Programming Using T24 APIs

TEMENOS EDUCATION CENTRE

TEMENOS
The Banking Software Company

Objectives

After completing this learning unit/course, you will be able to:

- Create subroutines using T24 APIs such as
  - OPF
  - F.READ
  - F.READU
  - F.WRITE
  - JOURNAL.UPDATE
  - F.RELEASE
  - EB.READLIST
  - CACHE.READ
  - EB.READ.PARAMETER
  - F.DELETE

Task

- Display the currency and category of account 11967

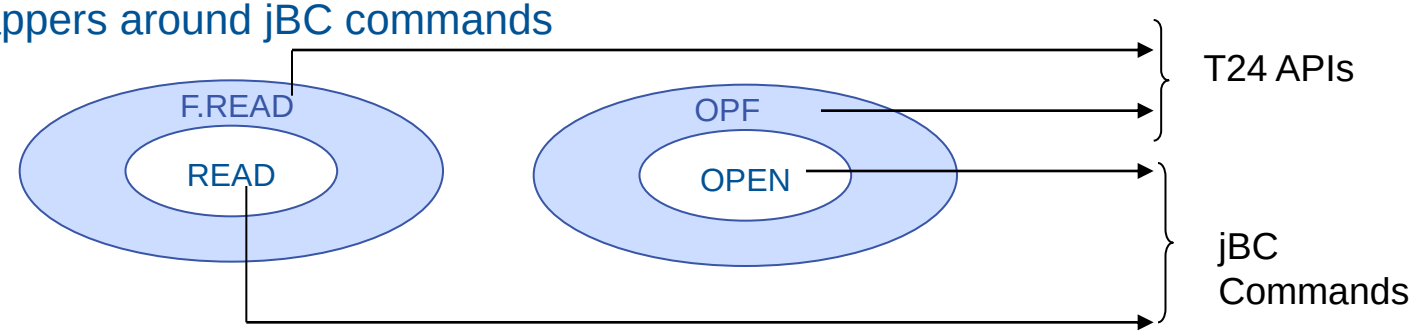
Algorithm

Programs can be executed from the database (jsh) prompt only

Subroutines are executed from within T24. In simple, any called routine from any executable should be a subroutine.

Action to be performed	jBASE command to be used
Open the customer file	OPEN
Read the customer record	READ
Display only mnemonic and sector	??

- T24 APIs
  - API stands for Application Programming Interface
  - Wrappers around jBC commands



- Subroutines
  - Executed from within T24 (Not from the jsh prompt)
  - Can make use of T24 APIs.

```
* Comments
SUBROUTINE Subroutinename
        Actual statements
        Actual statements
RETURN
END
```

Algorithm again!!!

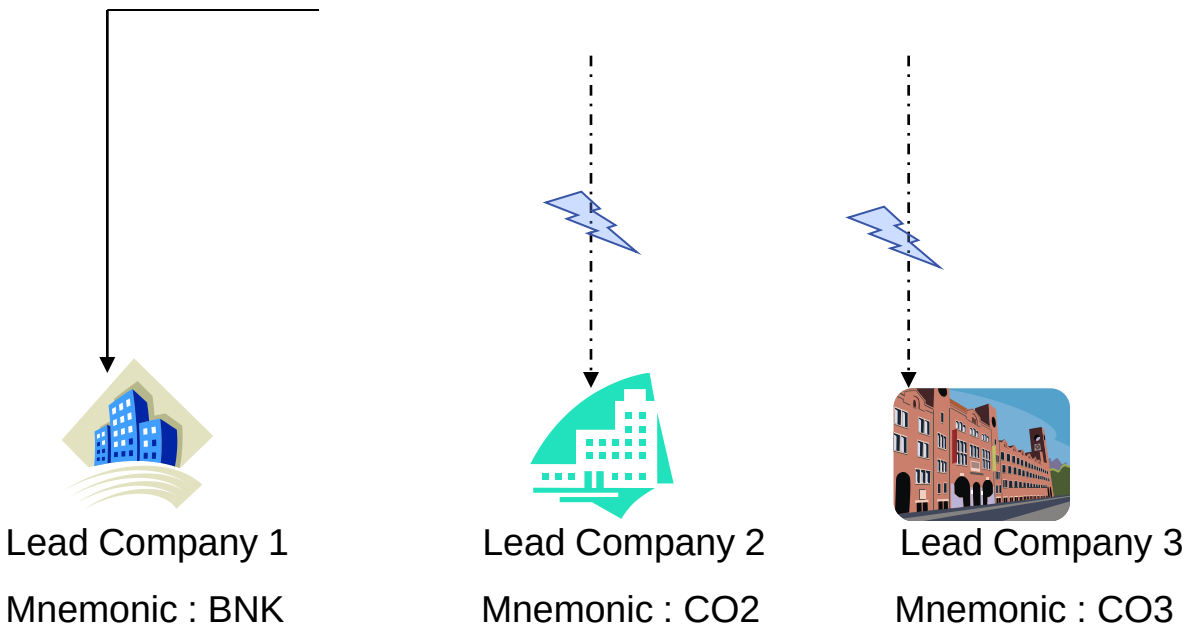
Algorithm to display the CATEGORY and CURRENCY of ACCOUNT 10693.

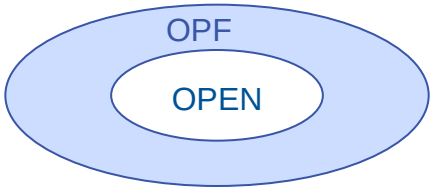
Subroutine to be created to achieve the task.

Action to be performed	jBASE command to be used
Open the ACCOUNT file	~~OPEN~~ OPF
Read the ACCOUNT record	~~READ~~ F.READ
Extract category and currency	?? You will learn as you proceed
Display category and currency	Use CRT to display

- Disadvantages of using the OPEN command
  - OPEN FBNK.ACCOUNT TO F.ACCOUNT THEN … ELSE…

    File names get hard coded
  - Code does not become portable in a multi company environment





- Stands for Open File
- Syntax

```
CALL OPF(Parameter1,Parameter2)
```

- Example

```
FN.ACC = 'F.ACCOUNT'   * File Name
F.ACC = '' * File Path

CALL OPF(FN.ACC,F.ACC) *Open the file
```

Program Variables and Common Variables

- Program variables
  - The scope of a variable used within a program is limited to the program. Meaning, the variable will loose its value when the program terminates
  - Any variable that is used within a program
- Common variables
  - Need to be defined as common
  - Values of these variables are lost only when the session is terminated

Hint to help you answer the question – I_COMMON

- What is I_COMMON?
  - It is a file under T24.BP
  - Contains the definition for most of the common variables used in
  - When do these variables get populated with values?

