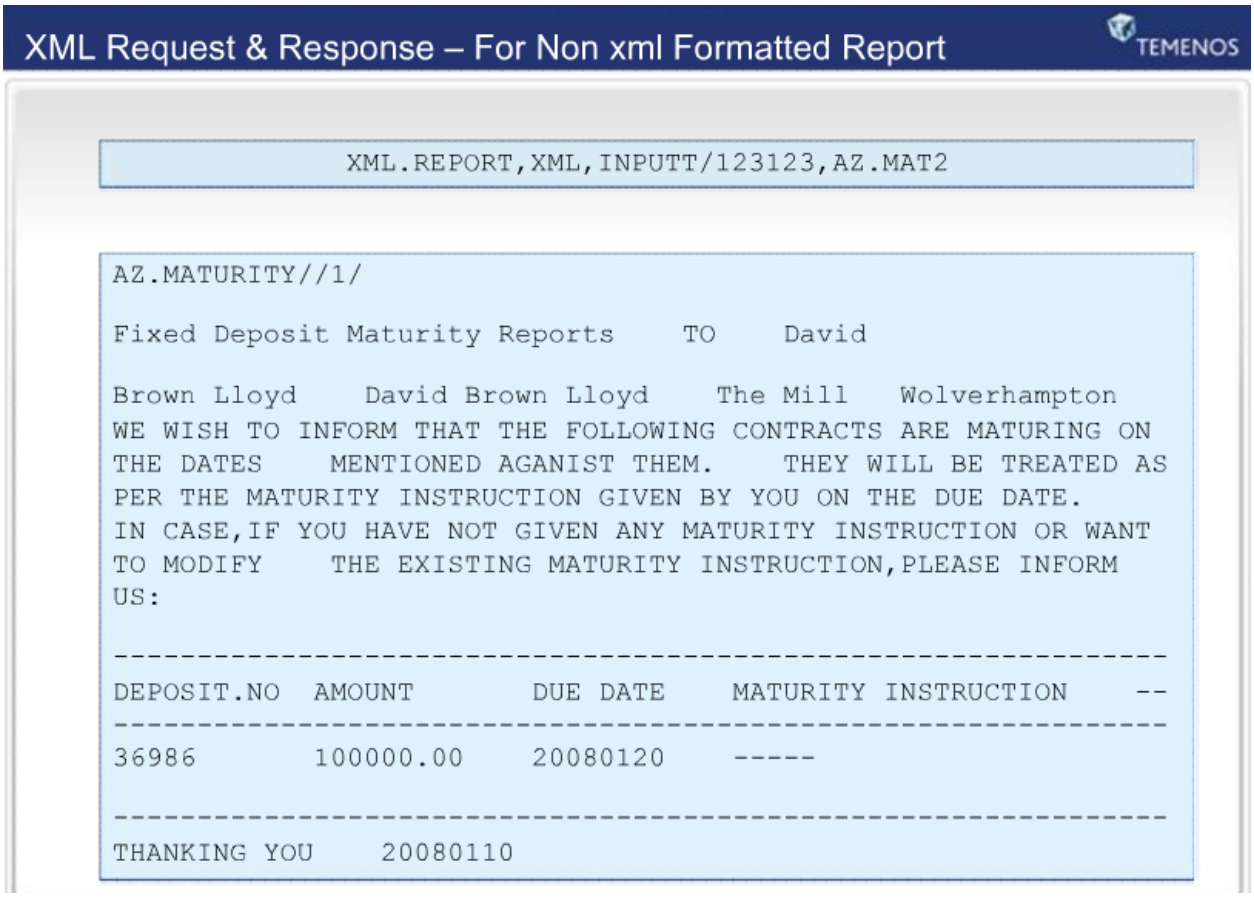
E.g INPUTT/123456/GB0010

Report Id:

This must be the id of the Enquiry Report that is used to produce the XML Report.



In this page you can see the XML Report request and response. Since the Enquiry Report AZ.MAT2 has not been configured for XML output, you see a ordinary kind of report.



We need to configure/create entries in the following applications and files to get XML format report.

ENQUIRY.REPORT

ENQUIRY

REPORT.CONTROL

DE.FORM.TYPE

PRINTER.ID

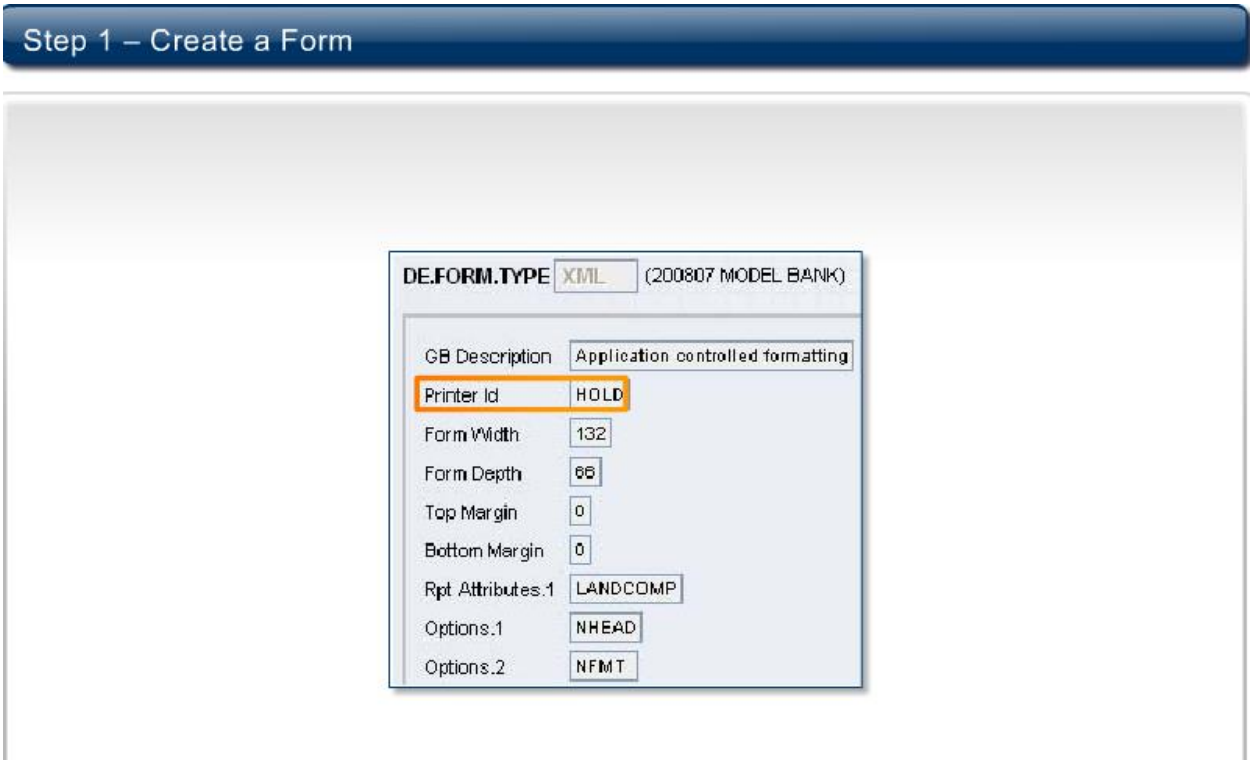
A non-hashed file

REPORT.TRANSFORM

Lets try to create a ENQUIRY.REPORT for the AZ Accounts which are reaching maturity. There is a existing enquiry named AZ.MATURITY that we can use to call out the required data.

Start by creating a DE.FORM.TYPE record. The sample shows one created with an id of XML. Of course you can call it anything you like. Printer id used is the special HOLD. The id used for DE.FORM.TYPE cannot be an existing DE.CARRIER record.

Step 1 – Create a Form



We need a folder to store the xml output data as well as error messages. Create a file within the bnk.run folder and call it as XML.LOCAL. Note – you may call it anything you like.

Secondly, create a record within this file say with an id TRANSLATE.XLS. (Again you may call this anything you like ….).

Create a record in REPORT.TRANSFORM. The example here shows one with an id of XML.TRANSFORM. Enter the file that you created in the Trans Path, Report Dest and Error Dest fields. Enter the filename in the Transname field.

Step 3 – Point to the Folders

You will now use Report Control to link the form, transformation paths and the transformation context together.

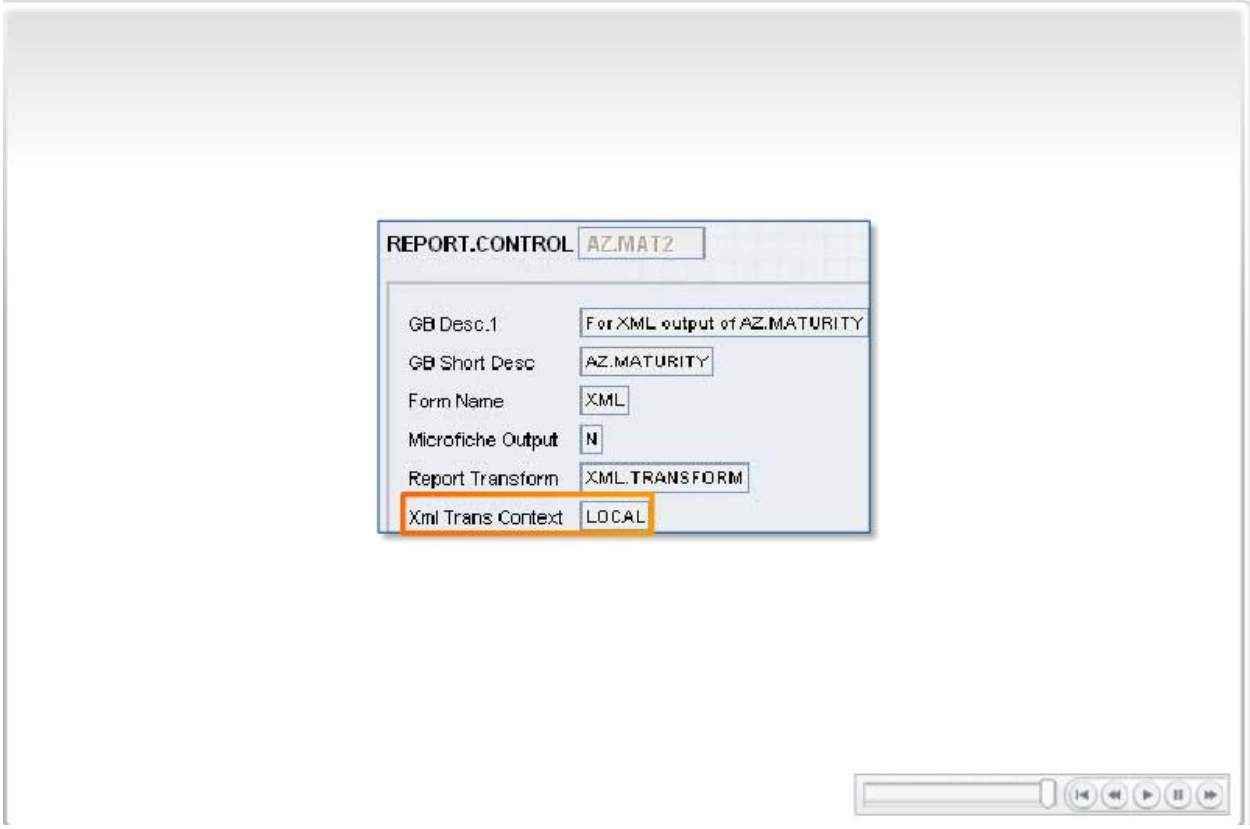
The important fields are given here. Each of these fields link to another application which is named within braces.

Form Name - XML (DE.FORM.TYPE)

Report Transform - XML.TRANSFORM (REPORT.TRANSFORM)

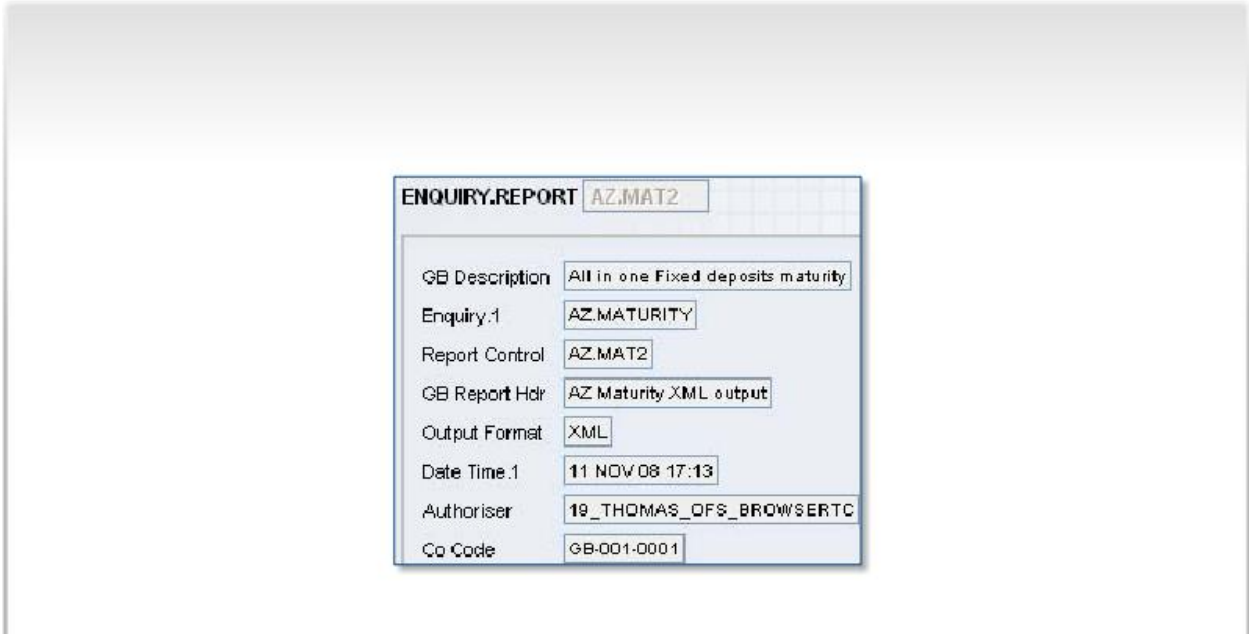
Xml Trans Context - LOCAL (XML.TRANS.CONTEXT)

Step 4 – Create a Report Control Record



The enquiry report binds the enquiry and the report control together. Notice the corresponding entries in the Enquiry field i.e. AZ.MATURITY, Report Control field i.e. AZ.MAT2. Remember to select the output format as XML.

Step 5 – Create the Enquiry Report



Verify the Enquiry Report.

A new report will be created and stored in &HOLD&.

You may get the Report Id from HOLD.CONTROL.

View the output from the jshell.

Ofs message for local clearing

Let's first see what is OFS.CLEARING?

OFS.CLEARING is an interface that is used to create accounting entries directly from a OFS string

The OFS message that is sent to the clearing interface follows a specific format

OFS.CLEARING uses an API called OFS.CLEARING.MANAGER to process messages and raise appropriate accounting entries You can see a sample CLEARING message in the page above. Shall we look at what each part means?

What is OFS Clearing?

- OFS.CLEARING interface to crate accounting entries directly using OFS
- Specific message format
- Uses an API called OFS.CLEARING.MANAGER to process messages and raise appropriate accounting entries.

```
CLEARING,,INPUTT/123123,LCLG,10,14621~SALARY~7000~C
```

This diagram describes the syntax of a CLEARING type request.

Operation

This must always be CLEARING

User Information

This contains T24 user information (id, password, company etc)

Transaction Id

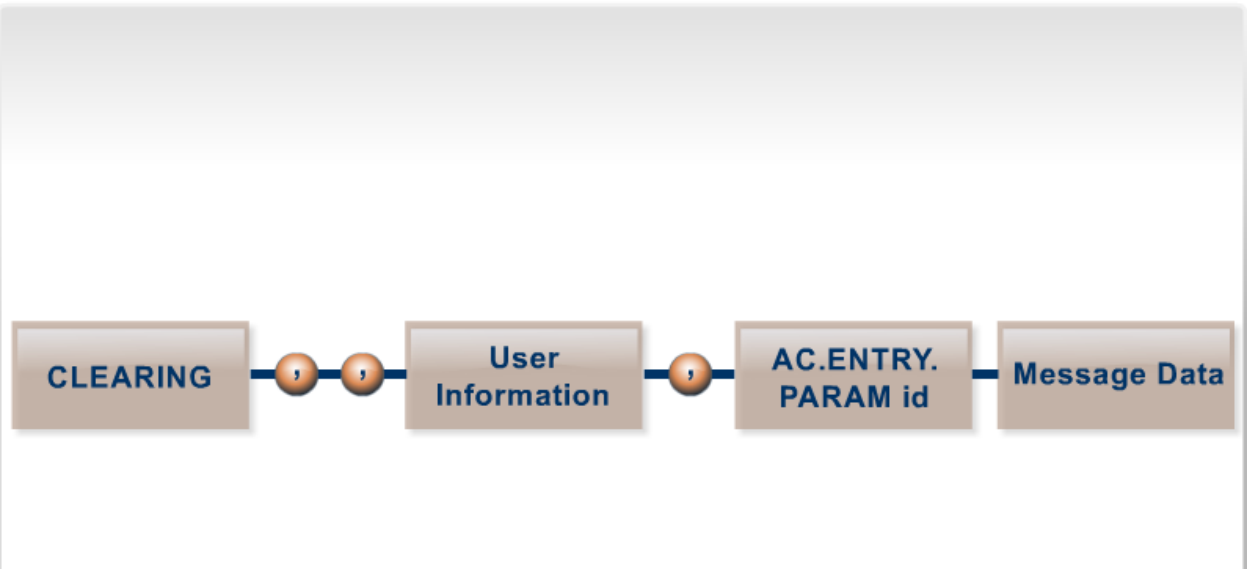
The id of a record in AC.ENTRY.PARAM

Message Data

This contains the Data values for the transaction alone. It does not contain the field names. The structure of the data to be supplied is described in a AC.ENTRY.PARAM record whose id has been supplied as the transaction id.

Note – OFS Clearing does not give a response message.

OFS Clearing Message Request Syntax



What do we need to use OFS.CLEARING? We need two things;

An entry in AC.ENTRY.PARAM. This spells out the layout of the data portion of the message

An OFS message following the syntax for OFS Clearing.

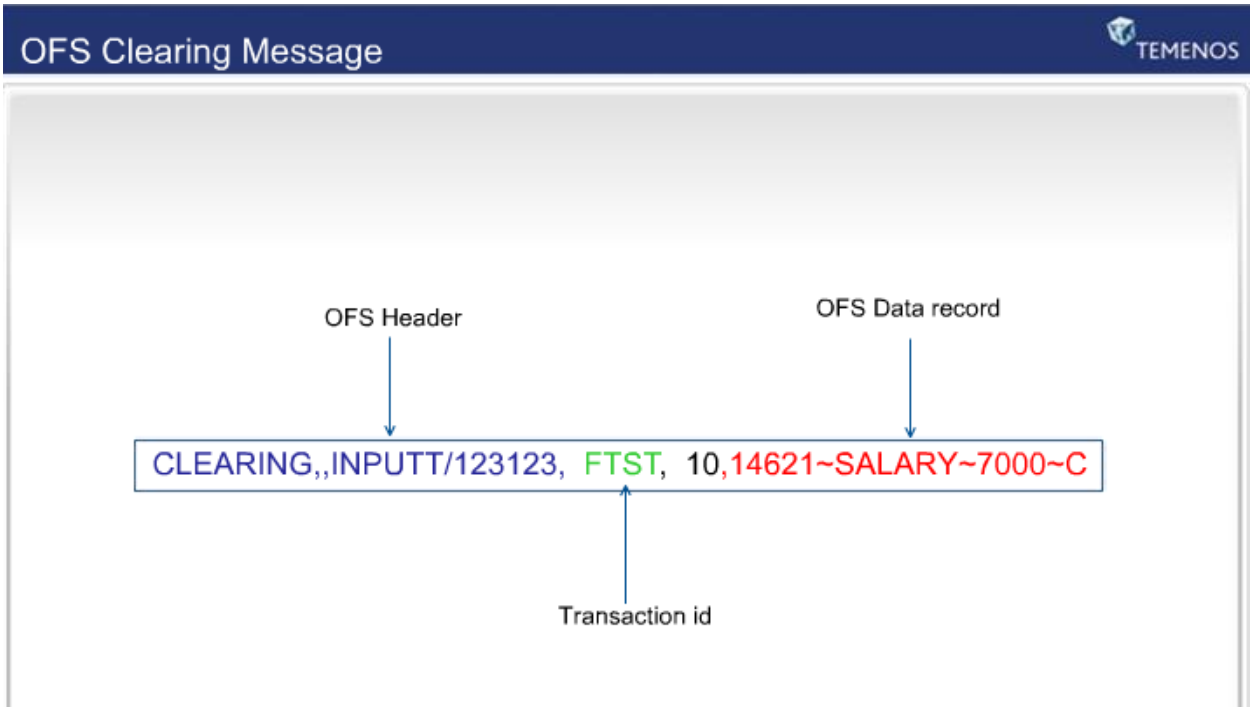
Note – Data for Clearing could be supplied through a file as well. However this course deals with only the OFS message for clearing.