    Some get values when a user signs on

    Example :

    ID.COMPANY (ID of the user's currently signed on company)

    R.USER (Currently signed on user's record)

    R.COMPANY (Dimension array which holds the current company record)

    Applications populate data on to some variables

    Example :

    ID.NEW (ID of the currently opened record)

    R.NEW (Contents of the currently opened record)

Open I_COMMON under T24.BP using the Eclipse editor and view the details

Error Handling - OPF

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
FN.ACC='F.ACOUNT'
F.ACC=''
DEBUG
CALL OPF(FN.ACC,F.ACC)
CRT ETEXT
RETURN
END
```

Note the Error

```
0009      DEBUG
jBASE debugger->S
0010      CALL OPF(FN.ACC,F.ACC)
jBASE debugger->S
Invalid or uninitialised variable -- NULL USED ,
Var MNEMONIC , Line   456 , Source OPF

** FATAL ERROR IN (OPF) **
NO FILE.CONTROL RECORD - F.ACCONT , MNEMONIC =

jsh mbr8 ~ -->
```

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
FN.ACC='F.ACOUNT'
FN.ACC<2>="NO.FATAL.ERROR"
F.ACC=''
DEBUG
CALL OPF(FN.ACC,F.ACC)
CRT ETEXT
RETURN
END
```

Note the Error

```
0009      DEBUG
jBASE debugger->S
0010      CALL OPF(FN.ACC,F.ACC)
jBASE debugger->S
0011      CRT ETEXT
jBASE debugger->S
NO FILE.CONTROL RECORD
0012      RETURN
jBASE debugger->
```

Concept of sessions

```
-- LAST SIGN.ON, DATE: 16 MAR 2010
19 MAR 2010 09:28:22  USER (05 JAN)
ACTION
AWAITING APPLICATION
```

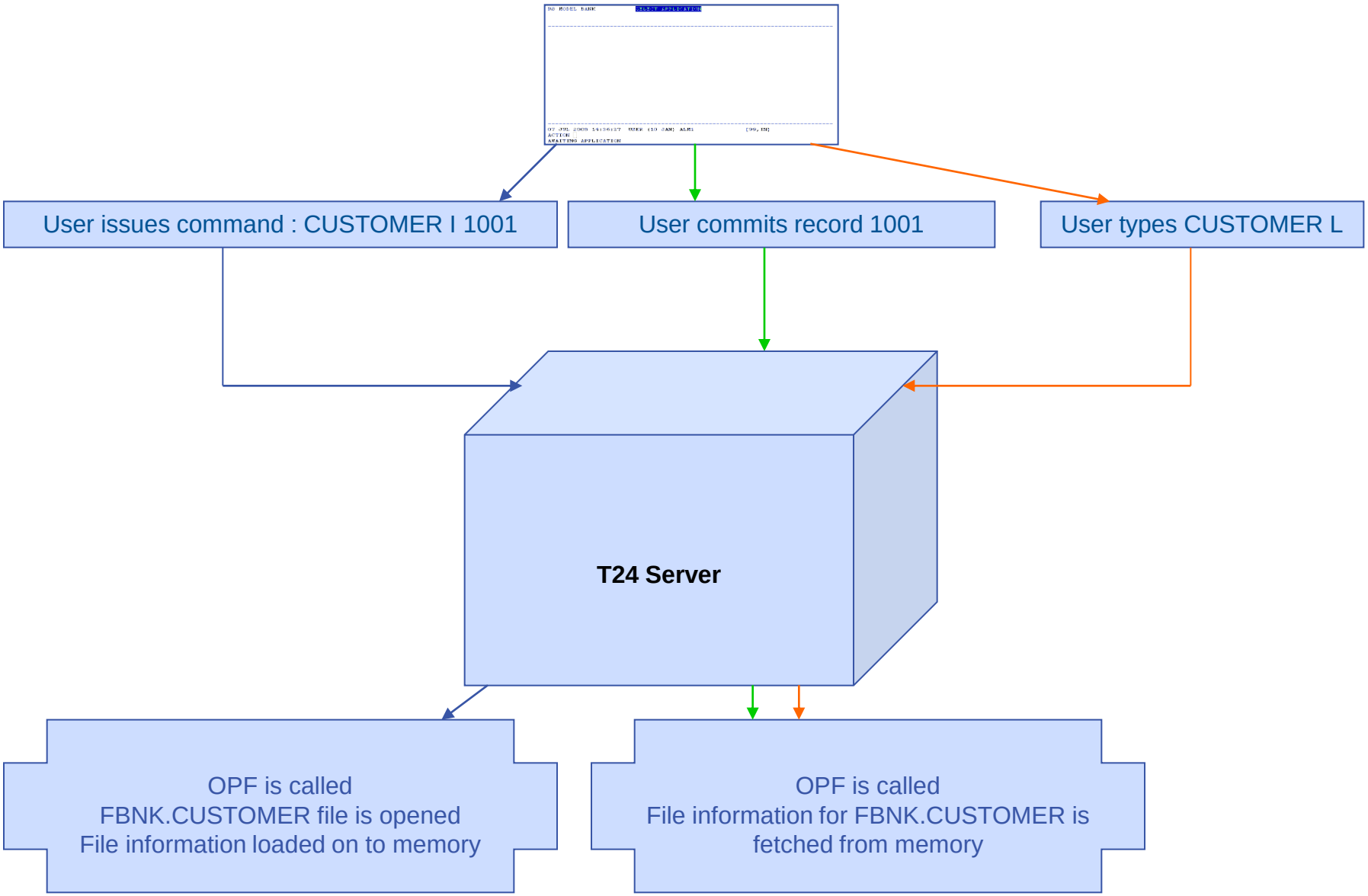
Classic user interface

- Connection always exists between the user and the server
- Session – Connection that exists between the user and the server
- A session is valid until the user is connected to the server