What is required for OFS.CLEARING?

TEMENOS

- An entry in AC.ENTRY.PARAM
- A message

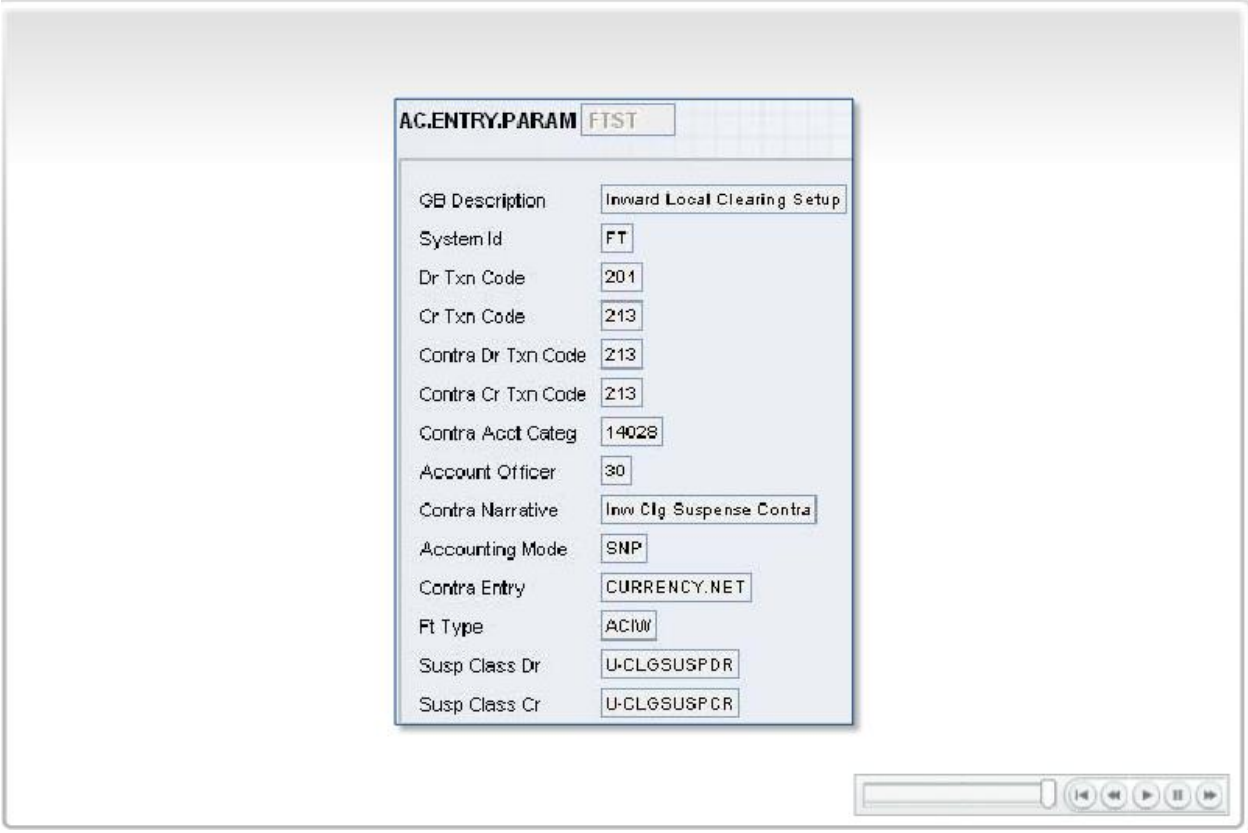
The OFS Clearing message can be classified into three portions the OFS Header, Transaction id and the OFS Data record.



AC.ENTRY.PARAM is a T24 parameter file, that defines the layout of the message and options for booking entries, used during the clearing process.

Discussion of each field here shown here on this page is beyond the scope of this course. Refer to the Course on Funds Transfer/Local clearing for a detailed description.

AC.ENTRY.PARAM



If you observe an OFS Clearing message, unlike other OFS transactions, the data part of the message does not contain field names. The fields and their positions within the data part of the OFS message are defined in AC.ENTRY.PARAM.

Field Delim : this contains the character that is used to delimit each field in the message data part of the message. This could contain the character itself or the ASCII value (1-255) of that character.

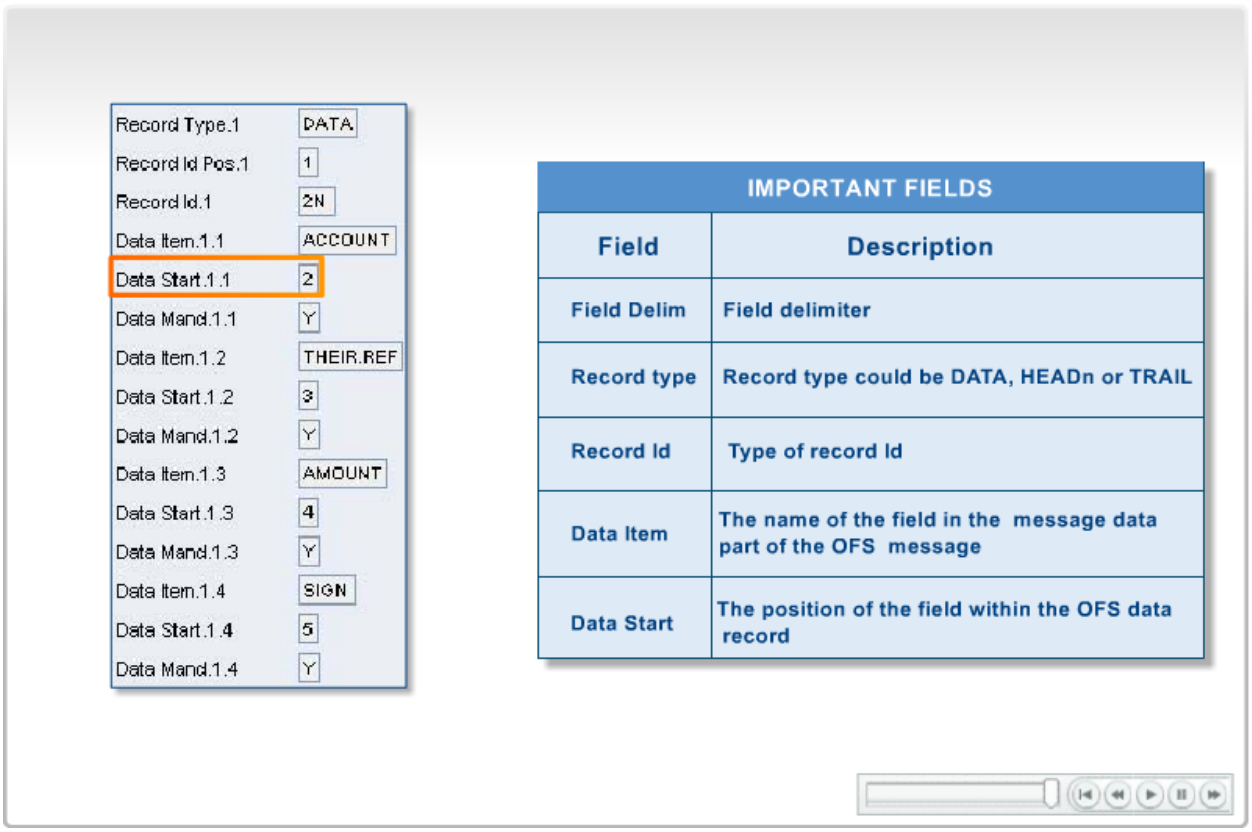
Record type : This specifies the record type could be DATA, HEADn (where n is a digit) or TRAIL. When data is supplied through a file, this serves to specify the Header, Data and Trailer part. For OFS Clearing messages, Record Type DATA alone is sufficient.

Record Id: Each message may be given a record id to identify it. This field specifies the Type of the record Id.

Data Item : You specify the actual name of a field that is going to be present in the message data part of the OFS message

Data Start : You can specify the position of the field within the OFS data record.

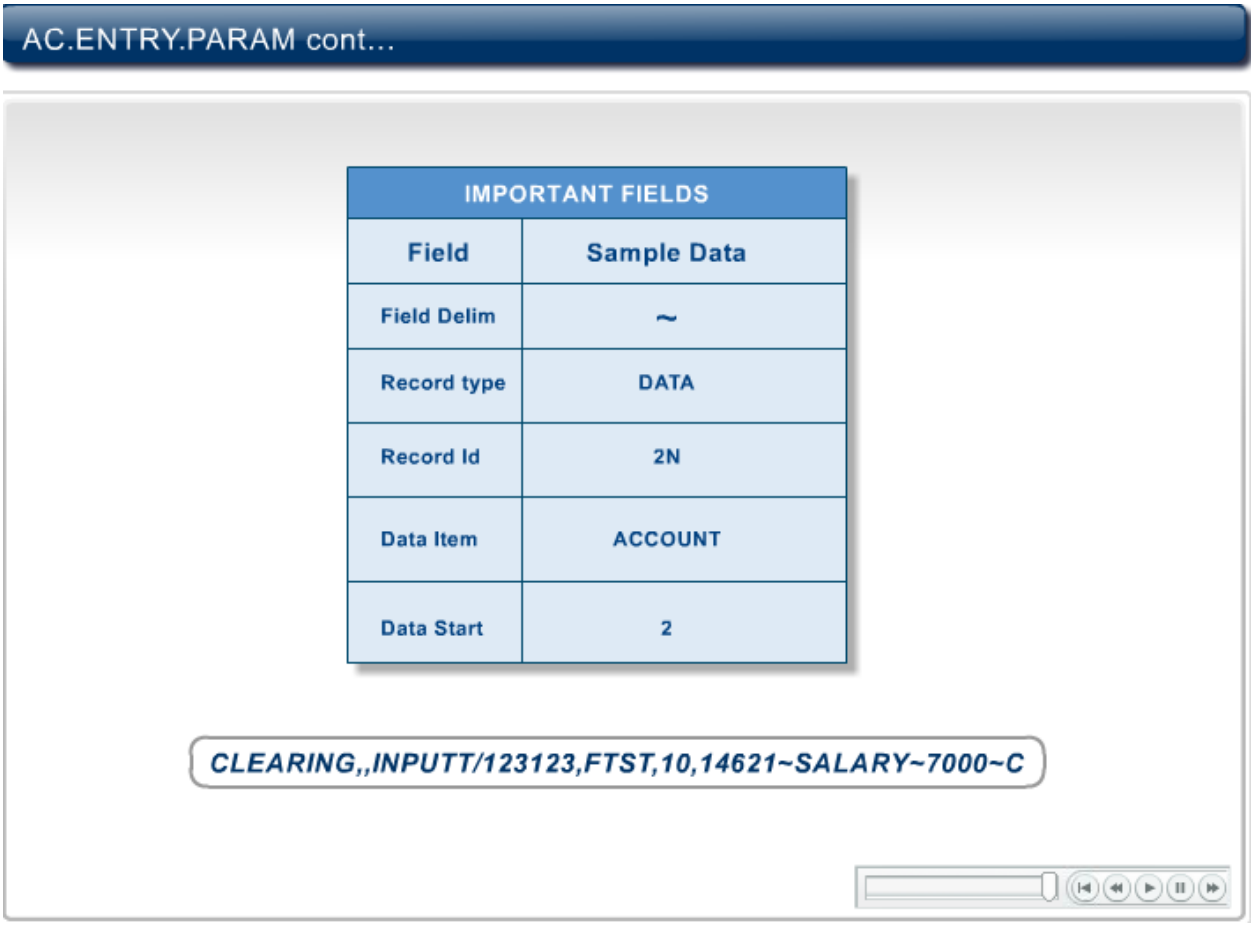
AC.ENTRY.PARAM cont...



What can we gather from the sample data shown above?

Each field (within the message part) is delimited by a tilde ~

The ACCOUNT number appears as the second field within the OFS data record



To successfully create accounting entries, the following elements must be supplied in DATA.ITEMS:

a. Account Number or P&L category

b. Amount

c. Sign, provided directly, or indirectly via a transaction code

d. The currency code if supplied is for additional verification against the account currency

The above table lists the values in the OFS message against the mapped items in AC.ENTRY.PARAM.

Message Mapping

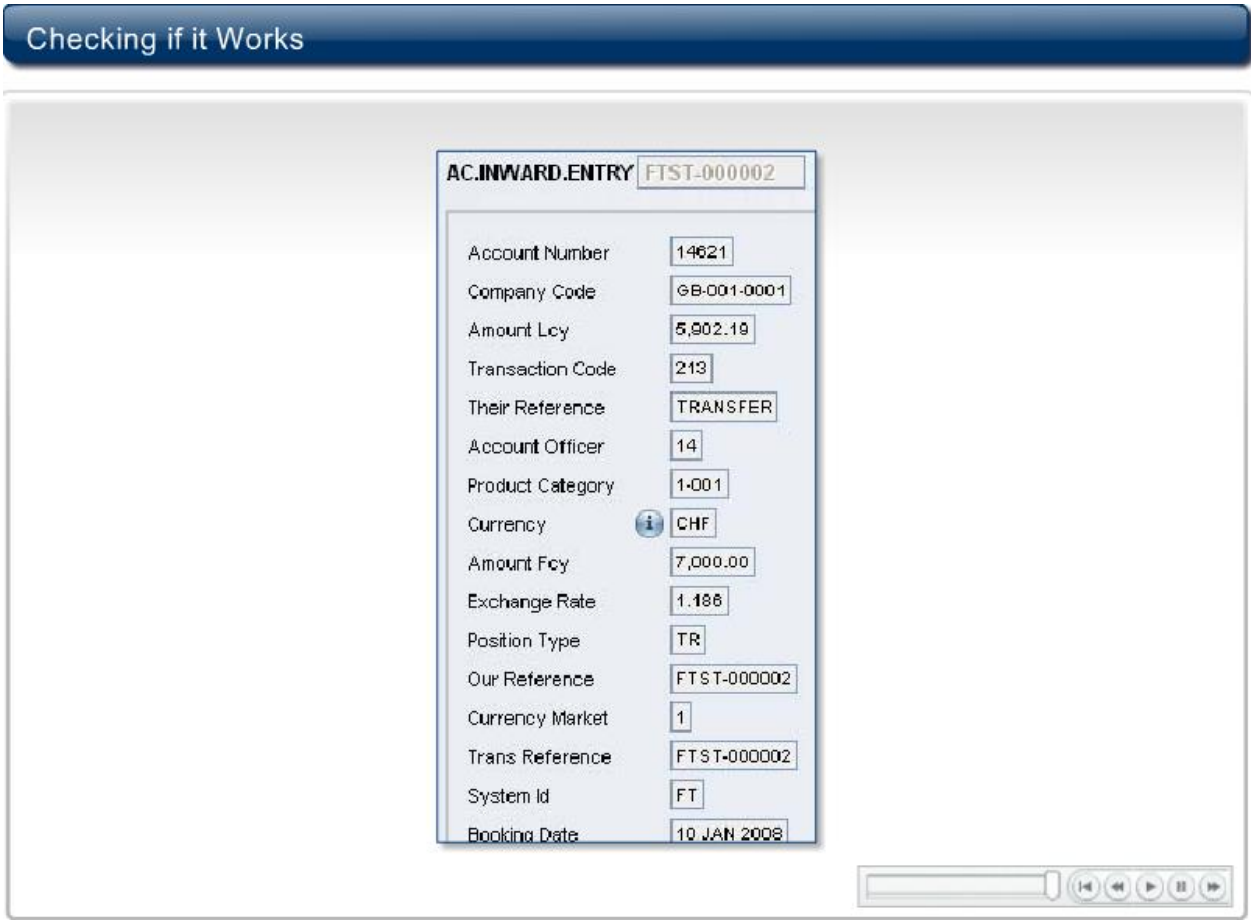
Value	Description	Data Item
CLEARING,, INPUTT/123123	OFS Header record	
FTST	Id of AC.ENTRY.PARAM record	
10	ID for the incoming Transaction. (This is for internal use in OFS.CLEARING.MANAGER)	Record Id
14621	Account Number	Account
SALARY	Reference	THEIR.REF
7000	Transaction amount	AMOUNT
C	Sign (it can be CR or DR or + or -)	SIGN

CLEARING,,INPUTT/123123,FTST,10,14621~SALARY~7000~C

Let’s check if this works!

Input the message through any of the OFS modes

Open the application AC.ENTRY.INWARD and check for the corresponding record



Trigger routines:

Where do we attach OFS trigger routines? OFS allows you to attach routines to OFS.SOURCE at specified trigger points.

Who launches these routines? These are launched by core OFS routines at different points in the life cycle of a message.

What purpose do OFS routines serve? These routines serve to convert or map the message format , update local applications and populate the in queue automatically.