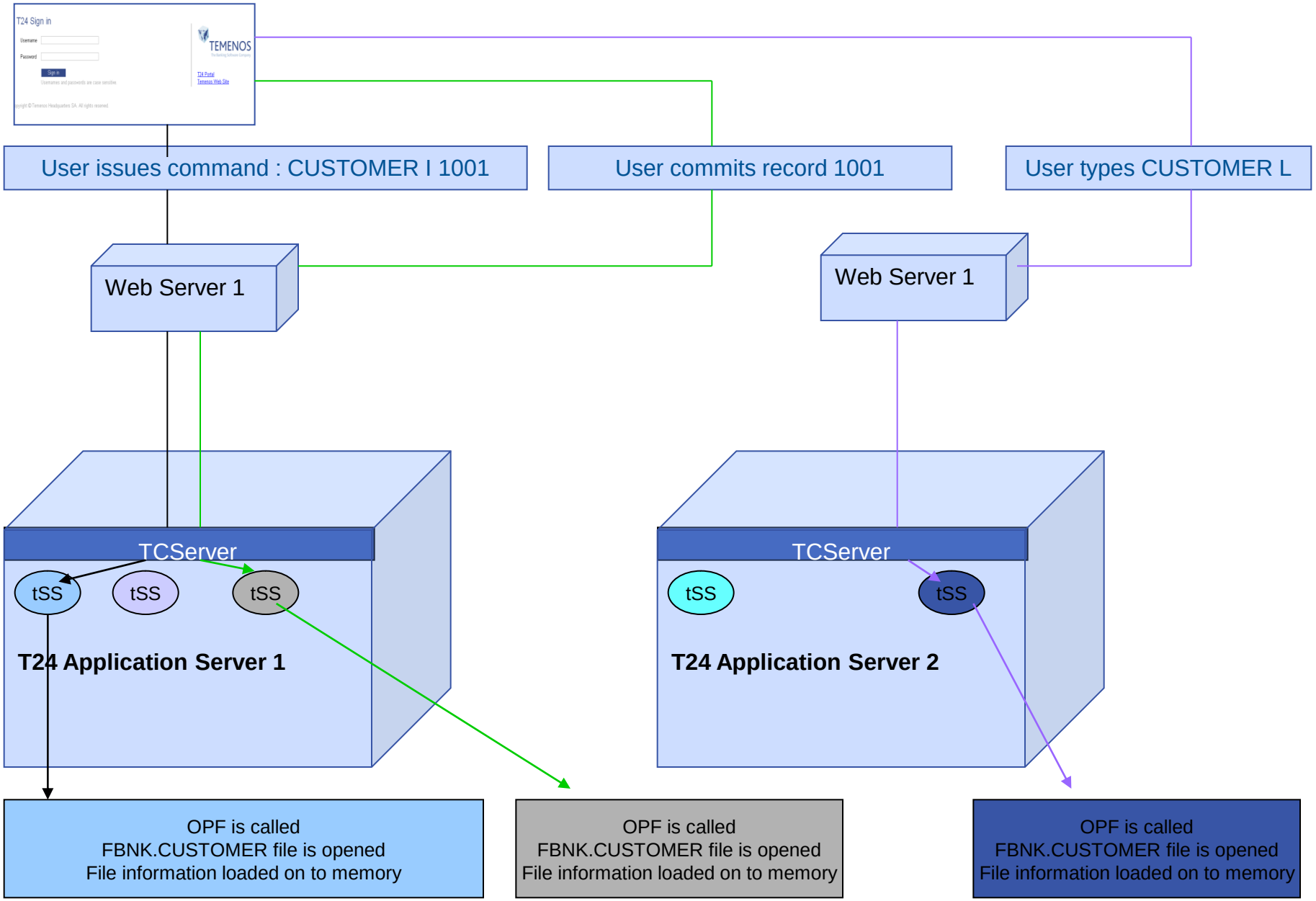
Browser

- Connection between user and T24 application server is stateless
- Session – The tSS session that is used to transport the request from TCServer to T24's OFS system

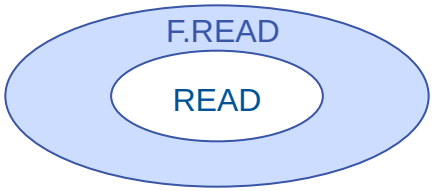
Session Management (Classic User Interface)



Session management - Browser



T24 API – F.READ



- F.READ – Read a record from a hashed file
- Will read a record only if a FILE.CONTROL record is present for the file being read
- Syntax

```
CALL F.READ(Filename,Key,Record,File path,Error variable)
```

- Example

```
CALL F.READ(FN.ACC,"11967",R.ACCOUNT,F.ACC,Y.ACC.ERR)
```

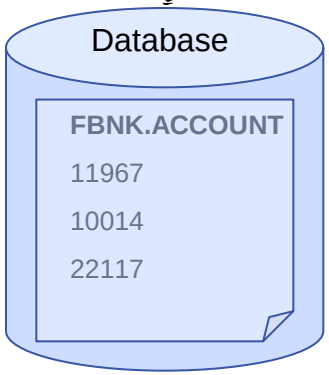
Internal working of F.READ

```
CALL F.READ(FN.ACC,"11967",R.ACCOUNT,F.ACC,Y.ACC.ERR)
```

File name : FBNK.ACCOUNT

Record Id : 11967

Record : xxxxxxxxx



Is the file pointed by FN.ACC open?

No → Perform OPF and open the file

Yes → Check if the record is in memory

No → Read record from database using jBC command **READ**
R.ACCOUNT = Actual account record