The core OFS routines and the trigger routines that are launched by them are given below.

a. OFS.ONLINE.MANAGER

INITIAL.ROUTINE

IN.MSG.RTN

MSG.PRE.RTN

MSG.POST.RTN

OUT.MSG.RTN

CLOSE.ROUTINE

b. OFS.QUEUE.MANAGER

QUEUE.INIT.RTN

IN.DIR.RTN

IN.MSG.RTN

MSG.PRE.RTN

MSG.POST.RTN

OUT.MSG.RTN

QUEUE.CLOSE.RTN

c. OFS.BULK.MANAGER

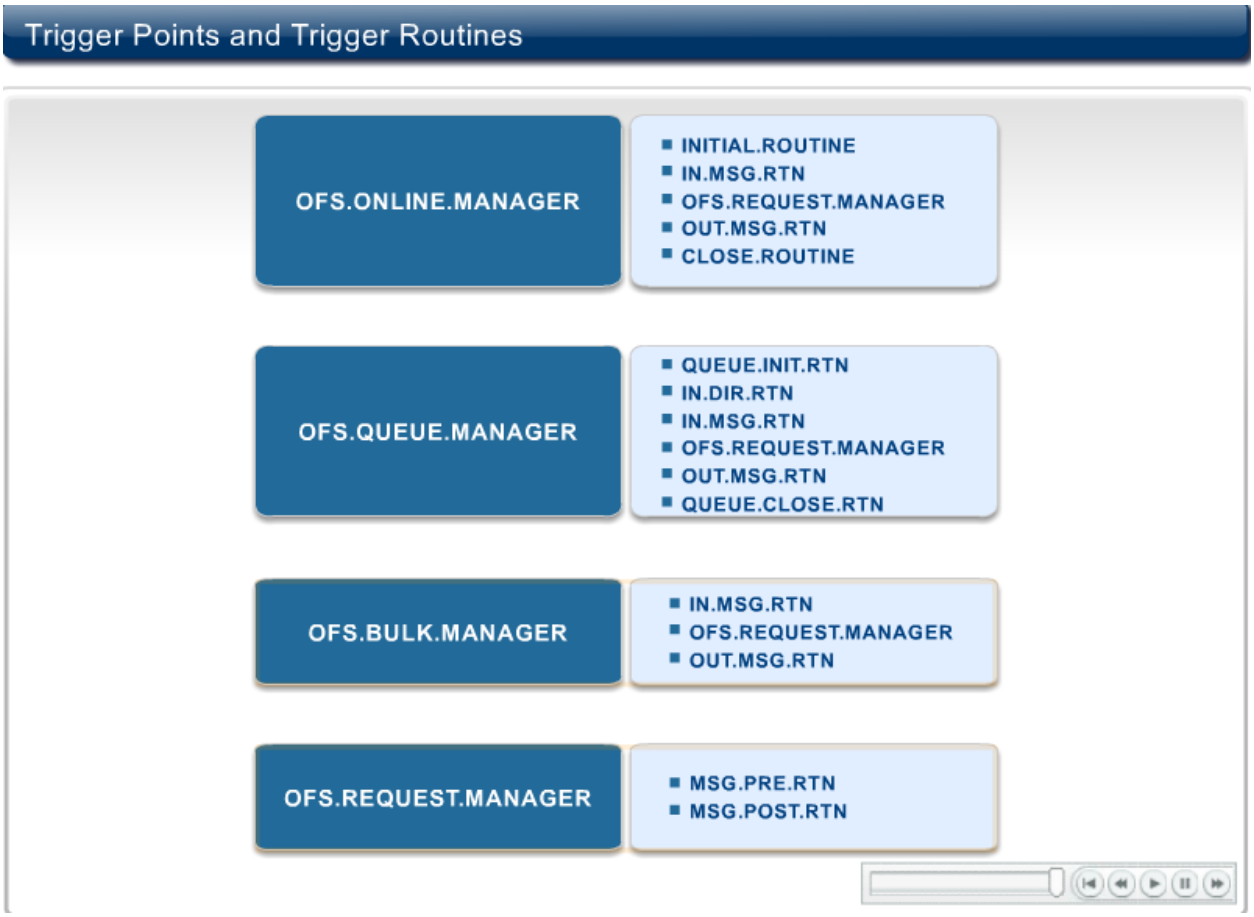
IN.MSG.RTN

MSG.PRE.RTN

MSG.POST.RTN

OUT.MSG.RTN

The MSG.PRE.RTN & MSG.POST.RTN are launched by OFS.REQUEST.MANAGER. The core OFS routines OFS.ONLINE.MANAGER, OFS.QUEUE.MANAGER & OFS.BULK.MANAGER by directly or indirectly launch OFS.REQUEST.MANAGER.



Let's look at some of these routines,

IN.MSG.RTN:

The routine specified here will be executed just BEFORE a message is processed by OFS.

Typically such a routine could convert or map the data received into the required OFS message format.

Common variables will not be accessible to these routine.

This routine is executed in both the batch and telnet modes.

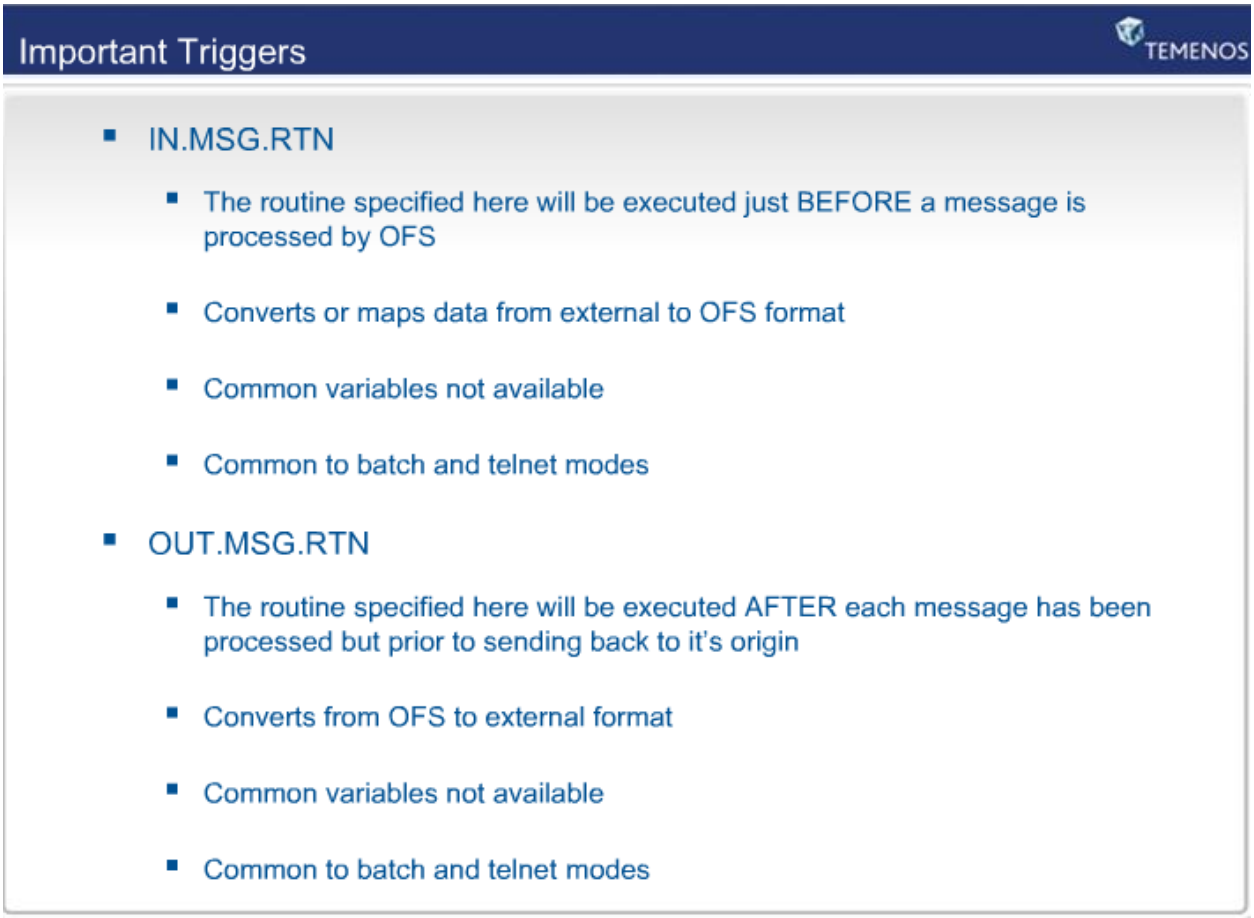
OUT.MSG.RTN:

The routine specified here will be executed AFTER each message has been processed but prior to sending back to it's origin.

This helps convert the message from OFS to external format.

The Common variables not available.

This routine is also common to batch and telnet modes.



MSG.PRE.RTN:

This routine is called by the OFS.REQUEST.MANAGER just prior to the message been processed but after the IN.MSG.RTN has been triggered.

The Common variables are available at this stage.

This routine does not allow you to alter the message.

It is typically used to update local applications.

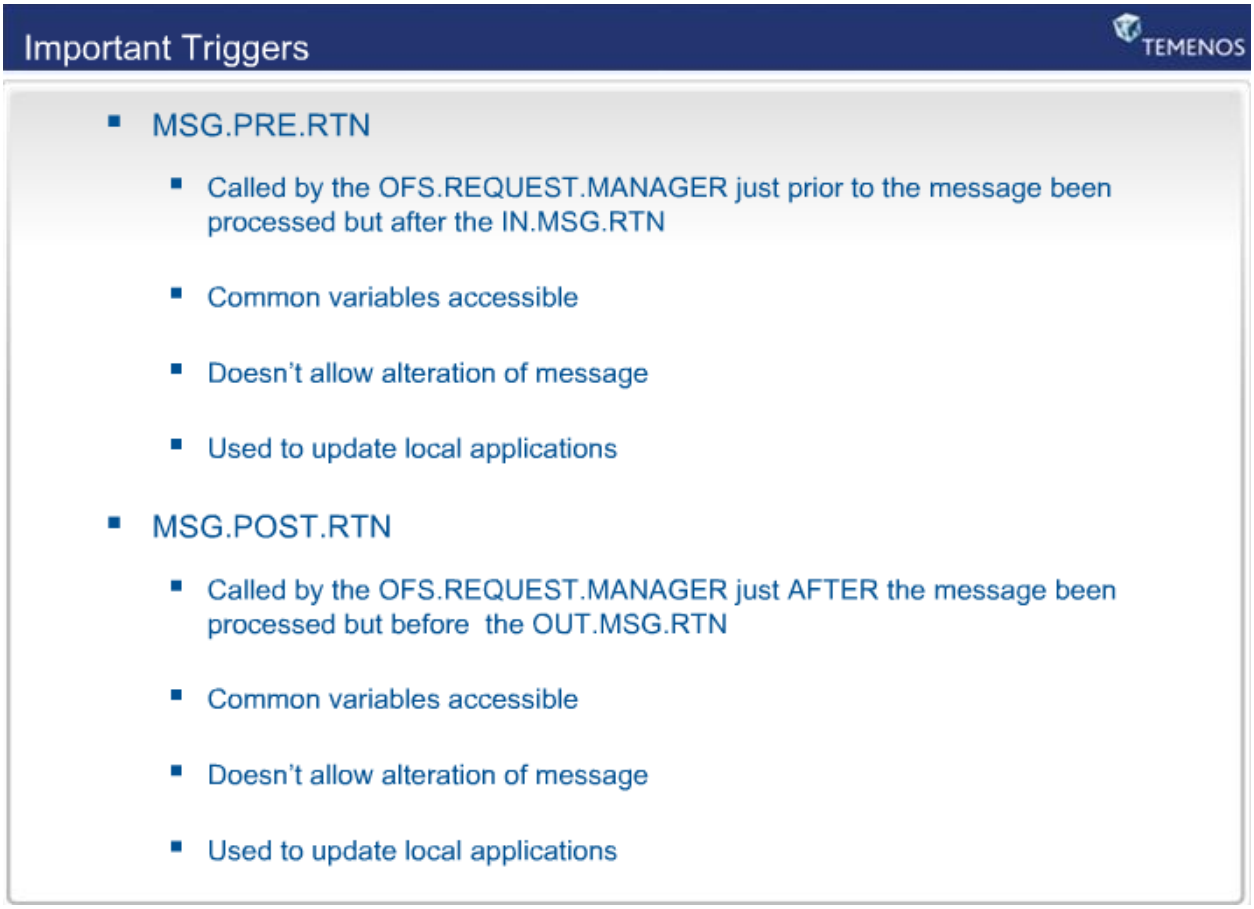
MSG.POST.RTN:

This routine is called by the OFS.REQUEST.MANAGER just AFTER the message been processed but before the OUT.MSG.RTN is called.

The Common variables are available at this stage.

This routine does not allow you to alter the message.

It is typically used to update local applications.



IN.DIR.RTN:

This routine is executed when the input directory defined in IN.QUEUE.DIR is empty, i.e. there are no messages to process.

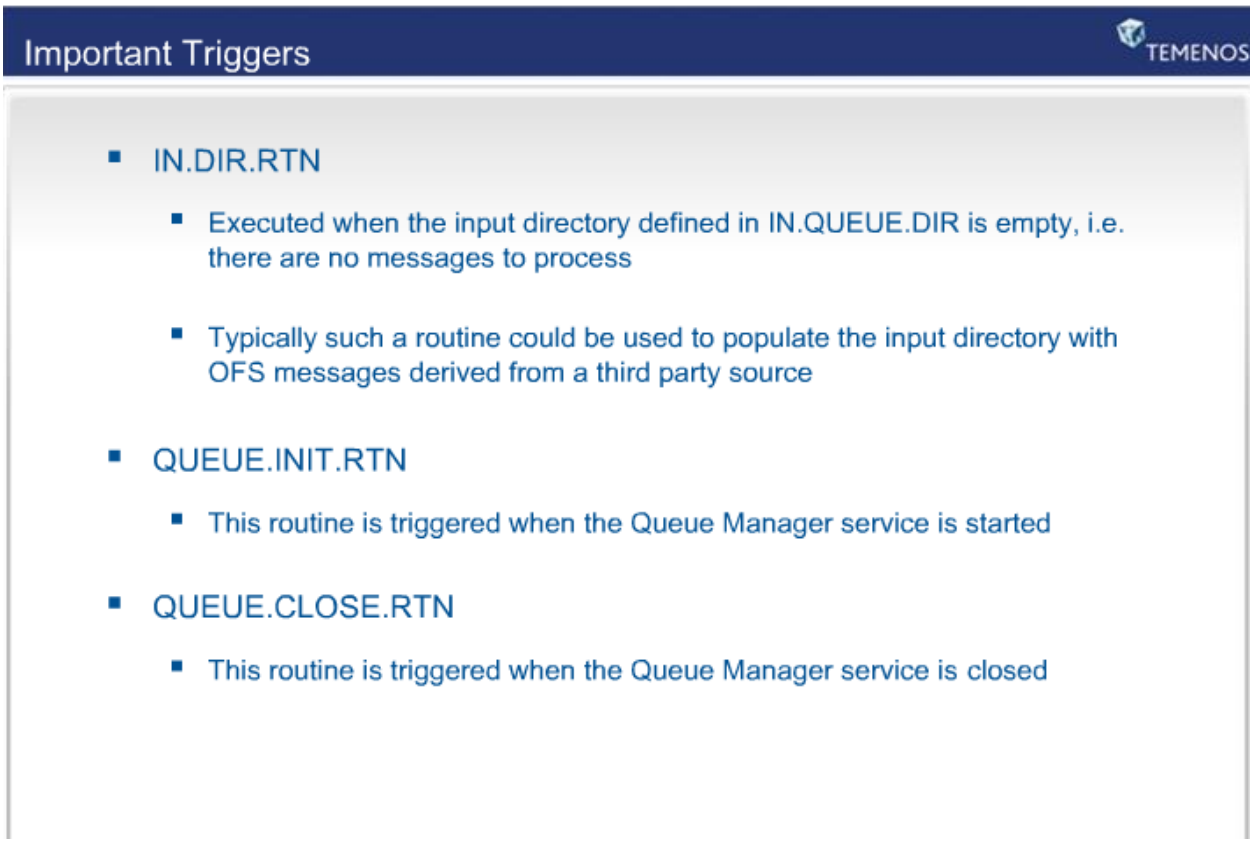
Typically such a routine could be used to populate the input directory with OFS messages derived from a third party source.

QUEUE.INIT.RTN:

This routine is triggered when the Queue Manager service is started.

QUEUE.CLOSE.RTN:

This routine is triggered when the Queue Manager service is closed.



The table given here shows some of the trigger routines used within model bank.

Examples

OFS.SOURCE	Trigger type	Routine name
SP.STP.OFS	IN.DIR.RTN	SC.OFS.INWARD.MAPPING
AC.EXPECTED.RECS	OUT.MSG RTN	STO.UPDATE.AC.EXP.RECS.ID
OFS.FX.LIM.ORDER	OUT.MSG RTN	OFS.UPD.FX.ID

Trigger Routines - Scenarios

TEMENOS

Let us take a closer look at some trigger routines and their need with the help of a few scenarios

- IN.MSG.RTN
- MSG.POST.RTN
- IN.DIR.RTN

Scenario - 1

TEMENOS

- A third party system is posting messages to T24. It sends messages in own format. Each message contains a message id, transaction type, debit account, debit currency, amount, credit account, credit currency. These details are written in simple comma delimited format into a sequential file
- You need to convert the message into an OFS format

Solution - 1

TEMENOS

- We need a place, a folder, where the third party system can put it's messages
- **Solution**: Define a OFS.SOURCE record of type BATCH. Define the IN.QUEUE.DIR where messages can be stored
- We need to convert the messages to OFS format
- **Solution**: Write a routine to convert
  - Use IN.MSG.RTN

Enter, compile the code shown above. Make sure you register it in PGM.FILE.

```
SUBROUTINE FT.OFS.CONVERT(Y.REQUEST)
$INSERT I_FT.OTHERS
$INSERT I_F.FUNDS.TRANSFER
APP.NAME = 'FUNDS.TRANSFER'
OFSFUNCT='I'
PROCESS='PROCESS'
OFSVERSION='FUNDS.TRANSFER,'
GTSMDOE=''
NO.OF.AUTH=''
TRANSACTION.ID='/':FIELD(Y.REQUEST,",",1)
R.RECORD=''
R.OFS.RECORD=''
```

Code to Convert

```
R.RECORD<FT.TRANSACTION.TYPE>=FIELD(Y.REQUEST,",",2
)
R.RECORD<FT.DEBIT.ACCT.NO>=FIELD(Y.REQUEST,",",3)
R.RECORD<FT.DEBIT.CURRENCY>=FIELD(Y.REQUEST,",",4)
R.RECORD<FT.DEBIT.AMOUNT>=FIELD(Y.REQUEST,",",5)
R.RECORD<FT.CREDIT.ACCT.NO>=FIELD(Y.REQUEST,",",6)
R.RECORD<FT.CREDIT.CURRENCY>=FIELD(Y.REQUEST,",",7)
CALL
OFS.BUILD.RECORD(APP.NAME,OFSFUNCT,PROCESS,OFSVERSI
ON,GTSMODE,NO.OF.AUTH,TRANSACTION.ID,R.RECORD,R.OFS
.RECORD)
CRT R.OFS.RECORD
FT.REQUEST=Y.REQUEST
FT.MSG.ID=FIELD(Y.REQUEST,",",1)
Y.REQUEST=R.OFS.RECORD
END
```

Code to Convert

Scenario - 2

TEMENOS

- The bank also requires that the FT id generated by T24 and the msg id are stored in a separate CSV file for auditing and tracking

Solution - 2

TEMENOS

- We need to store the transaction id and message id
- **Solution:** Write a routine to store the details on a file
  - Use MSG.POST.RTN

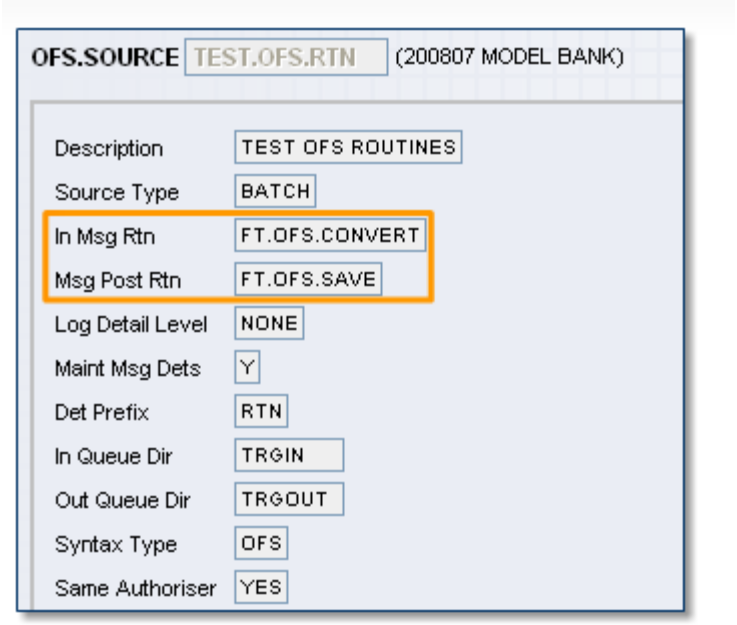
Think about: Why can't we use MSG.PRE.RTN to convert? Why can't we use IN.MSG.RTN for storing transaction and message details?

```
SUBROUTINE FT.OFS.SAVE(Y.REQUEST)
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.FUNDS.TRANSFER
$INSERT I_FT.OTHERS

GOSUB INIT
GOSUB WRITE.DETAILS
CLOSESEQ F.BP
RETURN

INIT:
Y.RECORD=''
OPENSEQ "TRGOUT","MSGDETAILS.txt" TO F.BP ELSE
    CREATE F.BP ELSE ETEXT = "Unable to write to file"
  END
RETURN
```

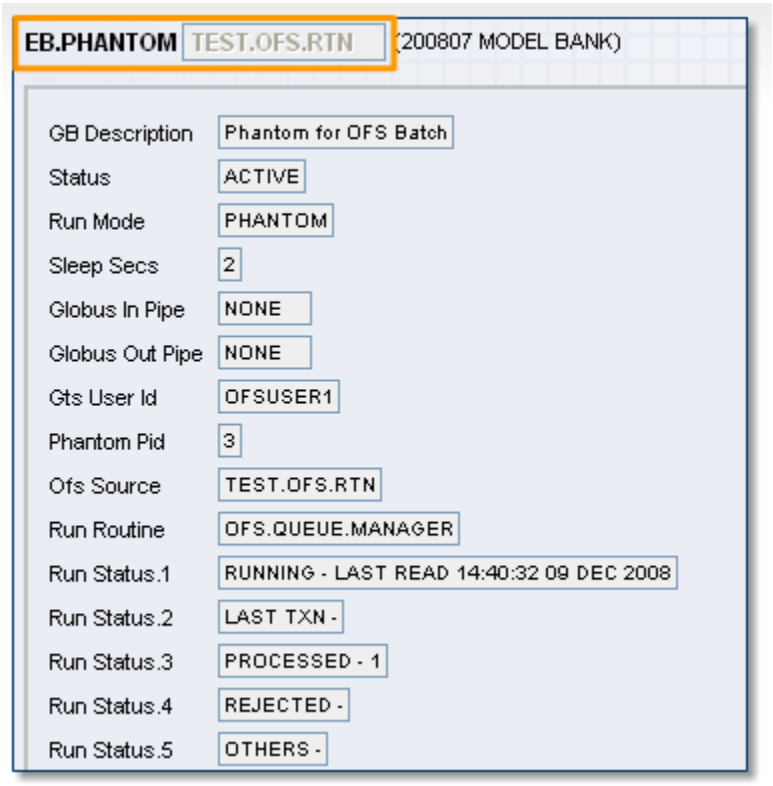
```
WRITE.DETAILS:
Y.RECORD= ID.NEW:",":FT.MSG.ID
WRITESEQ Y.RECORD TO F.BP ELSE
ETEXT=  "Unable to write transaction details"
END

RETURN
```

Code to Save

A sample OFS.BATCH record is shown above. Notice the two trigger routines FT.OFS.CONVERT and FT.OFS.SAVE.