Load record on to memory

Take a look at this sample routine

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
FN.ACC='F.ACCOUNT'
F.ACC=''
Y.ACC.ID=11967
R.ACC=''
Y.ACC.ERR=''
CALL OPF(FN.ACC,F.ACC)
CALL
F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
RETURN
END
```

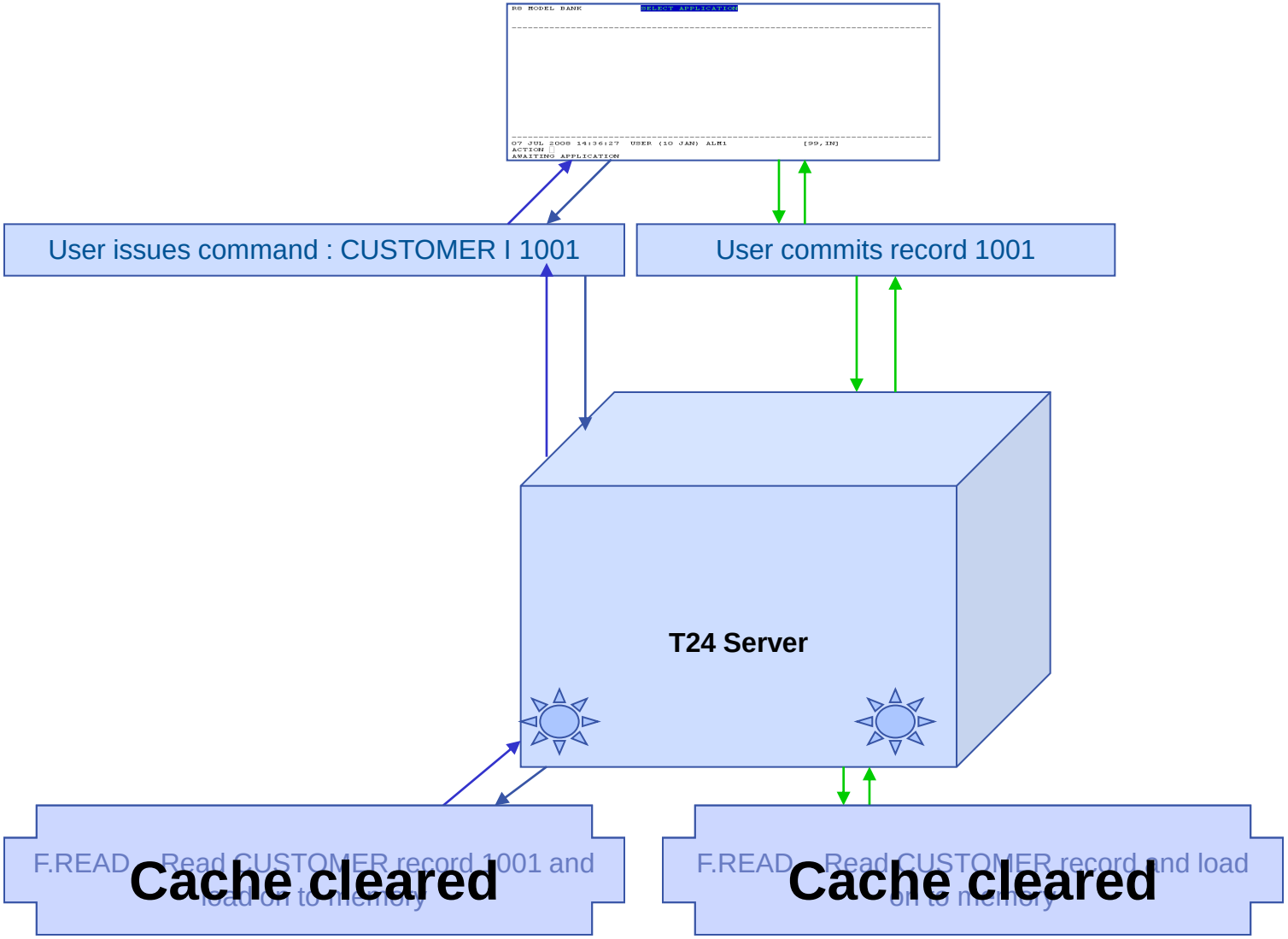
How does F.READ's caching work?

- Unlike OPF, F.READ does not maintain cache for a session
- It maintains cache for the life of a transaction only (transaction cache)

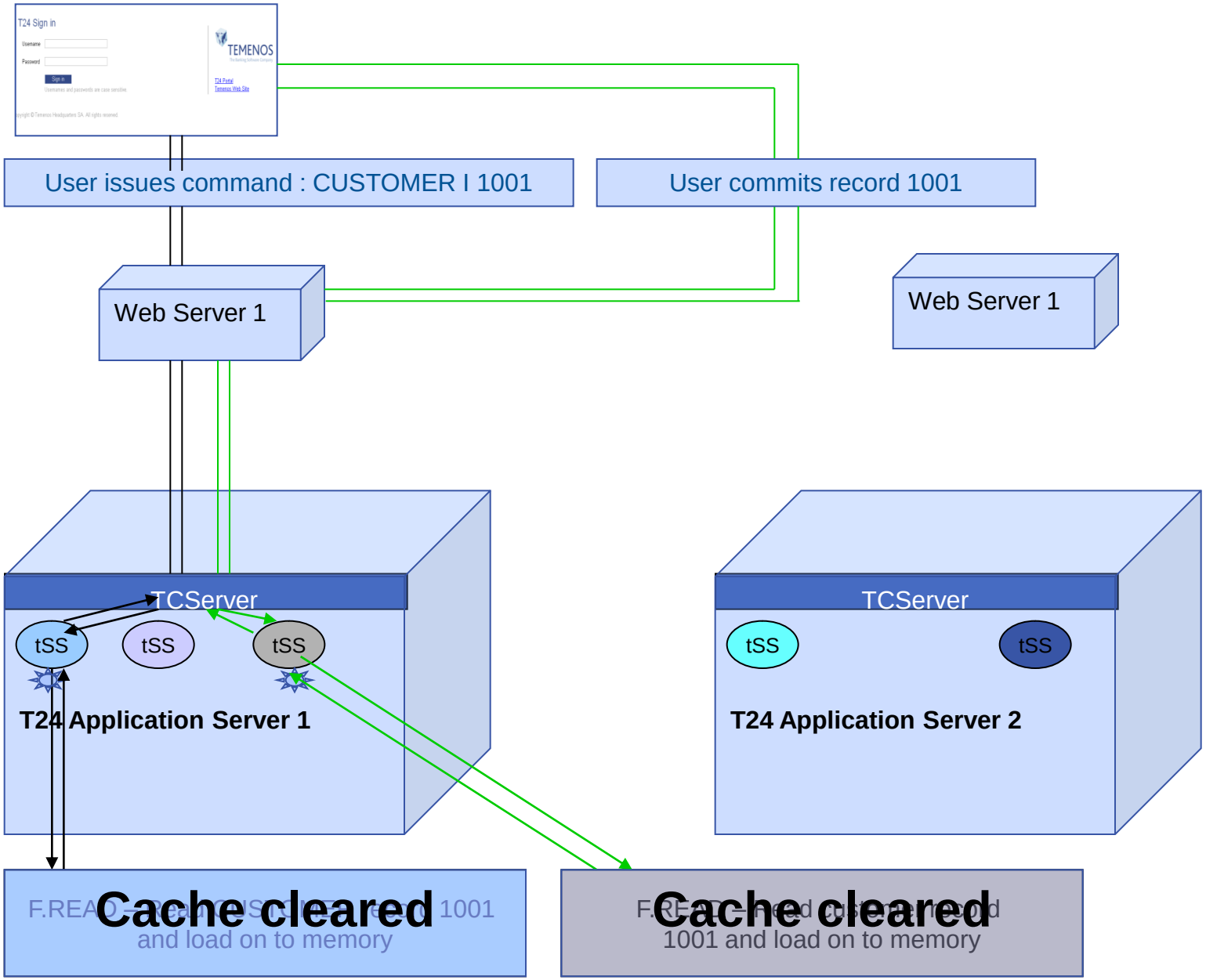
What is transaction cache?

- In simple terms, a transaction is any request that manipulates data in the database
- Example
  - Commit a record in an application
  - Authorize a record in an application
  - Reverse a record in an application

Transaction cache (Classic User Interface)



Transaction cache (Browser)

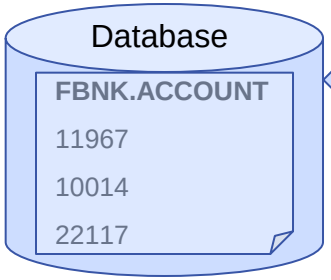


How does F.READ's caching work? - With an example

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
FN.ACC='F.ACCOUNT'
F.ACC=''
Y.ACC.ID=11967
R.ACC=''
Y.ACC.ERR=''
CALL OPF(FN.ACC,F.ACC)
CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
RETURN
END
```