An EB.PHANTOM entry for running OFS in batch mode is shown on this page.



Solution - Final Step

- Verify the EB.PHANTOM record to start the OFS Batch phantom
- Enter a OFS transaction in the IN.QUEUE.DIR
- Check if your have received an output in the OUT.QUEUE.DIR
- Check if you have a file called MSGDETAILS.txt in the same OUT.QUEUE.DIR

Scenario - 3

- The bank has received Funds Transfer requests from various sources. These FTs are still in an un-authorised state. The bank wants an OFS message to be automatically created to authorise any each FundsTransfer record in an INAU status

Solution - 3

We need to form the OFS messages (that will authorise the FTs)

- Solution: We need to write a routine
- What will we do with these messages?
- Solution: Write them into a queue. Batch mode
- But when does this routine run?
- Solution: Before the message is formed

Think about: What type of routine can you use?

Solution - 3

- You cannot use a routine that is triggered for each message

  E.g. IN.MSG.RTN, MSG.PRE.RTN, MSG.POST.RTN

- Which routines are not triggered for each message?
  - IN.DIR.RTN, QUEUE.INIT.RTN, QUEUE.CLOSE.RTN

Solution - 3

Solution:

Write a subroutine to populate the IN QUEUE based on the conditions (INAU records in this case), and attach it to IN.DIR.RTN in OFS.SOURCE

Think about: Why about why QUEUE.INIT.RTN or QUEUE.CLOSE.RTN are not suitable

Solution 3 – Code to populate IN QUEUE

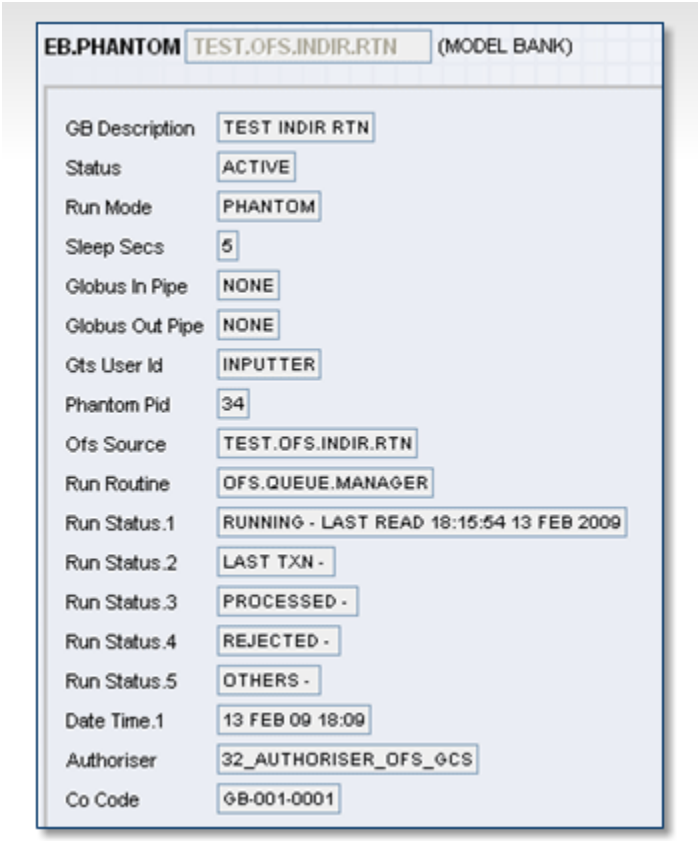
```
SUBROUTINE FT.OFS.AUT
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_GTS.COMMON
$INSERT I_F.FUNDS.TRANSFER
$INSERT I_F.OFS.SOURCE

GOSUB INIT
GOSUB SELECT.PROCESS
RETURN

INIT:
APP.NAME = 'FUNDS.TRANSFER'
OFSFUNCT='A'
PROCESS='PROCESS'
OFSVERSION='FUNDS.TRANSFER,'
GTSMODE='1'
```

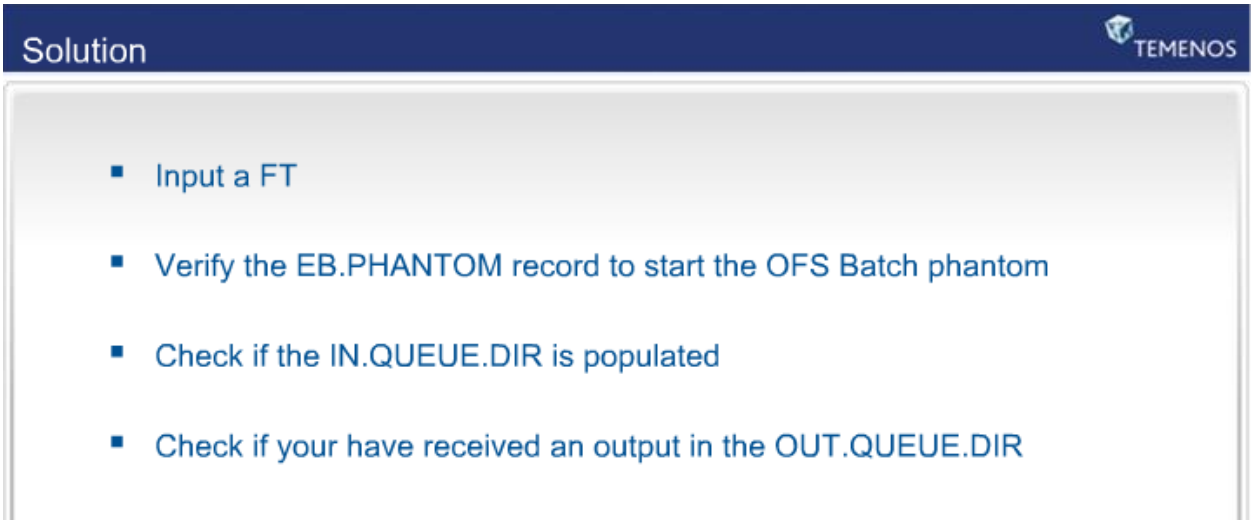
```
NO.OF.AUTH=''
R.RECORD=''
R.OFS.RECORD=''
R.OFS.SOURCE=OFS$SOURCE.REC
IN.QUEUE.PATH = R.OFS.SOURCE<OFS.SRC.IN.QUEUE.DIR>
V.ERROR=''
FN.FT = 'F.FUNDS.TRANSFER$NAU'
SEL.LIST = ''
NO.OF.REC = 0
RET.CODE = ''
Y.FT.ID = ''
R.FT = ''
F.FT = ''
Y.FT.ERR = ''
REC='R1'
INDIR='INDIR1'
F.BP=''
MSG.ID= ''

RETURN
```

```
 SELECT.PROCESS:
     CALL OPF(FN.FT,F.FT)
*TO OPEN INAU FT RECORDS FROM $NAUFILES
     SEL.CMD= "SELECT ":FN.FT:" WITH RECORD.STATUS='INAU'"
     CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
     LOOP
        REMOVE Y.FT.ID FROM SEL.LIST SETTING POS
     WHILE Y.FT.ID
        CALL F.READU(FN.FT,Y.FT.ID,R.FT,F.FT,Y.FT.ERR,'')
        CALL OFS.BUILD.RECORD(APP.NAME,OFSFUNCT,PROCESS,
        OFSVERSION,GTSMODE,NO.OF.AUTH,Y.FT.ID,R.RECORD,
        R.OFS.RECORD)
*TO OPEN THE IN QUEUE
      CALL ALLOCATE.UNIQUE.TIME(MSG.ID)
      OPENSEQ IN.QUEUE.PATH,MSG.ID TO  F.BP ELSE
        CREATE F.BP ELSE
         ETEXT = "UNABLE TO CREATE MsgDetails.txt"
        END
      END
```

```
*TO WRITE TO THE IN QUEUE

   WRITESEQ R.OFS.RECORD TO F.BP ELSE
   ETEXT = "UNABLE TO WRITE"
   END
*TO CLOSE THE INQUEUE AFTER WRITING

   CLOSESEQ F.BP


REPEAT
RETURN
END
```

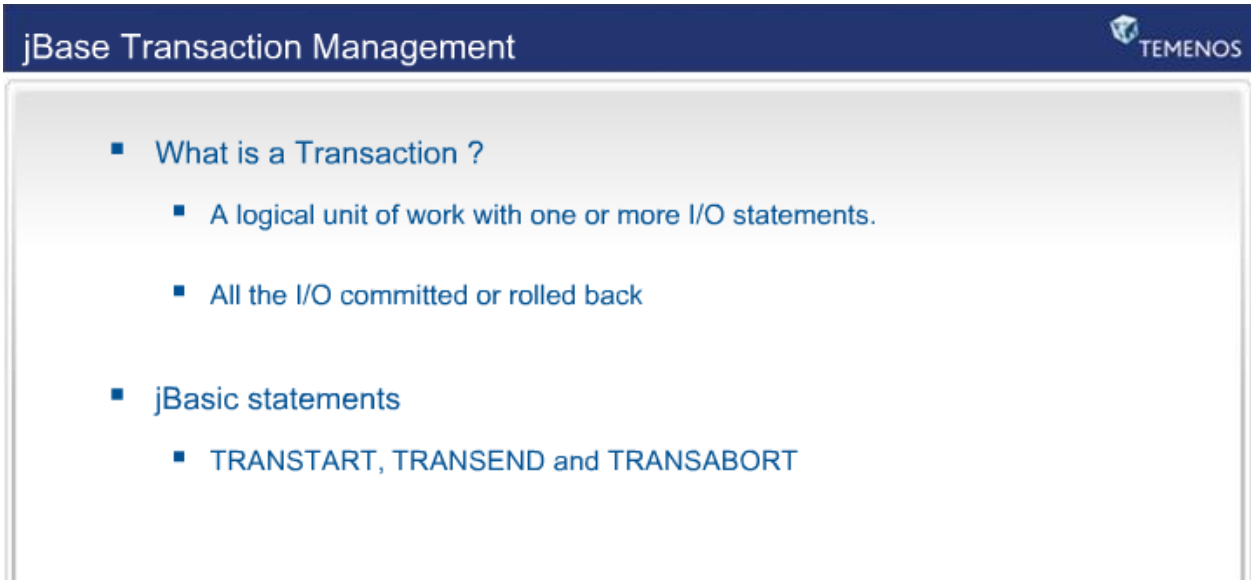

OFS.SOURCE
TEST.OFS.INDIR.RTN
(MODEL BANK)
Description
TEST INDIR RTN
Source Type
BATCH
Log Detail Level
NONE
In Queue Dir
IN1
Out Queue Dir
OUT1
Syntax Type
OFS
In Dir Rtn
FT.OFS.AUT
Same Authoriser
YES
Curr No
1
Inputter.1
34_AUTHORISER___OFS_GCS
Date Time.1
13 FEB 09 11:58
Authoriser
34_AUTHORISER_OFS_GCS
Co Code
GB-001-0001
Dept Code
1


EB.PHANTOM
TEST.OFS.INDIR.RTN
(MODEL BANK)
GB Description
TEST INDIR RTN
Status
ACTIVE
Run Mode
PHANTOM
Sleep Secs
5
Globus In Pipe
NONE
Globus Out Pipe
NONE
Gts User Id
INPUTTER
Phantom Pid
34
Ofs Source
TEST.OFS.INDIR.RTN
Run Routine
OFS.QUEUE.MANAGER
Run Status.1
RUNNING - LAST READ 18:15:54 13 FEB 2009
Run Status.2
LAST TXN -
Run Status.3
PROCESSED -
Run Status.4
REJECTED -
Run Status.5
OTHERS -
Date Time.1
13 FEB 09 18:09
Authoriser
32_AUTHORISER_OFS_GCS
Co Code
GB-001-0001



Transaction management and OFS

What is a transaction from a jBase perspective ?A transaction is a logical unit of work that contains one or more I/O statements. A transaction is an atomic unit. The effects of all the I/O statements in a transaction can be either all committed (written to the database) or all rolled back (undone from the database).



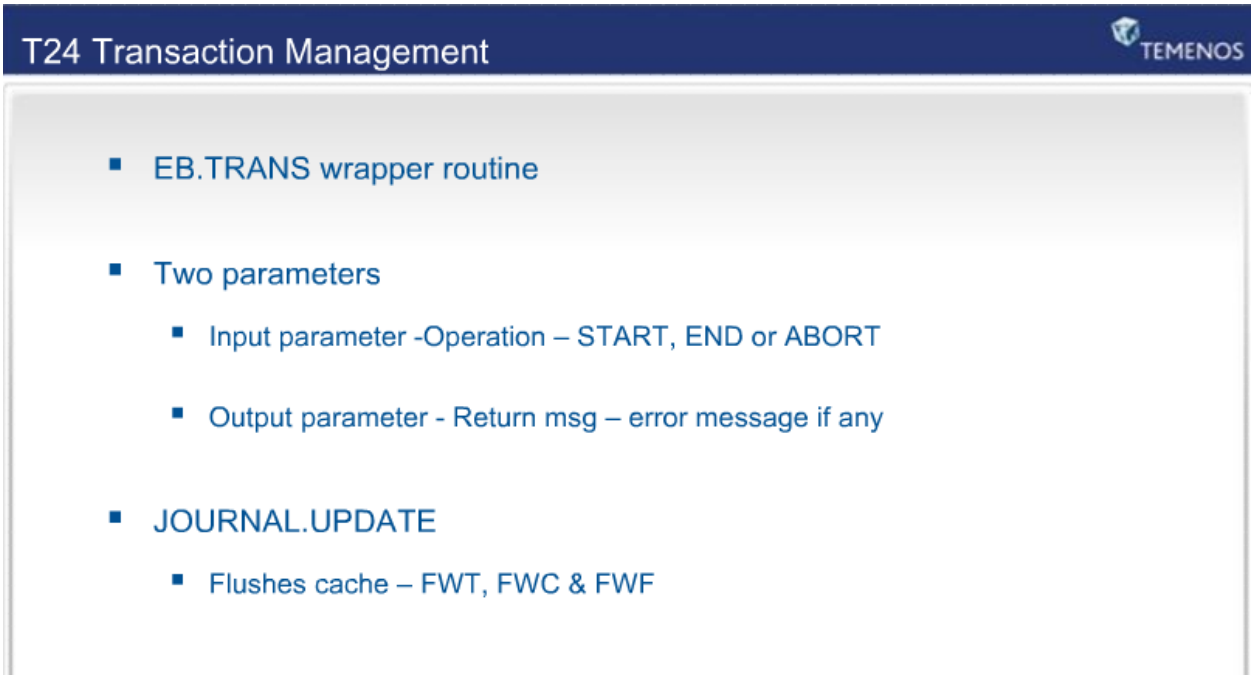
The jBase commands for managing transactions are normally not used within T24 code directly. We use a wrapper routine called EB.TRANS instead. This internally invokes the relevant jBase commands based on the parameter that is passed.

EB.TRANS takes two parameters. The first is an input parameter , which can be given a value of START, END or ABORT. This results in either a TRANSTART, TRANSEND or TRANSABORT been called. The second parameter is an output parameter, which returns a message in the case of errors.

T24 routines and applications normally call an API routine named JOURNAL.UPDATE to implement transaction management.

JOURNAL.UPDATE begins a transaction block by calling EB.TRANS and flushes the content of the cache onto persistent storage. The cache you may recall is implemented using the arrays FWT, FWC & FWF.

If an error occurs while writing data, the transactions that have taken place after switching on transaction management, i.e. all the writes that have taken place using the data in the cache, are rolled back.



Let's look at OFS messages from a transaction management viewpoint.

The typical good old kind of OFS messages that we are used to, were messages that mostly manipulated only a single application, and usually carried out a update to an application along with related updates within a transaction block.

The new breed of OFS messages that we now see are bulk messages. In this case, many messages are grouped within a single transaction block. The failure of one message would rollback all the transactions within that transaction block.

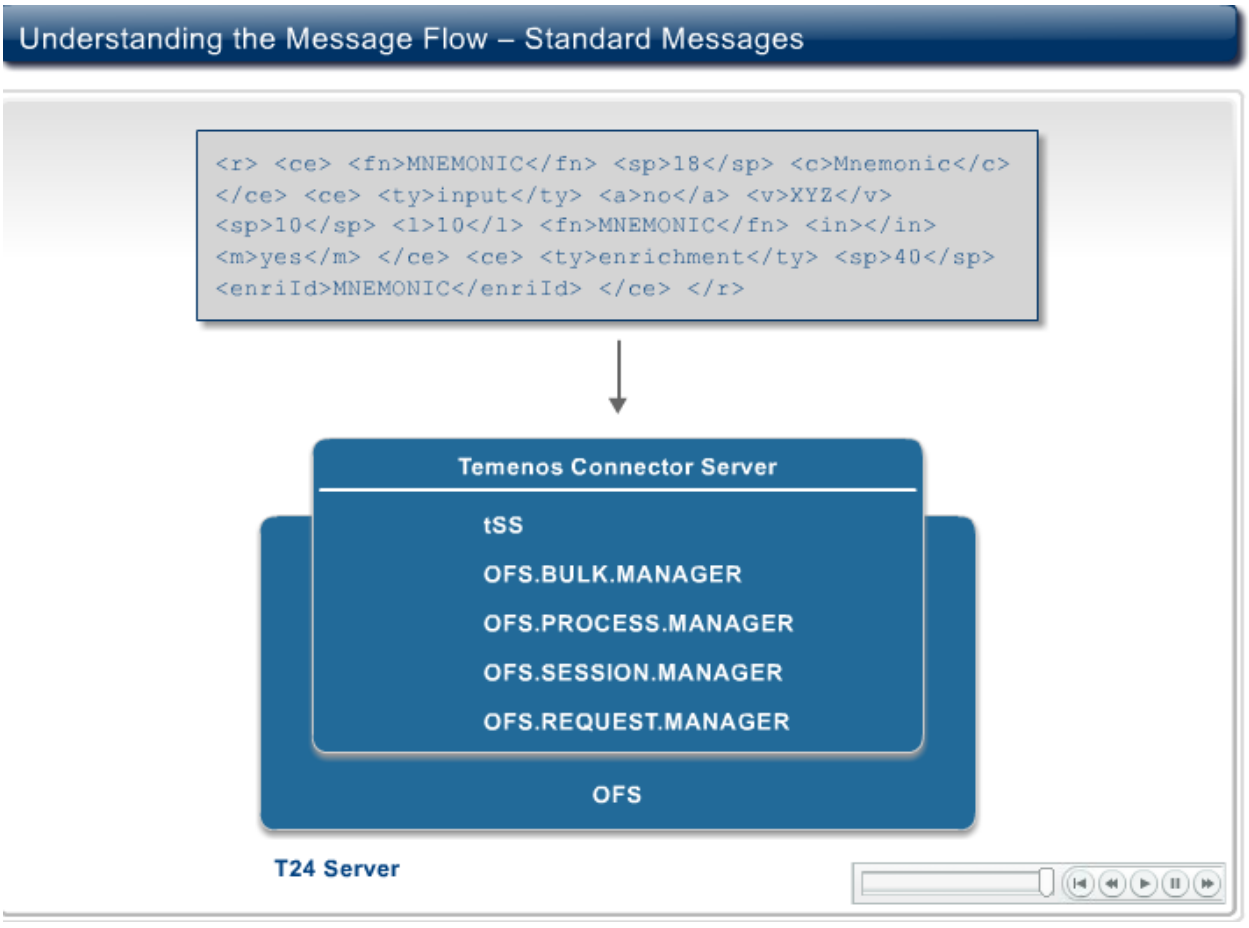
This kind of transaction management became necessary with the advent of the Arrangement Architecture. AA has at it's core the notion of an ARRANGEMENT which comprises of elements called properties and arrangement conditions. Each 'element' is an application in the old sense. Creating an arrangement would involve creating multiple elements and therefore updating multiple applications, i.e. involve multiple transactions. All the transactions have to be successful or none.

Lets recollect what happens when TCS receives a message.

Imagine a situation wherein a customer record is committed using T24 Browser. The request to commit the record would have passed the Web Server and reached TCS.

TCS then 'opens' a connection to T24. How does it do this? It spawns a program called tSS . tSS receives the request and invokes the OFS.BULK.MANAGER. The OFS.BULK.MANAGER's main job is to determine whether the request is a standard request or a bulk request. For a standard request like ours, it would pass the message to the OFS.PROCESS.MANAGER.

Bulk requests are handled in a similar fashion .The bulk manager would pass one request at a time to the OFS.PROCESS.MANAGER. The OFS.PROCESS.MANAGER would accept the request and decide how to handle it appropriately. Remember it calls OFS.SESSION.MANAGER for a normal OFS (or browser) message ,and OFS.DE.REQUEST for a Delivery message . Our message would therefore reach OFS.SESSION.MANAGER. This would eventually pass it onto OFS.REQUEST.MANAGER for processing.

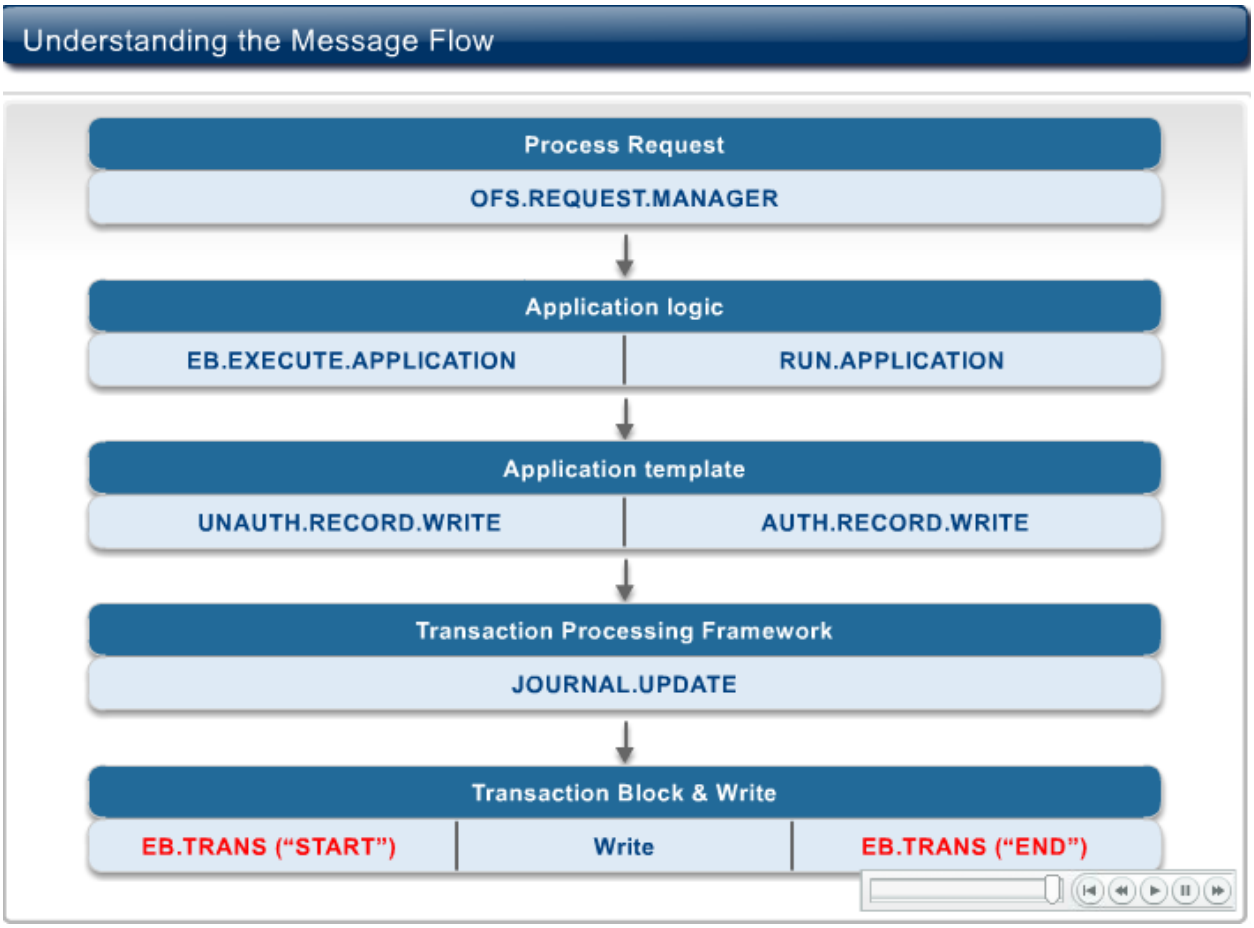


Once the control comes to OFS.REQUEST.MANAGER, please note that the actual application, CUSTOMER in this case is called. Within the CUSTOMER application, there is code to,

a. Validate data

b. Make ready the record to be written into the CUSTOMER data file (Live or Unauthorised as the case may be)

c. Make ready the record(s) to be written to other concat files, live files, other application's data files (E.g. When a customer record is input, files such as F.MNEMONIC.CUSTOMER which is a concat file, F.DE.ADDRESS which is another application in T24)
Once the data is all set to be written, the CUSTOMER application (or any application in T24 for that matter), calls the transaction processing framework. The Transaction processing framework here refers to calls made to UNAUTH.RECORD.WRITE (Which is called when a record is committed) and AUTH.RECORD.WRITE (W hich is called when a record is authorized). These T24 APIs internally call a T24 API named JOURNAL.UPDATE.



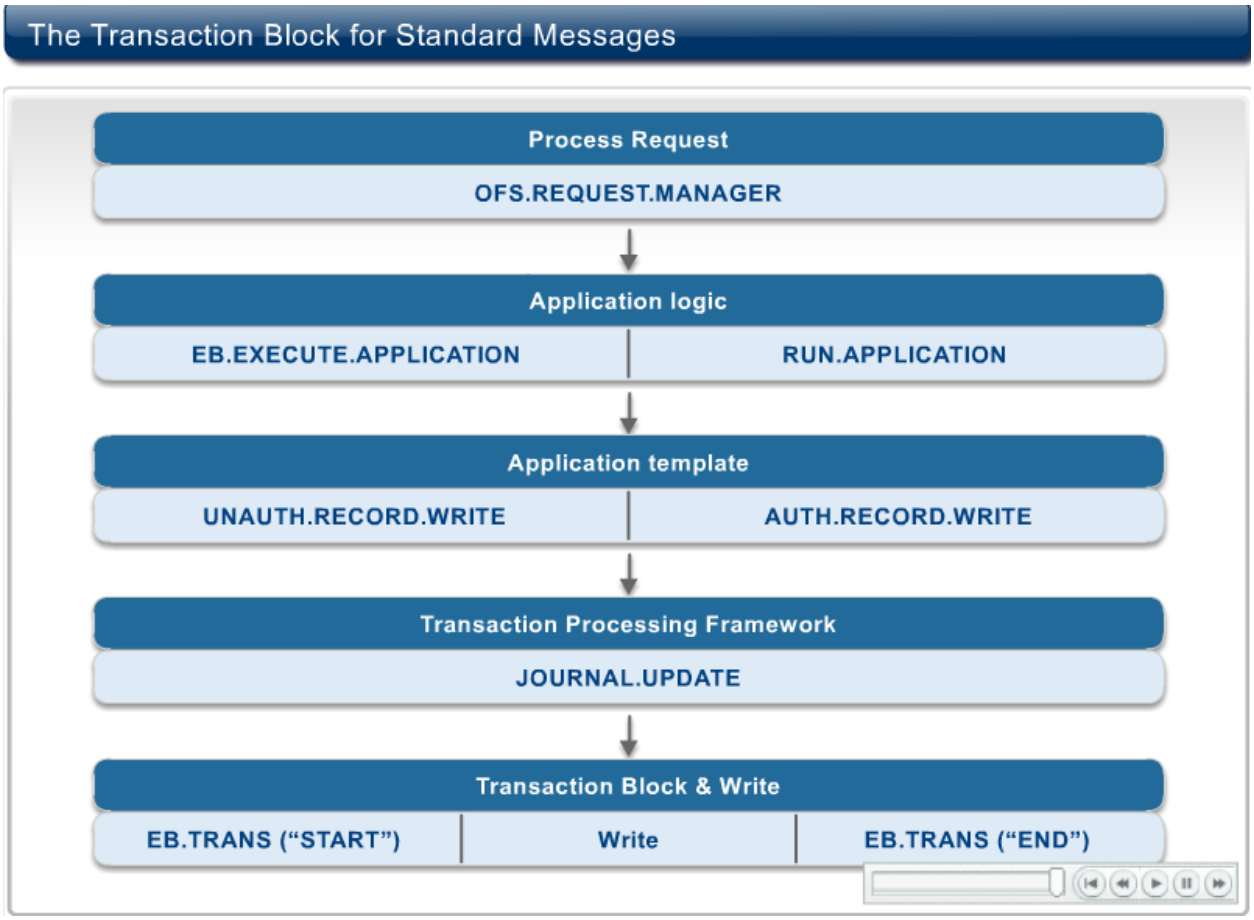
JOURNAL.UPDATE is the T24 API that does transaction management in T24. It internally invokes EB.TRANS with START to start a transaction block. While a transaction block in open, all records that need to be written to disk are written on to memory. Once all records have been written on to memory, a check is performed to see if the writes have been successful.

If yes, EB.TRANS with an END is called to denote the end of a transaction block. At this point, all data is flushed from memory to disk. If not successful, EB.TRANS with ABORT is called to abort the transaction block. At this juncture, all records that were written to memory are flushed out from memory.

There are two very important aspects point to note here.

a. Though multiple files and applications get updated, the OFS request that came in is just ONE

b. The transaction block is initiated by the application by calling the transaction processing framework

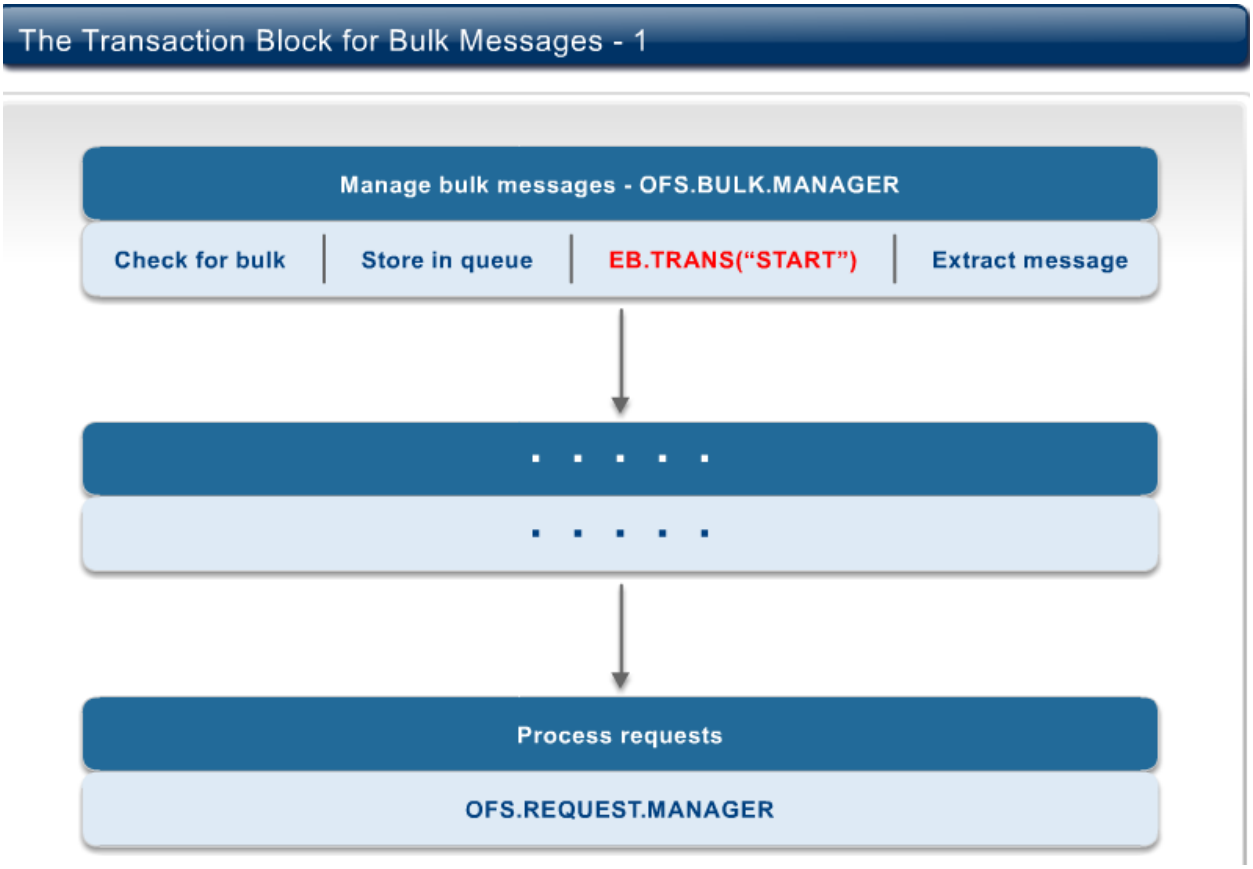


In order to support this concept of receiving multiple OFS messages together, OFS.BULK.MANAGER was introduced. As the name suggests, it helps process bulk OFS messages.

Meaning, when it receives an OFS message, it checks to see if multiple OFS messages have been supplied delimited with FM.

If yes, it splits them, and stores them in a queue in memory called cTxn_RequestQueue.

Once all the messages are ready, it starts a transaction block and for each message uses the same route of OFS.PROCESS.MANAGER, OFS.SESSION.MANAGER, OFS.REQUEST.MANAGER and finally the application.



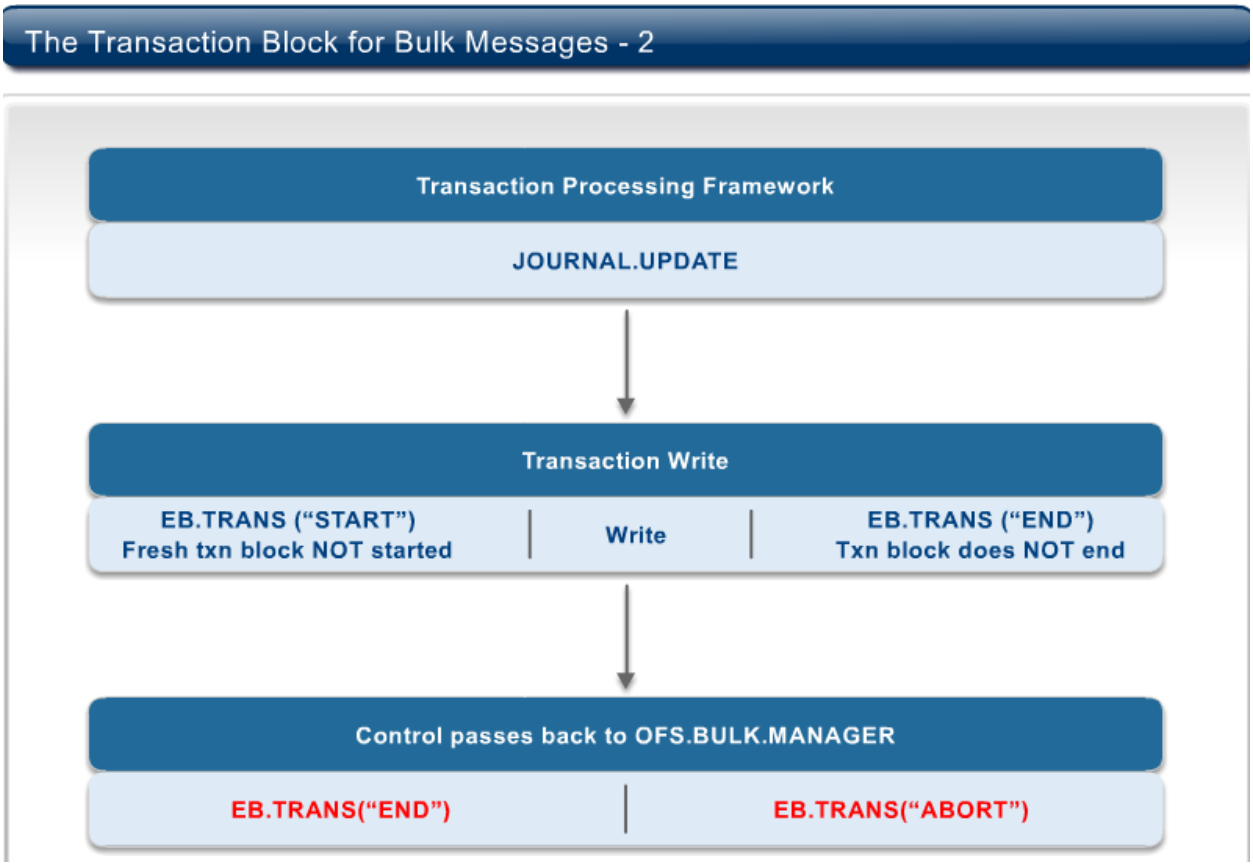
The application would then invoke the transaction processing framework. Now this framework has been modified in such a way that it respects the transaction boundary started by OFS.BULK.MANAGER and does not start another one.

Once all messages have been processed and if successful, OFS.BULK.MANAGER marks the end of the transaction block thus committing all the transactions. If there was an error in any one of the messages, all are rolled back as OFS.BULK.MANAGER internally calls EB.TRANS with "ABORT".