How will F.READ work for TRG.TEST1?

```
Read 1 : CALL F.READ(FN.ACC,"11967",R.ACC,F.ACC,Y.ACC.ERR)
```



Is the file pointed by FN.ACC open?

No → Perform OPF and open the file

Yes → Check if the record is in memory

No

Read record from database using jBC command **READ**
R.ACC = Actual account record

Load record on to memory

How will F.READ work for TRG.TEST1?

```
Read 2 : CALL F.READ(FN.ACC,"11967",R.ACC,F.ACC,Y.ACC.ERR)
```

Is the file pointed by FN.ACC open?

Yes

Check if the record is in memory

Yes

Read record from memory
R.ACC = Actual account record

Note that the jBASE command READ has not be used and hence one IO (Input Output) to the database has been saved.

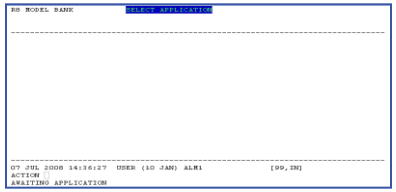
More on F.READ

- Would you ever want to read a record twice within the same routine?
  - Surely NO
  - Then, why do we need F.READ?

- Consider the following scenario

```
SUBROUTINE TRG.TEST1
$INSERT  I_COMMON
$INSERT  I_EQUATE
FN.ACC='F.ACCOUNT'
F.ACC=''
Y.ACC.ID=11967
R.ACC=''
Y.ACC.ERR=''
CALL OPF(FN.ACC,F.ACC)
CALL
F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
CALL CALLED.RTN1
---------------------------------
CALL CALLED.RTN2
---------------------------------
CALL CALLED.RTN23
RETURN
END
```

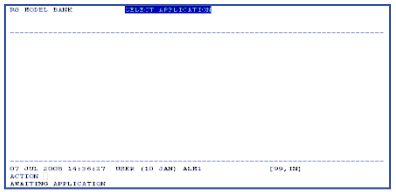
If CALLED.RTN1, CALLED.RTN2 and CALLED.RTN3 wish to access the account record 11967 to check certain values, will they be able to use R.ACC or would they have to read the record again using F.READ?



Session 1

- Time : 10.30 AM
- User 1 logs in
- Executes routine TRG.TEST2 in Session 1
- Will the record be fetched from the database or will it be fetched from memory?

Session 2



- Time : 10.35 AM
- User 2 logs in
- Executes routine TRG.TEST2 in Session 2
- Will the record be fetched from the database or will it be fetched from memory?

This must be easy!!!!

Golden rules about F.READ

- Do you remember READU? READU helps lock and read a record where as READ only reads a record
- Since F.READ uses READ internally, it does not hold a lock on the record that is read
- When to use F.READ
  - When you wish to query data in a record, use F.READ
  - Example : You wish to check the category of an account
- When not to use F.READ
  - Never read a record using F.READ if you wish to update data that has been read
  - Example : You wish to update the balance in an account. In this case do not read the account record using F.READ

Working of F.READ with an example

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT I_EQUATE
FN.ACC='F.ACCOUNT'
F.ACC=''
Y.ACC.ID=13935
R.ACC=''
Y.ACC.ERR=''
CALL OPF(FN.ACC,F.ACC)
CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
CRT "RECORD DETAILS"
CRT R.ACC
RETURN
END
```

```
0014      DEBUG
jBASE debugger->S
0015      CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)       ;*Read record
jBASE debugger->S
0016      CRT R.ACC
jBASE debugger->S
100334þ1001þBERNIE ECCLESTONEþþBERNIE ECCLESTONEþECCLESTCHFþTRþCHFþ1þþ60þþþþþþþ
þþ1þþþþþþþþþþþþþþþþþþþþþþþþþ20071231þþþþþþþþþþþþþþþþþþþþþþþþþþþNOþþ20060417þþ
þþþþþ1001þþþþþþþþCHFþþCHFþþþþLEGACYþþþþþþþþNOþþþþþþþþþþþþþþþþþþþþþþþþþþþþþ
þþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþ1þ53_INPUTTER___OFS_OFS.LOADþ0
712052135þ1_AUTHORISER_OFS_MB.OFS.AUTHþGB0010001þ1
0017      RETURN
jBASE debugger->V Y.ACC.ERR
  Y.ACC.ERR                    :
jBASE debugger->
```

How do you extract parts of a record returned by F.READ?

```
0014      DEBUG
jBASE debugger->S
0015      CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)       ;*Read record
jBASE debugger->S
0016      CRT R.ACC
jBASE debugger->S
100334þ1001þBERNIE ECCLESTONEþþBERNIE ECCLESTONEþECCLESTCHFþTRþCHFþ1þþ60þþþþþþþ
þþ1þþþþþþþþþþþþþþþþþþþþþþþþ20071231þþþþþþþþþþþþþþþþþþþþþþþþþþþþNOþþ20060417þþ
þþþþþ1001þþþþþþþþCHFþþCHFþþþþLEGACYþþþþþþþþNOþþþþþþþþþþþþþþþþþþþþþþþþþþþþþ
þþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþþ
712052135þ1_AUTHORISER_OFS_MB.OFS.AUTHþGB0010001þ1
0017      RETURN
jBASE debugger->V Y.ACC.ERR
  Y.ACC.ERR                    :
jBASE debugger->
```

- F.READ always returns the record in a dynamic array
- Extract parts of a dynamic array using the following convention

Y.CURRENCY = R.ACC<8>

(or)

Y.CURRENCY = R.ACC<Name of the field CURRENCY in the ACCOUNT file>

Insert files

- There are some insert files that are common to entire T24 like
  - I_COMMON
  - I_EQUATE
- There are application specific insert files – one for each application
  - I_F.ACCOUNT
  - I_F.CUSTOMER
- All T24 core insert files will be available under GLOBUS.BP

- Non application specific insert file naming convention

  I_<Insert file name>

- Application specific insert file naming convention

  I_F.<ApplicationName>

I_F.ACCOUNT

```
* File Layout for ACCOUNT Created 25 FEB 07 at 07:47AM by kr05a
*       PREFIX[AC.]       SUFFIX[]
         EQU AC.CUSTOMER TO 1,              AC.CATEGORY TO 2,
     AC.ACCOUNT.TITLE.1 TO 3,       AC.ACCOUNT.TITLE.2 TO 4,
          AC.SHORT.TITLE TO 5,              AC.MNEMONIC TO 6,
        AC.POSITION.TYPE TO 7,              AC.CURRENCY TO 8,
     AC.CURRENCY.MARKET TO 9,            AC.LIMIT.REF TO 10,
    AC.ACCOUNT.OFFICER TO 11,        AC.OTHER.OFFICER TO 12,
   AC.POSTING.RESTRICT TO 13,       AC.RECONCILE.ACCT TO 14,
 AC.INTEREST.LIQU.ACCT TO 15,   AC.INTEREST.COMP.ACCT TO 16,
     AC.INT.NO.BOOKING TO 17,         AC.REFERAL.CODE TO 18,
   AC.WAIVE.LEDGER.FEE TO 19,            AC.LOCAL.REF TO 20,
    AC.CONDITION.GROUP TO 21,       AC.INACTIV.MARKER TO 22,
    AC.OPEN.ACTUAL.BAL TO 23,     AC.OPEN.CLEARED.BAL TO 24,
  AC.ONLINE.ACTUAL.BAL TO 25,   AC.ONLINE.CLEARED.BAL TO 26,
    AC.WORKING.BALANCE TO 27,    AC.DATE.LAST.CR.CUST TO 28,
  AC.AMNT.LAST.CR.CUST TO 29,    AC.TRAN.LAST.CR.CUST TO 30,
  AC.DATE.LAST.CR.AUTO TO 31,    AC.AMNT.LAST.CR.AUTO TO 32,
  AC.TRAN.LAST.CR.AUTO TO 33,    AC.DATE.LAST.CR.BANK TO 34,
  AC.AMNT.LAST.CR.BANK TO 35,    AC.TRAN.LAST.CR.BANK TO 36,
  AC.DATE.LAST.DR.CUST TO 37,    AC.AMNT.LAST.DR.CUST TO 38,
  AC.TRAN.LAST.DR.CUST TO 39,    AC.DATE.LAST.DR.AUTO TO 40,
```

Part of the I_F.ACCOUNT file that links field names to field positions

```
0001 * File Layout for CUSTOMER Created 15 OCT 07 at 04:14PM by tpOtba
0002 *      PREFIX[EB.CUS.]      SUFFIX[]
0003 EQU EB.CUS.MNEMONIC TO 1,
0004     EB.CUS.SHORT.NAME TO 2,
0005     EB.CUS.NAME.1 TO 3,
0006     EB.CUS.NAME.2 TO 4,
0007     EB.CUS.STREET TO 5,
0008     EB.CUS.ADDRESS TO 6,
0009     EB.CUS.TOWN.COUNTRY TO 7,
0010     EB.CUS.POST.CODE TO 8,
0011     EB.CUS.COUNTRY TO 9,
0012     EB.CUS.RELATION.CODE TO 10,
0013     EB.CUS.REL.CUSTOMER TO 11,
0014     EB.CUS.REVERS.REL.CODE TO 12,
0015     EB.CUS.REL.DELIV.OPT TO 13,
0016     EB.CUS.ROLE TO 14,
```

Part of the I_F.CUSTOMER file that links field names to field positions

Try this now

Open and view the insert file for application FUNDS.TRANSFER using Eclipse

Note down the prefix used

Note down the full name of the field (Along with prefix and suffix if any) for field DEBIT.CURRENCY

How will you extract only CURRENCY and CATEGORY from R.ACC?

R.ACC<AC.CURRENCY>

R.ACC<AC.CATEGORY>

Take a look at the entire subroutine

```
0001      SUBROUTINE TRG.TEST2
0002      $INSERT I_COMMON
0003      $INSERT I_EQUATE
0004      $INSERT I_F.ACCOUNT
0005      GOSUB INIT
0006      GOSUB OPENFILES
0007      GOSUB PROCESS
0008      RETURN
0009 INIT:
0010      DEBUG
0011      FN.ACC = 'F.ACCOUNT'
0012      F.ACC = ''
0013      Y.ACC.ID = 11967
0014      R.ACC = ''
0015      Y.ACC.ERR = ''
0016      RETURN
0017 OPENFILES:
0018      CALL OPF(FN.ACC,F.ACC)     ;* Open File
0019      RETURN
0020 PROCESS:
0021      CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)    ;* Read Record
0022      CRT "Currency : ": R.ACC<AC.CURRENCY>
0023      CRT "Category : ": R.ACC<AC.CATEGORY>
0024      RETURN
0025 END
```

Compiling and cataloguing routines

COMPILE

BASIC TRG.BP TRG.TEST2

Check for errors

On error
Display error and exit

No error
Produce object code
$TRG.TEST2

TRG.BP

$TRG.TEST2

CATALOG

CATALOG TRG.BP TRG.TEST2

Check variable JBCDEV_LIB

```
JBCDEV_LIB=%HOME%\trglib
```

Place the object code under any one library file (Under the path pointed by JBCDEV_LIB) depending on which has space

Compiling And Cataloguing Routines

- What happened after compilation?

```
jsh testbase ~ -->jshow -c TRG.RTN1
Subroutine:          C:\alm\LocalHosts\TestBase\testbase\TestBase.run\trglib\lib0.dll
                     jBC TRG.RTN1 version 5.0 Sun Feb 18 00:48:53 2007
                     jBC TRG.RTN1 source file TRG.BP
jsh testbase ~ -->jshow -a lib0.dll
Subroutine object: C:\alm\LocalHosts\TestBase\testbase\TestBase.run\trglib\lib0.dll, contains 5 subroutines
        JBC_TRG_2EDAS_2ERTN
        JBC_STEST_2ETRG_2ESEAT_2ERTN
        JBC_TRG_2ESEAT_2ERTN
        JBC_TRG_2ERTN1
        JBC_TEST_2ESUB1
```

- During the cataloguing process, jBASE might shift object codes from one library file to another in order to make best use of the space available in the library files
- Which means the object code of TRG.RTN1 will not remain inside lib0.dll for a lifetime

Executing routines

Login into T24

Make an entry in the PGM.FILE with TYPE M

At the command line
type the routine name (TRG.RTN1)

Check where the object code is?

JBCOBJECTLIST=$HOME/T24lib;$HOME/lib;$HOME/trglib

Execute the routine

Debugging …….

- Insert the ‘DEBUG’ statement anywhere in the routine to see execution line by line