Why is the transaction block managed by OFS.BULK.MANAGER and not the transaction processing framework as discussed earlier?.

Just imagine if OFS.BULK. MANAGER did not start a transaction block, when each message gets processed, the respective application would start and end its own transaction block . What this implies is that, the first message might be successfully processed and committed to the database, but if there was an error in the second message, the second message alone would fail.

By the same principle, subsequent messages may or may not be committed. At the end, there will be a situation where some of the messages would have got processed while some would not have. That would not be correct would it? When you input an arrangement for example, you want all related applications to be updated or none to be updated. This is the functionality that OFS.BULK.MANAGER provides.

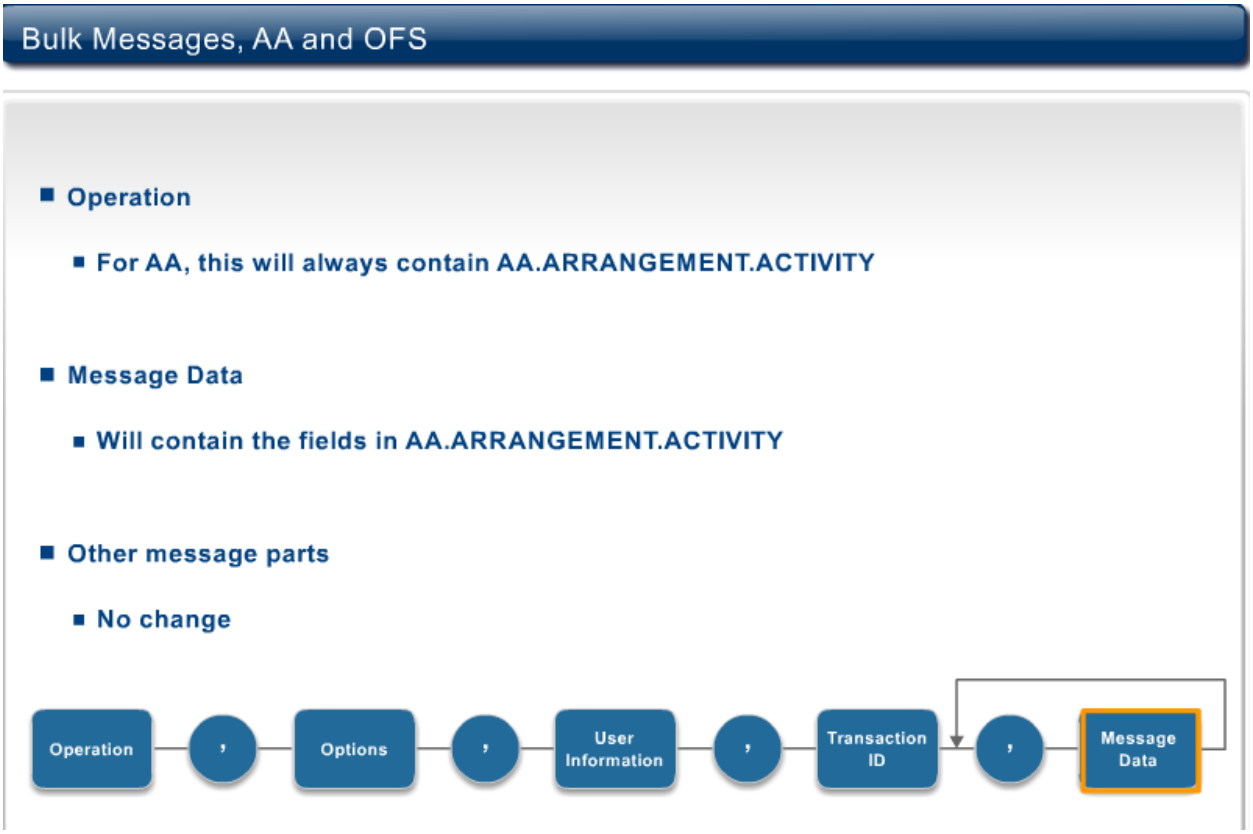


We have already discussed that one scenario where bulking is necessary is the Arrangement Architecture. You will now look at the OFS messages generated in bulking and the use of bulking. First, let's look at the OFS syntax for AA.

As you are aware, the 'Operation' section contains the name of the application that needs to be updated. For AA, this will always hold the value AA.ARRANGEMENT.ACTIVITY. In AA, everything is an activity. When you create a new lending arrangement, you are performing an activity called LENDING-NEW-ARRAMGEMENT. When you wish to decrease the term of the arrangement or the amount of the arrangement, then the activity is LENDING-DECREASE-TERM.AMOUNT. Since everything is an activity, the application name to be used here is AA.ARRANGEMENT.ACTIVITY.

The Message Data part of the message then will contain field names and values for fields in AA.ARRANGEMENT.ACTIVITY.

There no change to the other parts of the message.

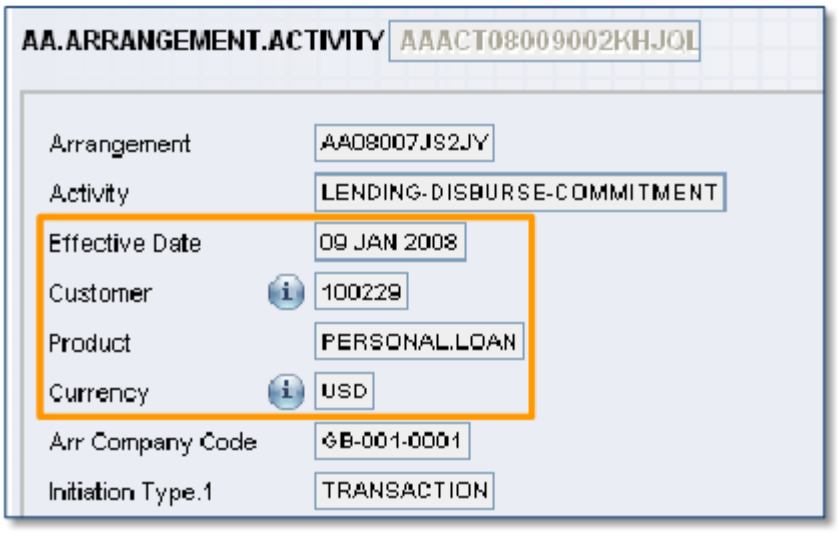


The application name used is AA.ARRANGEMENT.ACTIVITY.

The user credentials are supplied.

Since this is a new arrangement (You are not trying to modify an existing arrangement), a record needs to be created in AA.ARRANGEMENT. In order to create a new record in AA.ARRANGEMENT, supply the field name AA.ARRANGEMENT and supply a value NEW to it. T24 will automatically create a new record in AA.ARRANGEMENT and also populate the ID of the new arrangement record in this field in AA.ARRANGMENT.ACTIVITY.

Since the activity to be performed is LENDING-DISBURSE-COMMITMENT, supply that value to the field ACTIVITY. Supply values for EFFECTIVE.DATE, CUSTOMER and CURRENCY.Supply values for PRODUCT.



Apart from AA.ARRANGEMENT and AA.ARRANGEMENT.ACTIVITY, a number of other applications get updated. These applications have names beginning with AA.ARR

Following is the syntax to supply values for all properties

PROPERTY: Multi value position: Sub value position = {Property name}

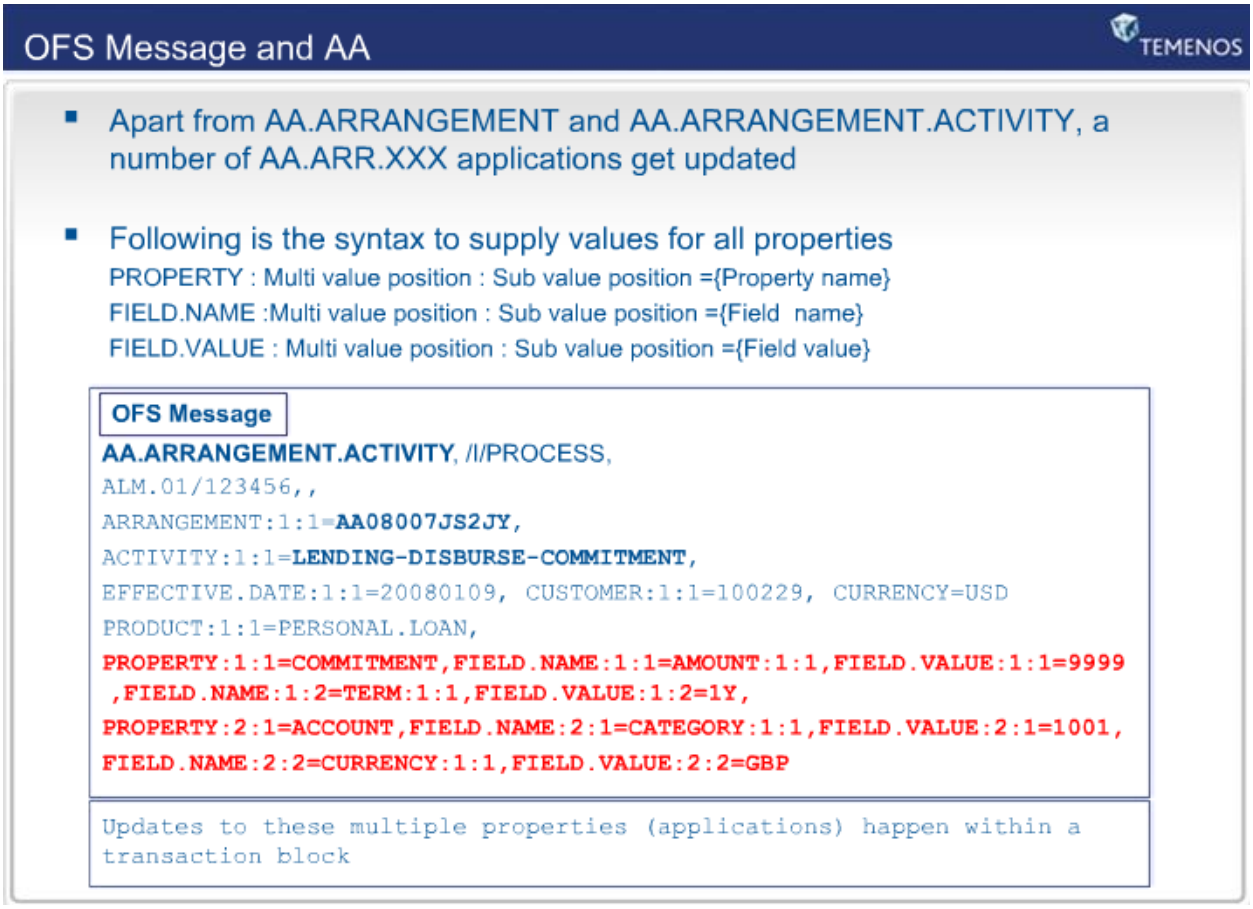
FIELD.NAME: Multi value position: Sub value position = {Field name}

FIELD.VALUE: Multi value position: Sub value position = {Field value}

Why do we have to specify PROPERTY, FIELD.NAME and FIELD.VALUE?.

These fields belong to the AA.ARRANGEMENT.ACTIVITY application. These fields are used by AA.ARRANGEMENT.ACTIVITY to import the supplied property values into the respective properties for that arrangement. This is because it is not possible to send an OFS request directed towards a specific property that is part of the arrangement.

OFS message Take a look at the entire OFS message to create a new arrangement in AA via OFS. As you can see the values for the various properties which in-turn update values in AA.ARR.XXX applications are supplied using the amended OFS syntax. If you have multiple properties for which values need to be supplied, then you may do so, by supplying values as shown. Updates to these multiple applications happen within a bulk transaction block.
Updates to these multiple applications happen within a bulk transaction block.

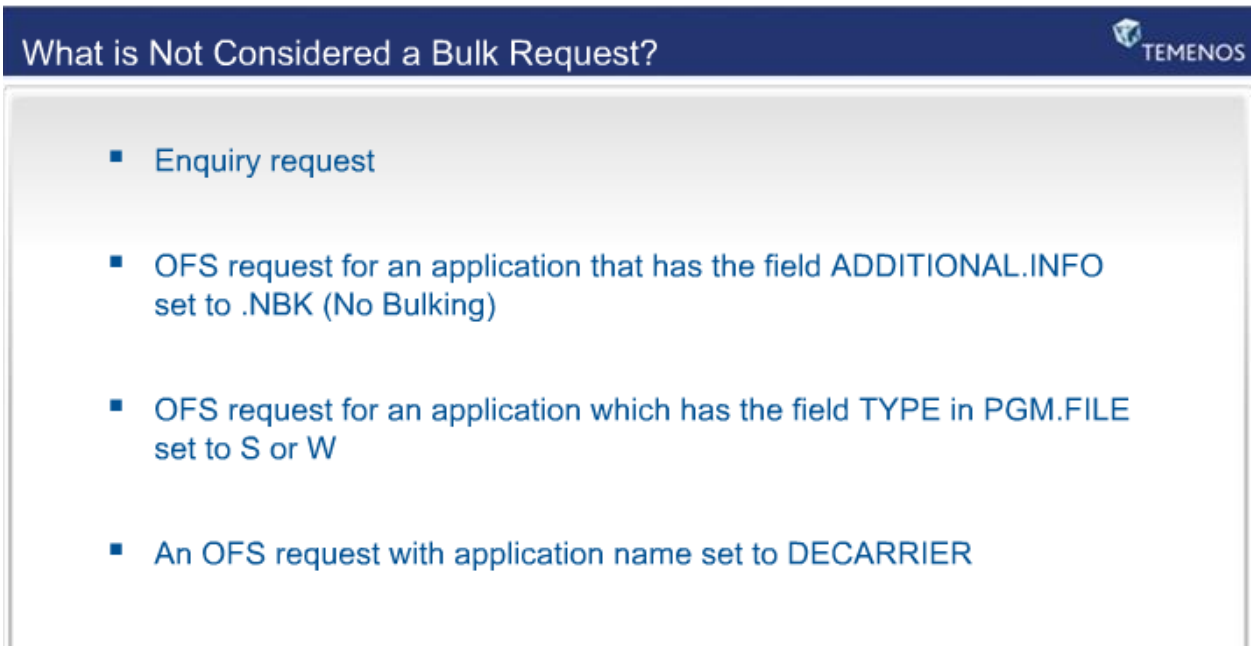


Any request to execute an enquiry is not considered a bulk request. An enquiry request just picks data from the database and does not update data in a database. For enquiry requests, the application name in OFS would be ENQUIRY.SELECT.

Applications in T24 can explicitly state that they do not want OFS.BULK.MANAGER to handle transaction management. In such a case, these applications will the field ADDITIONAL.INFO set to .NBK (No Bulking) in their PGM.FILE record. Example: AA.PRODUCT.MANAGER.

OFS request for an application which has the field TYPE in PGM.FILE set to S or W. Records in PGM.FILE with TYPE set to S denote that they are subroutines and TYPE set to W denotes that they are work files (Meaning you can verify records in such applications). For such type of applications, there is no need to bulk and hence not considered a bulk request.

An OFS message for delivery which will have application name in the OFS message set to DECARRIER will not be bulked. Please note that, such requests are internally handled by OFS.DE.REQUEST.



Encryption and decryption of OFS messages

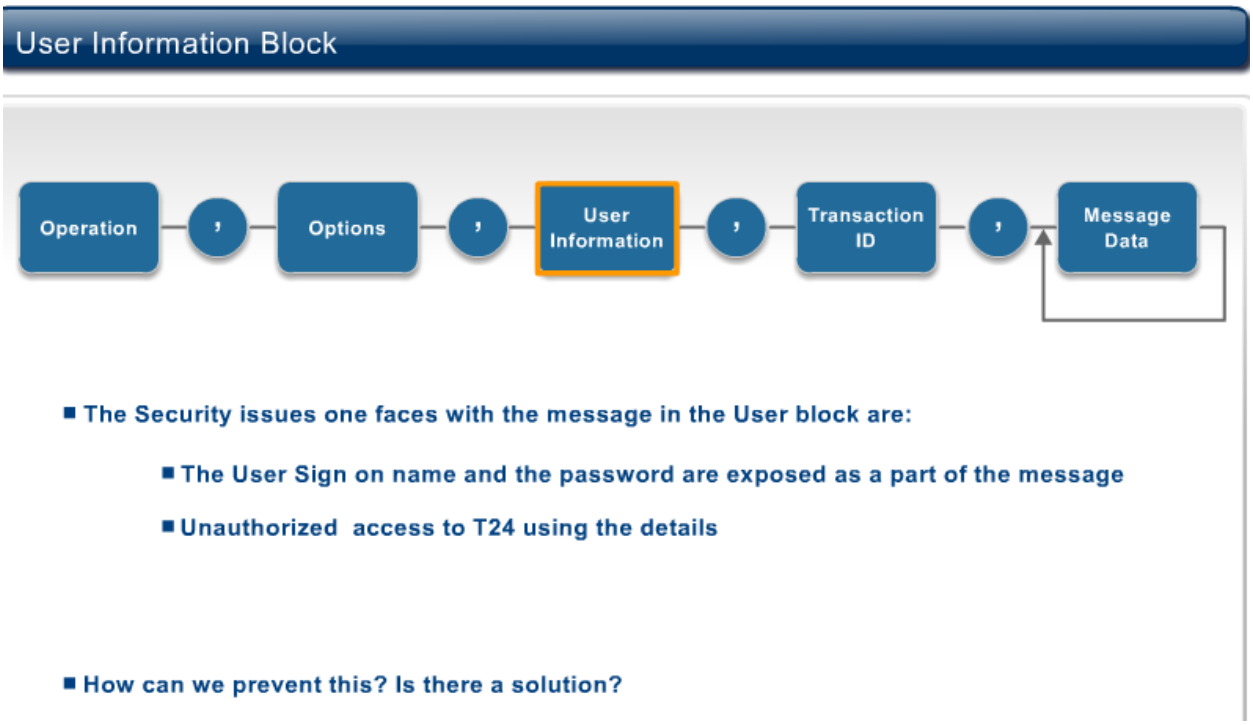
Our focus for this learning unit is to prevent sensitive information such as the sign on name and the password, from being exposed in OFS messages.

The Security issues one faces with the message in the User block are:

The User Sign on name and the password are exposed as a part of the message.

Unauthorized access to T24 using the details. When OFS messages are currently passed to T24, they contain T24 user sign on name and password details in order to perform any tasks within T24. The password is currently a clear text, meaning that anyone who manages to intercept an OFS message will then have unauthorized login details to T24.

How can we prevent this? Is there a solution?, Yes.

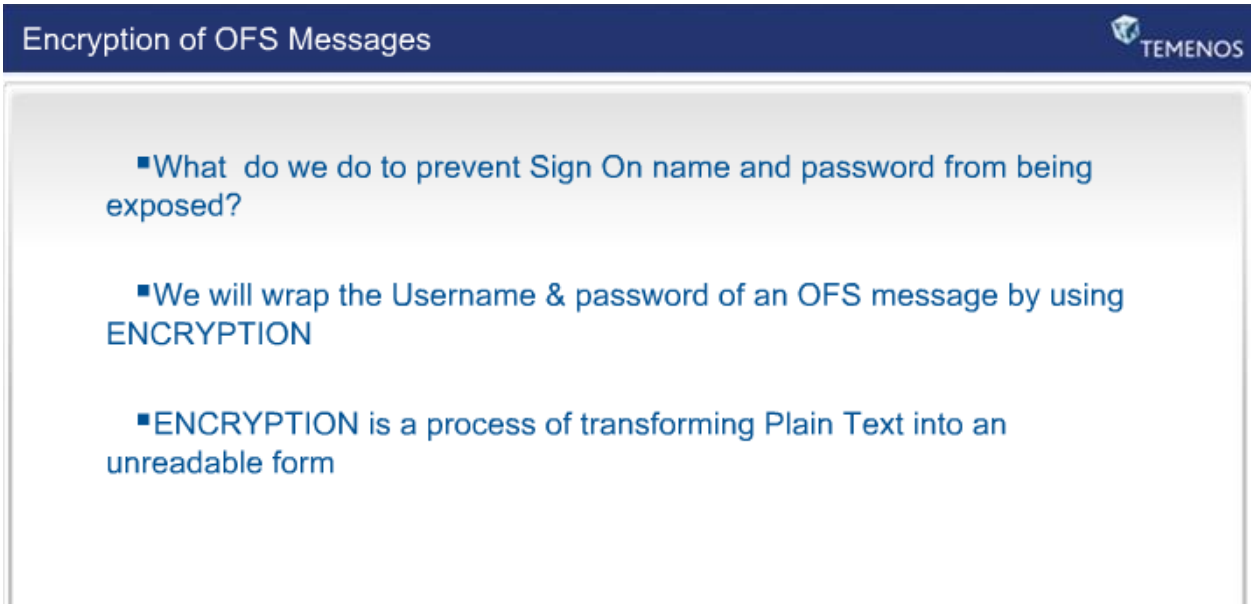


How we do to prevent the Sign on name and password from being exposed?