```
0001      SUBROUTINE TRG.TEST2
0002      $INSERT I_COMMON
0003      $INSERT I_EQUATE
0004      $INSERT I_F.ACCOUNT
0005      GOSUB INIT
0006      GOSUB OPENFILES
0007      GOSUB PROCESS
0008      RETURN
0009 INIT:
0010      DEBUG
0011      FN.ACC = 'F.ACCOUNT'
0012      F.ACC = ''
0013      Y.ACC.ID = 11967
0014      R.ACC = ''
0015      Y.ACC.ERR = ''
0016      RETURN
0017 OPENFILES:
0018      CALL OPF(FN.ACC,F.ACC)     ;* Open File
0019      RETURN
0020 PROCESS:
0021      CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)    ;* Read Record
0022      CRT "Currency : ": R.ACC<AC.CURRENCY>
0023      CRT "Category : ": R.ACC<AC.CATEGORY>
0024      RETURN
0025 END
```

Compiling with the keyword DEBUG

- Compile the subroutine again
- R7 and later changes
  - Routines with DEBUG statements are not compiled
  - To force compilation -D switch required.
  - EB.COMPILE TRG.BP TRG.TEST2 -D

Debugging……

```
Source changed to TRG.BP/TRG.TEST2
0011 DEBUG
jBASE debugger->S
0012          FN.ACC = 'F.ACCOUNT'
jBASE debugger->S
0013          F.ACC = ''
jBASE debugger->S
0014          Y.ACC.ID = 11967
jBASE debugger->S
0015          R.ACC = ''
jBASE debugger->S
```

Line seen on typing S at the debugger prompt is the line that is waiting to be executed

Debugging…….

```
----------------------------------------
----------------------------------------
0020 PROCESS:
jBASE debugger->V FN.ACC
  FN.ACC                        : FBNK.ACCOUNT
jBASE debugger->V F.ACC
  F.ACC                         : File '../mbdemo.data/ac/FBNK.ACCOUNT'
```

To view parts of a dynamic arrays at the debugger prompt, type

V DynamicArray<Position>

V ARR1<2>

Workshop 1

- Create a subroutine named XXX.RTN1 which will display the mnemonic and the sector of any customer in your database
- Routine to be placed under XXX.BP where XXX is your initial
- Routine to be catalogued to xxxlib where xxx is your initial

Task

- Insert a value “From Training” in the field ACCOUNT.TITLE.2 for account 11967

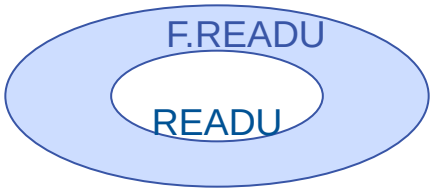
Algorithm

Task	T24 API to be used
Open ACCOUNT file	OPF
Read record with key 11967	F.READ
Place “Valued customer” in the TEXT field	dynamicarray<position> = value
Write the record with key 11967	F.WRITE 

Is the algorithm correct?

Corrected algorithm

Task	T24 API to be used
Open ACCOUNT file	OPF
Read and lock record with key 11967	F.READU
Place “Valued customer” in the field TEXT	dynamicarray<position> = value
Write the record with key 11967 to the database	F.WRITE



- F.READU – Read and lock a record from a hashed file
- Syntax

```
CALL F.READU(Filename,Key,Record,File path,Errorvariable,Option)
```

- Example

```
CALL F.READU(FN.ACC,"11967",R.ACC,F.ACC,Y.ACC.ERR,'')
```

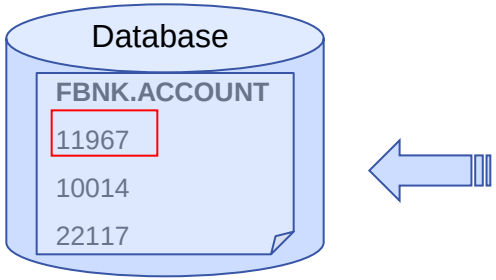
If the record is locked, wait until the lock is released and then lock

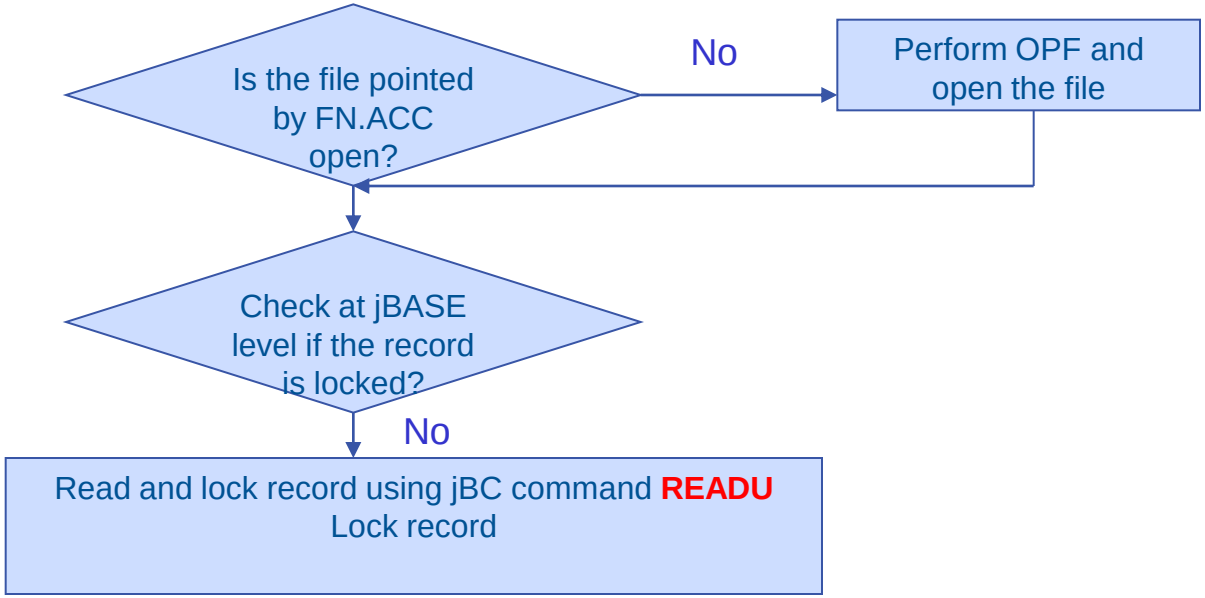
How will the subroutine look with F.READU incorporated?

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
FN.ACC='F.ACCOUNT'
F.ACC=''
Y.ACC.ID=13935
R.ACC=''
Y.ACC.ERR=''
RETURN
OPENFILES:
CALL OPF(FN.ACC,F.ACC)
RETURN
PROCESS:
CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'')
CRT "RECORD DETAILS"
CRT R.ACC
RETURN
END
```