We will wrap the Username & password of an OFS message by using ENCRYPTION.

In order to emphasize security over confidential and sensitive information like the T24 login details, encryption has to be implemented to convert the plain text to an unreadable form.

Encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as the key. The result of the process is encrypted information (generally referred to as cipher text). We will wrap the user name and password using encryption.



Which part of the OFS Message are we going to encrypt?

At this point of time, your answer would be – The User information block.

The Sign ON Name and the Password is contained in the user information block.

But the encrypted password could be put into another message if intercepted. Instead we must encrypt the whole OFS message. It is not sufficient to encrypt just the password in the OFS message. This is because the encrypted password could be put into another message if intercepted. Instead we must encrypt the whole OFS message. This could be compared to acquiring a lock on the entire OFS message.

You must be having a question in mind right now.

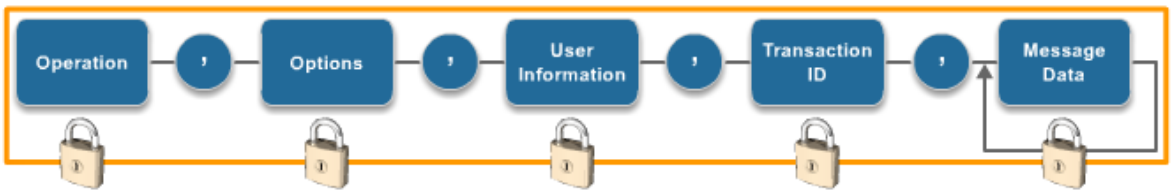
What if an unauthorized User tries to send the whole encrypted OFS message?

This does still give the ability to replay the whole message that is encrypted.Well, This would be caught as a duplicate transaction within T24.

Encrypting OFS Message as a Whole

- Which part of the OFS message are we going to encrypt?

  Your answer at this point of time would be – The Sign ON Name & Password
  But the encrypted password could be put into another message if intercepted



- You must be having a question in mind right now

  What if an unauthorized User tries to send the whole encrypted OFS message?

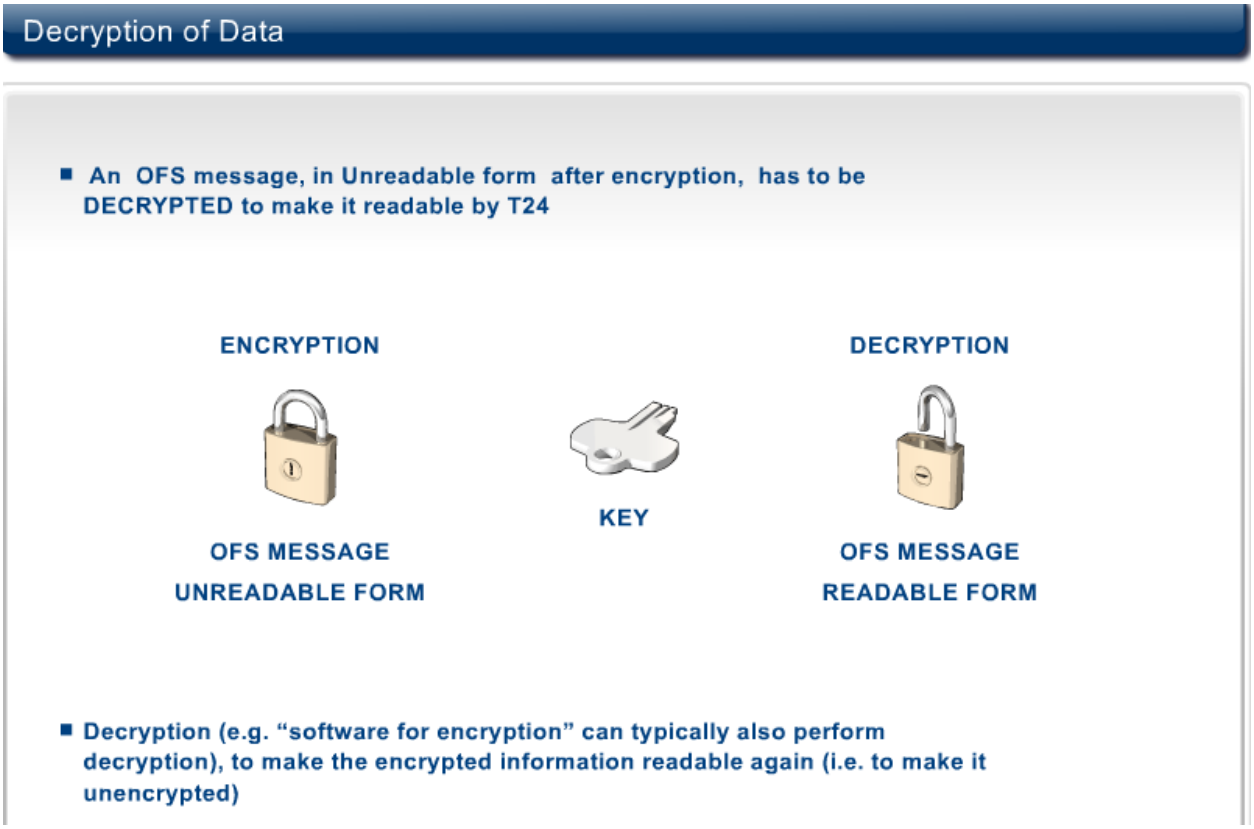
  This would be caught as a duplicate transaction within T24

An OFS message, in Unreadable form after encryption, has to be DECRYPTED to make it readable by T24.

After encrypting the request, it should again be transformed into the readable form. The lock that was acquired by the message has to be removed. The lock needs a key to be opened.

The process of using a key and removing the lock on the message is termed as “Decryption” . Decryption is the process of transforming encrypted data in to readable form . The software that is used for encryption can typically also perform decryption using the same logic.

When the OFS message is passed on to T24, OFS must decrypt before passing to Core T24 for authentication.



What is a key?

Key is a code/value that will lock/unlock the encrypted/decrypted Message.

We could use the same key or different keys for encryption and decryption.

The are two kinds of keys : Public Key : Key that is used for encryption. It may be widely distributed.

Private Key: A Secret key which is used for decryption. The Public & Private keys are related mathematically, but the private key cannot be practically derived from the public key.

The theory behind the concept of encryption and decryption is termed as "Cryptography". Cryptography is broadly classified in to two types: Asymmetric Cryptography: Also known as Public-key cryptography. The key used to encrypt a message differs from the key used to decrypt it. In Public key Mechanism, a public key must be passed to a trusted user who legitimately creates OFS messages. The user will use the key to encrypt the OFS message. In public key encryption, the message will be decrypted using private key which is never provided to third parties. The message cannot be decrypted with the public key that was initially used to encrypt it. This solution also solves security issues.

Public key is passed securely to a legitimate creator of OFS messages.

External user/application creates an encrypted OFS message using this public key.

OFS decrypts message and then passes the interpreted message to T24 as a normal OFS message.

Symmetric Cryptography: It uses single secret key for both encryption and decryption. The sender and receiver must share a key in advance. We will use this method to test decryption in all our workshops, though this new feature of decryption is designed to incorporate asymmetric cryptography.

A message encrypted by a public key has to be decrypted by its corresponding private key.

Keys

- Key is a code/value that will lock/unlock the encryption/decryption
- There are two kinds of keys:
  - Public Key : Key that is used for encryption. It may be widely distributed
  - Private Key: A Secret key used for decryption

PUBLIC KEY – ENCRYPTION

PUBLIC KEY – ENCRYPTION

- A message encrypted by a public key has to be decrypted by its corresponding private key

Different ways by which an OFS String is Encrypted

SOURCES OF ENCRYPTING OFS STRING	T24 SETTINGS TO BE DONE
OFS message encrypted by third party software	PSWD.ENCRYPTED in OFS.SOURCE must be blank. OFS.MESSAGE.DECRYPT is set to YES DECRYPT.KEY must be configured
Encryption by LDAP	PSWD.ENCRYPTED should be set to YES
Encryption by attaching a hook routine to ENCRYPTION ALGORITHM field in SPF	PSWD.ENCRYPTED in OFS.SOURCE must be blank
Encryption using T24 proprietary logic	PSWD.ENCRYPTED in OFS.SOURCE must be blank

Functions used in jBASE Basic

ENCRYPTION	DECRYPTION
Transforming Plain Text into an unreadable form	Encrypted information is readable again
ENCRYPT(string, key, method)	DECRYPT(string, key, method)
Encoding done after encryption using BASE64 algorithm	Decoding done using BASE64 algorithm prior to decryption

- String specifies the string to be encrypted/decrypted
- Key is the value used to encrypt/decrypt the string. Its use depends on method
- Method is an algorithm which indicates the encryption/decryption mechanism to use. It is internally converted to a numeric value

Functions used in jBASE Basic

ENCRYPTION	DECRYPTION
Transforming Plain Text into an unreadable form	Encrypted information is readable again
ENCRYPT(string, key, method)	DECRYPT(string, key, method)
Encoding done after encryption using BASE64 algorithm	Decoding done using BASE64 algorithm prior to decryption

- String specifies the string to be encrypted/decrypted
- Key is the value used to encrypt/decrypt the string. Its use depends on method
- Method is an algorithm which indicates the encryption/decryption mechanism to use. It is internally converted to a numeric value

Applications Used

TEMENOS

The applications used in OFS to implement encryption are:

- OFS.SOURCE – To enable decryption for OFS Messages.
- OFS.DECRYPT.KEY – Provides detailed description of the key.

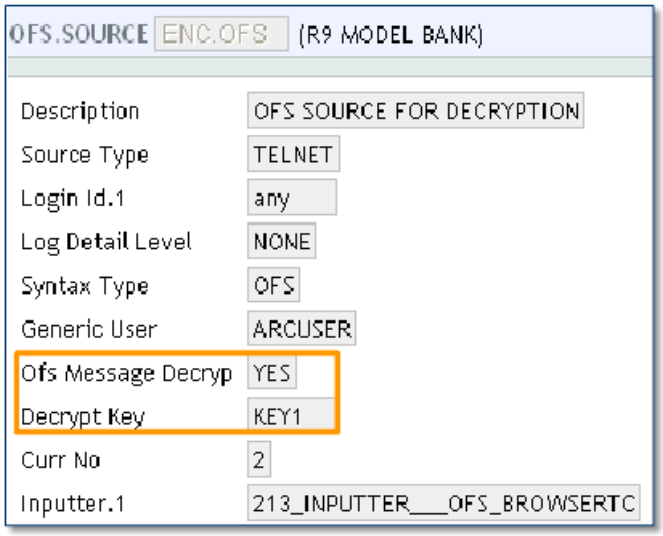
First application that is to be discussed is OFS.SOURCE. Apart from the fields like SOURCE TYPE, SYNTAX TYPE that we already use to set a basic OFS SOURCE, there are two special fields that we configure to support decryption.

There are two fields in OFS.SOURCE which has to be populated in order to enable decryption. The fields are: a. OFS.MESSAGE.DECRYPT

b. DECRYPT.KEY

OFS.MESSAGE.DECRYPT - It is more like a flag that represents whether Decryption should take place or not. If it is set to “YES” if encryption has to be enabled.

If OFS.MESSAGE.DECRYPT - is set to 'Yes', then we need to give a valid record ID of OFS.DECRYPT.KEY application which would contain the necessary information for Decryption.



KEY FIELDS WITH SAMPLE DATA	
FIELDS	SAMPLE DATA
OFS MESSAGE DECRYP	YES
DECRYPT KEY	RECORD ID OF OFS.DECRYPT.KEY

To generate the key, log in to bin of jdk and use the command Key Tool(jsh r0812\C:\java\jdk keytool).

Key tool is the command for generation of private key.Key tool is also a command which is used for listing the keys generated . Keys generated using :

MD5(Message Digest Algorithm 5 method) and, SHA1(Secure Hash Algorithm method).

The key value that is used in OFS decrypt key could also be a user-defined value.

Let us take a closer look at the second application – OFS.DECRYPT.KEY.

OFS.DECRYPT.KEY is an INT file of type "H" in PGM.FILE (Similar to OFS.SOURCE)

In OFS.DECRYPT.KEY, the fields that are populated are:

DESCRIPTION holds the description of the OFS.DECRYPT.KEY record.

MSG.DECRYPT.KEY - This field will contain the key with which the decryption will take place.

CIPHER.METHOD - This field will holds the method that takes our desired Decryption method.

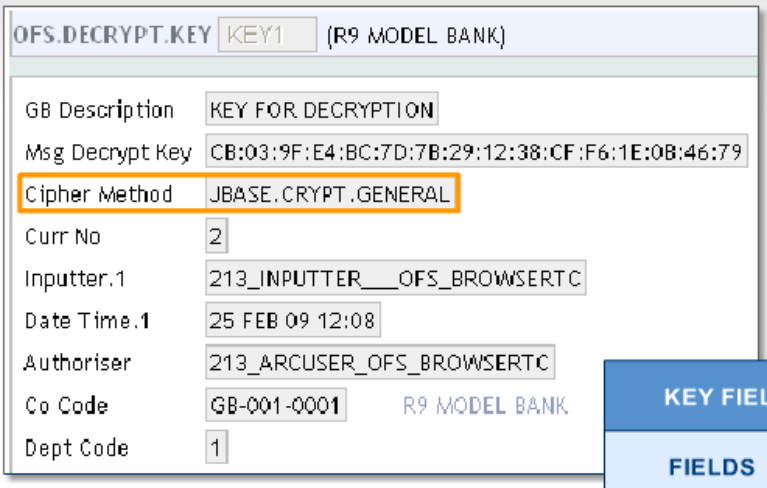
The options for CIPHER.METHOD is provided with a Drop down list box. The cipher methods are given with 'dots'(.) replacing the 'underscores'(_) use in JBC.H header file. For instance, JBASE_CRYPT_GENERAL will be input as JBASE.CRYPT.GENERAL in OFS.DECRYPT.KEY but when it has to be used as a parameter in the routine, it is converted back to JBASE_CRYPT_GENERAL. This is the method that is defaulted when no option is chosen.

Note that When Cipher method that is selected is ROT13(rotate by 13 places), it will not accept a key value since encryption happens by picking up the 13th position in rotation. ROT13 is its own inverse - that is, undo ROT13, the same algorithm is applied, so the same action can be used for both encoding and decoding. The algorithm provides no real cryptographic security and should never be used as such. It is often cited as a canonical example of weak encryption.

Generation of a key

```
jsh r0812 ~ -->keytool -genkey -v -alias richu -keyalg RSA -validity 3650 -keys
tore d:\key\richu.keystore -storepass password -keypass richupasss
What is your first and last name?
  [Unknown]:  TRAINER
What is the name of your organizational unit?
  [Unknown]:  CORPORATE TRAINING
What is the name of your organization?
  [Unknown]:  TEMENOS
What is the name of your City or Locality?
  [Unknown]:  CHENNAI
What is the name of your State or Province?
  [Unknown]:  TN
What is the two-letter country code for this unit?
  [Unknown]:  TN
Is CN=TRAINER, OU=CORPORATE TRAINING, O="TEMENOS ", L=CHENNAI, ST=TN, C=TN corre
ct? (type "yes" or "no")
  [no]:  YES
Generating 1,024 bit RSA key pair and self-signed certificate (MD5WithRSA)
        for: CN=TRAINER, OU=CORPORATE TRAINING, O="TEMENOS ", L=CHENNAI, ST=TN,
C=TN
[Storing d:\key\richu.keystore]
jsh r0812 ~ -->keytool -list -v -alias richu -keystore d:\key\richu.keystore -s
torepass password -keypass richupass

Alias name: richu
Creation date: Feb 2, 2009
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=TRAINER, OU=CORPORATE TRAINING, O="TEMENOS ", L=CHENNAI, ST=TN, C=TN
Issuer: CN=TRAINER, OU=CORPORATE TRAINING, O="TEMENOS ", L=CHENNAI, ST=TN, C=TN
Serial number: 4986796d
Valid from: 2/2/09 10:11 AM until: 1/31/19 10:11 AM
Certificate fingerprints:
         MD5:  CB:03:9F:E4:BC:7D:7B:29:12:38:CF:F6:1E:0B:46:79
         SHA1: 45:33:EF:47:C9:E9:05:86:78:62:02:5C:B0:86:0E:9B:C7:8B:48:14
```

OFS.DECRYPT.KEY



KEY FIELDS WITH SAMPLE DATA	
FIELDS	SAMPLE DATA
MSG DECRYPT KEY	KEY GENERATED USING KEYTOOL COMMAND
DECRYPT KEY	JBASE.CRYPT. GENERAL

Task - 1

TEMENOS

- Write an OFS transaction request to create an new account for the customer 100724

A. Encode the String using ENCRYPT function with User input obtained during run time

Hint : Use the numerical equivalent for the Cipher Methods & the key generated using the key tool from OFS.DECRYPT.KEY

B. Use JBASE_CRYPT_GENERAL as the Cipher method

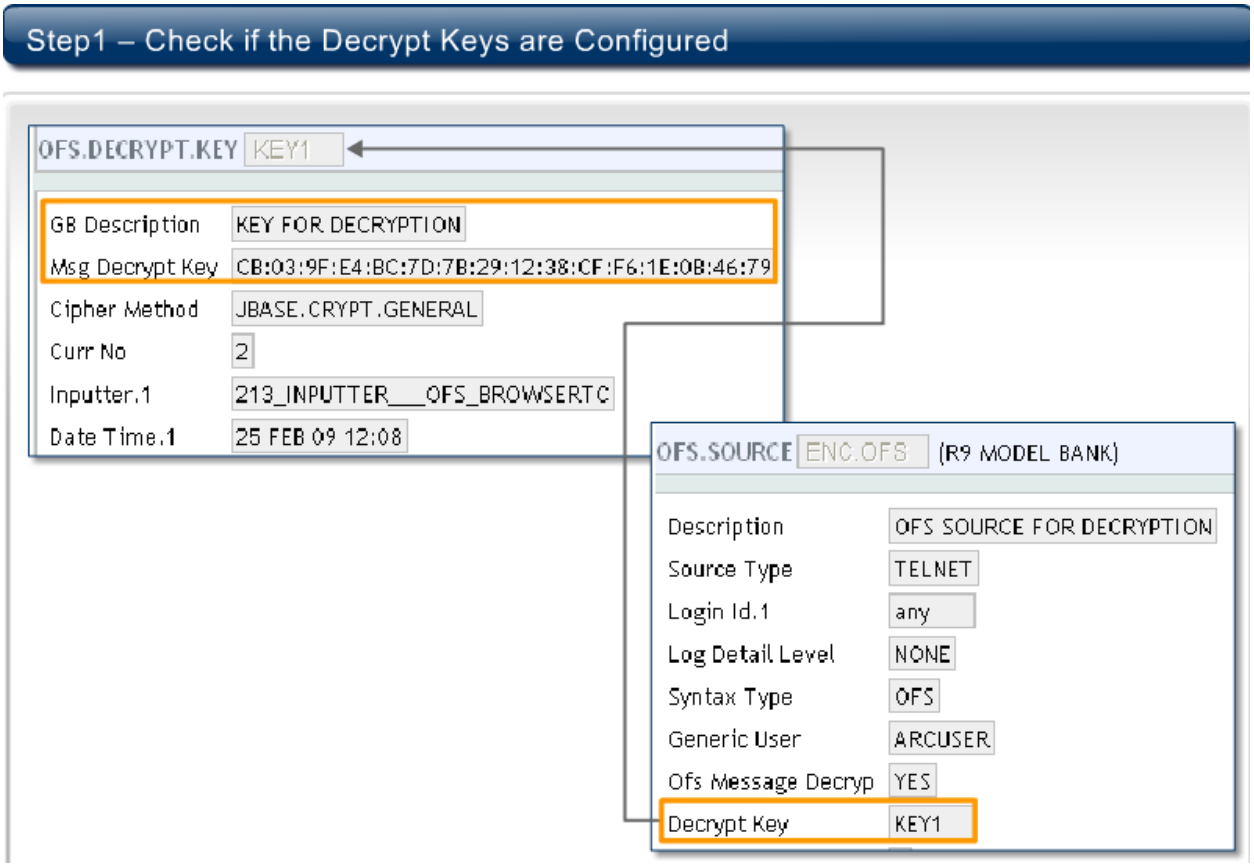
C. Use TELNET mode to process the encoded string

Step-1:

Configure T24 applications for enabling decryption.

Set up OFS.DECRYPT.KEY first, since the record id has to be mentioned in OFS.SOURCE. The Decrypt key value is the value that is generated by the keytool command but as mentioned earlier, it could be any user-defined value.

Populate values in OFS.SOURCE. Make sure you turn on decryption by setting OFS MESSAGE DECRYP field to YES and, the DECRYPT KEY is the record id of OFS.DECRYPT.KEY application.



Step – 2 Encryption Using a Program

```
PROGRAM CODE.ENCRYPT

$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.OFS.SOURCE
$INSERT I_F.OFS.DECRYPT.KEY
$INSERT I_GTS.COMMON
INCLUDE JBC.h    ;*HEADER FILE

GOSUB INIT
GOSUB OPEN
GOSUB USERINPUT
GOSUB GETVALUE
GOSUB ENCRYPT
GOSUB RESULT
```

```
INIT:
     F.OFS = ''
  F.KEY = ''
     KEY1 = ''
     METHOD = ''
  R.OFS.DECR.KEY = ''
     YERR1 = ''
     YERR2 = ''
RETURN
OPEN:
     CALL OPF('F.OFS.SOURCE',F.OFS)
     CALL OPF('F.OFS.DECRYPT.KEY',F.KEY)
RETURN
USERINPUT:
     CRT "ENTER STRING"         ;* Get the OFS message string
     INPUT STR1
```

```
   CRT "ENTER OFS.SOURCE ID" ;* Get the OFS.SOURCE record
   INPUT SRC1
   RETURN
GETVALUE:
     CALL
F.READ('F.OFS.SOURCE',SRC1,R.OFS.SOURCE,F.OFS,YERR1)
   OFS$SOURCE.REC = R.OFS.SOURCE
   OFS$SOURCE.ID = OFS$SOURCE.REC<1>
*GET RECORD ID OF OFS.DECRYPT.KEY FROM OFS.SOURCE
   OFS.DECR.KEY.ID = OFS$SOURCE.REC<OFS.SRC.DECRYPT.KEY>
   CALL
F.READ('F.OFS.DECRYPT.KEY',OFS.DECR.KEY.ID,R.OFS.DECR.KEY,F.KEY,YE
RR2)
*GET ACTUAL KEY VALUE FROM OFS.DECRYPT.KEY
   KEY1 = R.OFS.DECR.KEY<OF.DECR.MSG.DECRYPT.KEY>
RETURN
```

```
ENCRYPT:
* encrypting
 ENCR.MSG = ENCRYPT(STR1,KEY1,0)
* jBASE encoding
   ENCD.MSG = ENCRYPT(ENCR.MSG, '', JBASE_CRYPT_BASE64)
RETURN
RESULT:
   CRT "*****************************************"
   CRT "KEY USED:      ": KEY1
   CRT "ENCRYPTED MSG: ": ENCR.MSG
   CRT "ENCODED MSG:   ": ENCD.MSG
   CRT "*****************************************"

END
```

Execute the program. There are 2 inputs required. The OFS String to be encrypted and the OFS.SOURCE record id which will be used for decryption. The encoded message is obtained as the final output of the program.

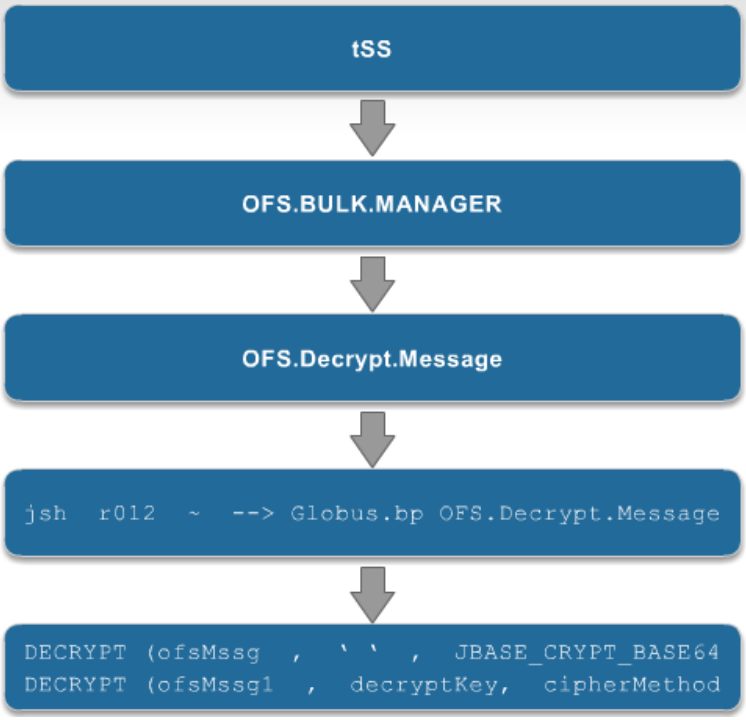
When we invoke OFS.SOURCE through tss, instead of the actual OFS message, we place the encoded string. Note that the response is that of an account being created.

Step – 3 Decryption by OFS.Decrypt.Message

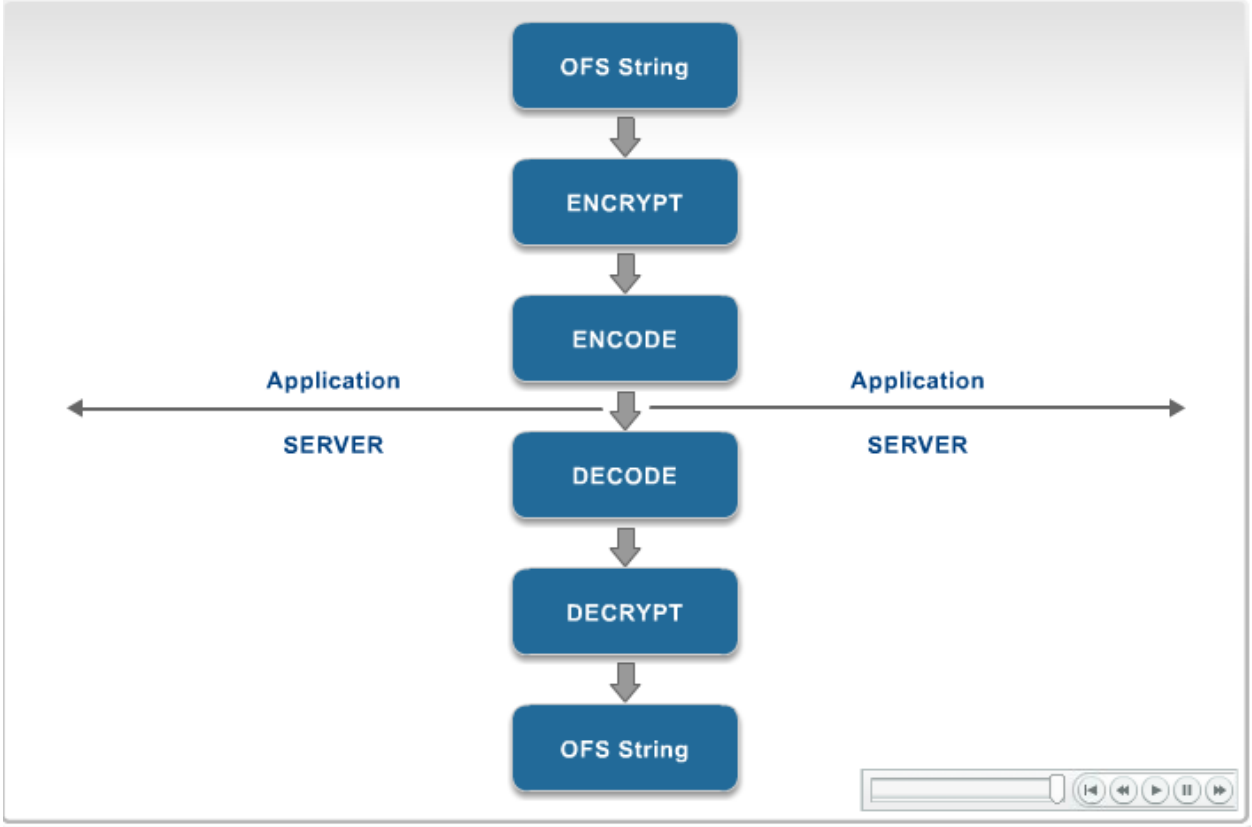
```
jsh r0812 ~ -->CODE.ENCRYPT
ENTER STRING
?ACCOUNT,/I/PROCESS//0,PRIYATS/654321,,CUSTOMER=100297,CATEGORY=1001,CURRENCY=USD
ENTER OFS.SOURCE ID
?ENC.OFS
**********************************************************************
KEY USED:       CB:03:9F:E4:BC:7D:7B:29:12:38:CF:F6:1E:0B:46:79
ENCRYPTED MSG: âµl4Ð¨T.|ÕF>^WÃ÷5ãÖ¤nÃÇjiöù]3¦Ú¶#gVÄ·ø²  W²ÿ~i>
«ÊA?-T3ö5Ýë$lÊ±uVú»yA
ENCODED MSG:    4rVsNNAYqFQufJfVRj6ZXlfD9zXj1qRuw8dqafb5nlOzptq2lyOZZ1bEt5L4EJGyoFcVsv9+aT6Fq
8pBP61UM572Nd3rp2zKsXVW+gW7eUE=
**********************************************************************
```

```
jsh r0812 ~ -->tSS ENC.OFS
<tSS version="1.1"><t24version>200812</t24version><t24pid>84004</t24pid><t24ofssource>ENC.OF
S</t24ofssource><clientIP/></tSS>
4rVsNNAYqFQufJfVRj6ZXlfD9zXj1qRuw8dqafb5nlOzptq2lyOZZ1bEt5L4EJGyoFcVsv9+aT6Fq8pBP61UM572Nd3r
p2zKsXVW+gW7eUE=
41017/ENC080090007261649.01/1,CUSTOMER=100297:1:1,CATEGORY=1001:1:1,ACCOUNT.TITLE.1=Richard
Branson:1:1,SHORT.TITLE=Richard Branson:1:1,POSITION.TYPE=TR:1:1,CURRENCY=USD:1:1,CURRENCY.M
ARKET=1:1:1,ACCOUNT.OFFICER=34:1:1,CONDITION.GROUP=1:1:1,CAP.DATE.CHARGE=20080131:1:1,PASSBO
OK=NO:1:1,OPENING.DATE=20080109:1:1,OPEN.CATEGORY=1001:1:1,CHARGE.CCY=USD:1:1,CHARGE.MKT=1:1
:1,INTEREST.CCY=USD:1:1,INTEREST.MKT=1:1:1,ALT.ACCT.TYPE=LEGACY:1:1,ALLOW.NETTING=NO:1:1,SIN
GLE.LIMIT=Y:1:1,CURR.NO=1:1:1,INPUTTER=72_PRIYA2___OFS_ENC.OFS:1:1,DATE.TIME=0902051707:1:1,
AUTHORISER=72_PRIYA2_OFS_ENC.OFS:1:1,CO.CODE=GB0010001:1:1,DEPT.CODE=1:1:1
```

How did this Happen?



Work Flow Summary



Workshop 1 – Encryption & Decryption with Same Key & Method

TEMENOS

- Write an OFS Enquiry request to display all the customer records

A. Encode the String using ENCRYPT function with User input obtained during run time

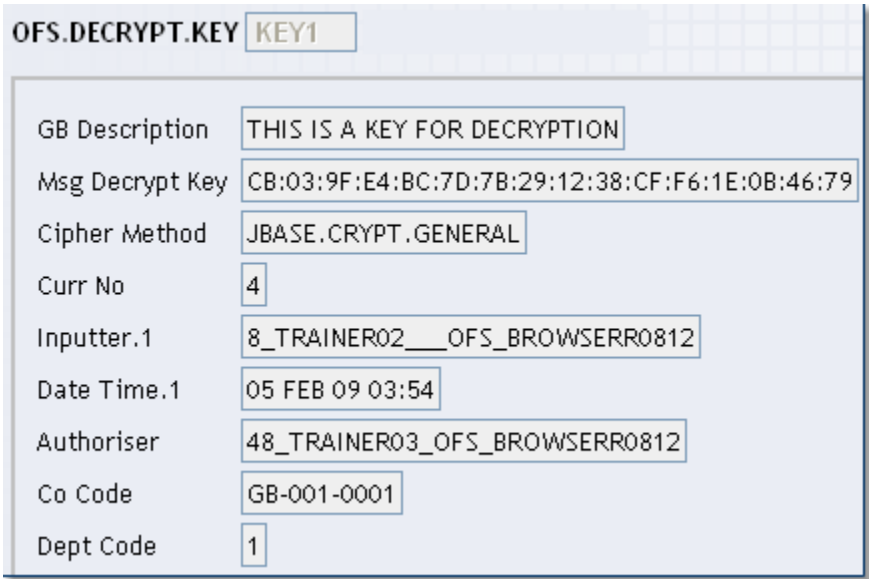
B. Use Cipher Method – JBASE.CRYPT.GENERAL for encryption & decryption

C. Use the same keys for Encryption & Decryption

D. Use TELNET mode of processing

Solution

```
ENCRYPT:
    ENCR.MSG = ENCRYPT(STR1,KEY1,0)     ;* encrypting
    ENCD.MSG = ENCRYPT(ENCR.MSG, '', JBASE_CRYPT_BASE64)
```




```
jsh r0812 ~ -->CODE.ENCRYPT
ENTER STRING
?ENQUIRY.SELECT,,PRIYATS/654321,%CUSTOMER
ENTER OFS.SOURCE ID
?ENC.OFS
**********************************************************************
KEY USED:       CB:03:9F:E4:BC:7D:7B:29:12:38:CF:F6:1E:0B:46:79
ENCRYPTED MSG: æÄ~>È ±ZV|¸Î;GIXÆcø«O´§aLÙÿQFÌÿÝ¾Hh
ENCODED MSG:    5sR+PsggsVpWfLjOOOeGSVjGFWP4AqtPtKdhTNn/mFFGzP/dvkifaA==
**********************************************************************
jsh r0812 ~ -->tSS ENC.OFS
<tSS version="1.1"><t24version>200812</t24version><t24pid>84004</t24pid><t24ofss
ource>ENC.OFS</t24ofssource><clientIP/></tSS>
5sR+PsggsVpWfLjOOOeGSVjGFWP4AqtPtKdhTNn/mFFGzP/dvkifaA==
```

Workshop 2 - Same Keys & Different Cipher Methods

TEMENOS

For the enquiry message in Workshop 1

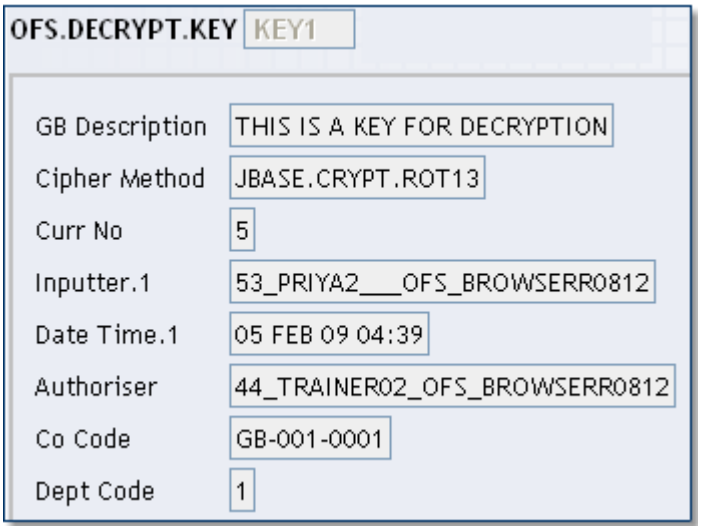
A. Use DES Algorithm for Encryption

B. Use ROT13 Algorithm for Decryption

C. Use TELNET Mode of processing

D. Use the key in OFS.DECRYPT.KEY for both Encryption & decryption

```
ENCRYPT:
    ENCR.MSG = ENCRYPT(STR1,KEY1,5)     ;* encrypting
    ENCD.MSG = ENCRYPT(ENCR.MSG, '', JBASE_CRYPT_BASE64)     ;* jBASE encoding
```



```
jsh r0812 ~ -->CODE.ENCRYPT
ENTER STRING
?ENQUIRY.SELECT,,PRIYATS/654321,%CUSTOMER
ENTER OFS.SOURCE ID
?ENC.OFS
**********************************************************************
KEY USED:
ENCRYPTED MSG:
xÈô
   eö½¤à¶~²xt
ENCODED MSG:   hBWbZrNA4vTpj1GOrd9tpMiY+Hr6kIBGObaXAQybOXjI9AsVZfa9pB7gtn6ylnhO
**********************************************************************
jsh r0812 ~ -->tSS ENC.OFS
<tSS version="1.1"><t24version>200812</t24version><t24pid>84004</t24pid><t24ofss
hBWbZrNA4vTpj1GOrd9tpMiY+Hr6kIBGObaXAQybOXjI9AsVZfa9pB7gtn6ylnhO
INVALID/ NO SIGN ON NAME SUPPLIED DURING SIGN ON PROCESS
```

Points to Remember

TEMENOS

- Encryption/Decryption is done with different , only when we generate Public key and a corresponding Private key which are mathematically related
- When SOURCE TYPE field of OFS.SOURCE is set to SESSION, and the SYNTAX TYPE is OFS, only enquiry requests can be decrypted
- When SOURCE TYPE field of OFS.SOURCE is set to SESSION, and the SYNTAX TYPE is XML, only requests from browser can be decrypted
- OFFLINE requests(BATCH mode) cannot be decrypted using OFS.Decrypt.Message routine as it is called by OFS.BULK.MANAGER which is invoked by tSS

Nice to Know Stuff - Encryption Libraries

- The Encryption Libraries used are Java Libraries
- The java Libraries are called using the CALLJ function
- The wrapper can perform the encryption / decryption

Summary

- Encryption is done to the OFS message as a whole and not only the user information block
- tSS invokes OFS.BULK.MANAGER which in turn calls OFS.Decrypt.Message to decode & decrypt the encoded message
- This is not applicable to the Offline Batch mode of processing
- Decryption in SESSION mode(OFS syntax type) processes only encrypted enquiry requests
- The function ENCRYPT/DECRYPT and all the cipher methods used are defined in a file called JBC.h
- The applications used to enable decryption is OFS.SOURCE and the key description is in OFS.DECRYPT.KEY