Internal working of F.READU

```
CALL F.READU(FN.ACC,"11967",R.ACC,F.ACC,Y.ACC.ERR,'')
```





F.READ and F.READU - Think

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
FN.ACC='F.ACCOUNT';F.ACC='';Y.ACC.ID=13935;R.ACC=''
Y.ACC.ERR=''
RETURN
OPENFILES:
CALL OPF(FN.ACC,F.ACC)
RETURN
PROCESS:
CALL TRG.TEST2
CALL
   F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'')
CRT "RECORD DETAILS"
CRT R.ACC
RETURN
END
```

```
SUBROUTINE TRG.TEST1
------------------------
------------------------
Y.ACC.ID=13936
CALL
F.READU(FN.ACC,Y.ACC.ID,
R.ACC,F.ACC,Y.ACC.ERR,''
)
RETURN
END
```

Will F.READU read the record 11967 from cache or will it read from the disk?

F.READ and F.READU

```
 *Routine to emphasise the working of F.READ
 SUBROUTINE TRG.TEST2
-----------
-----------
CALL TRG.TEST3
CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'') ; *Read and lock record
RETURN
END
```

```
CALL F.READ(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR)
```

Read record from database and load on to memory

File name and ID : FBNK.ACCOUNT 11197 FBNK.ACCOUNT 11967

Record : xxxxx yyyyy

Type : **L**

When does the lock get released?

You will learn this after you learn about F.WRITE

T24 API – F.WRITE

- F.WRITE – Write a record to the database
- Syntax

```
CALL F.WRITE(Filename,Key,Record)
```

- Example

```
CALL F.WRITE(FN.ACC,"11967",R.ACCOUNT)
```

How will the subroutine look with F.WRITE incorporated?

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT  I_EQUATE
$INSERT I_F.ACCOUNT
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
FN.ACC='F.ACCOUNT';F.ACC='';Y.ACC.ID=13935
R.ACC='';Y.ACC.ERR=''
RETURN
OPENFILES:
CALL OPF(FN.ACC,F.ACC)
RETURN
PROCESS:
CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'')
CRT "RECORD DETAILS BEFORE WRITE"
CRT R.ACC
R.ACC<AC.ACCOUNT.TITLE.2>="FROM TRAINING";
CALL F.WRITE(FN.ACC,Y.ACC.ID,R.ACC)
CRT "RECORD DETAILS AFTER WRITE"
CRT R.ACC
RETURN
END
```

Name of the field picked up from I_F.ACCOUNT

Working of F.WRITE

```
*Routine to emphasise the working of F.READU and F.WRITE
SUBROUTINE TRG.TEST2
---------------
--------------
        CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'') ; *Read and lock record
        R.ACC<AC.ACCOUNT.TITLE.2> = "From Training" ; *Set value
        CALL F.WRITE(FN.ACC,Y.ACC.ID,R.ACC) ; *Write record to file
RETURN
END
```

FWT : FBNK.ACCOUNT,11197	FBNK.ACCOUNT 11967	FBNK.ACCOUNT 11967	FBNK.ACCOUNT 11967
FWC : xxxxx	yyyyy	xxxxxxxxxxx	xxxxxxxxxxx
FWF:	**L**	**W**	**WL**

First READ	READU without WRITE	READ and WRITE	READU and WRITE

↑

DO NOT USE

Why doesn't F.WRITE write directly to the disk? Why does it cache?

- You know
  - Cache is maintained for every transaction
  - A request to T24 can be called a transaction
  - In this case TRG.TEST2 is a transaction
- Assume a scenario like the one below

```
*Routine to emphasise the working of F.READU and F.WRITE
SUBROUTINE TRG.TEST2
---------------
--------------
         CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'') ; *Read and lock record
     R.ACC<AC.ACCOUNT.TITLE.2> = "From Training" ; *Set value
          CALL F.WRITE(FN.ACC,Y.ACC.ID,R.ACC) ; *Write record to file
     CALL F.READU(……….)
     CALL F.WRITE(……)
RETURN
END
```

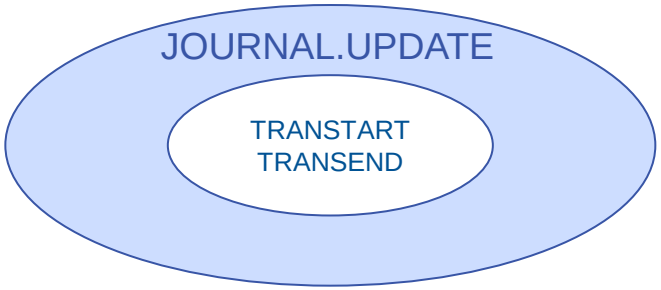
  - Assume the first F.WRITE goes through without any errors and data is updated in the database
  - Assume the second F.WRITE fails (May be the file is corrupted or there are insufficient permissions on the file)

    Would it be fine if one of the F.WRITEs fail and the other one successfully updates the database?

Why doesn't F.WRITE write directly to the disk? Why does it cache?

- To ensure that all data in a transaction is written to disk or none is written to disk, F.WRITEs cache data
- Who will then write data to the database
  - Answer : JOURNAL.UPDATE
- Transaction management when bulk messages (BROWSER)
  - Requests from browser are considered as BULK
  - Data is flushed to the disk when bulk transaction is complete
  - SYSTEM(47) determines whether transaction management is active or not.

T24 API - JOURNAL.UPDATE



- T24 API that controls transaction management

```
SUBROUTINE JOURNAL.UPDATE
-------
-------
CALL EB.TRANS("START",'')

        Takes all data with 'W' marker (Recollect FWT, FWC and FWF arrays) and writes to a buffer


CALL EB.TRANS("END",'')


RETURN
END
```

TRANSTART

TRANSEND

- EB.TRANS("ABORT",'') is also called by JOURNAL.UPDATE when the transaction needs to be aborted (One of the statements within the block fail and hence the need to abort)
- This internally calls the jBASE TRANSABORT command to abort or rollback any data changes that has happened within the transaction

How will the subroutine look with JOURNAL.UPDATE incorporated?

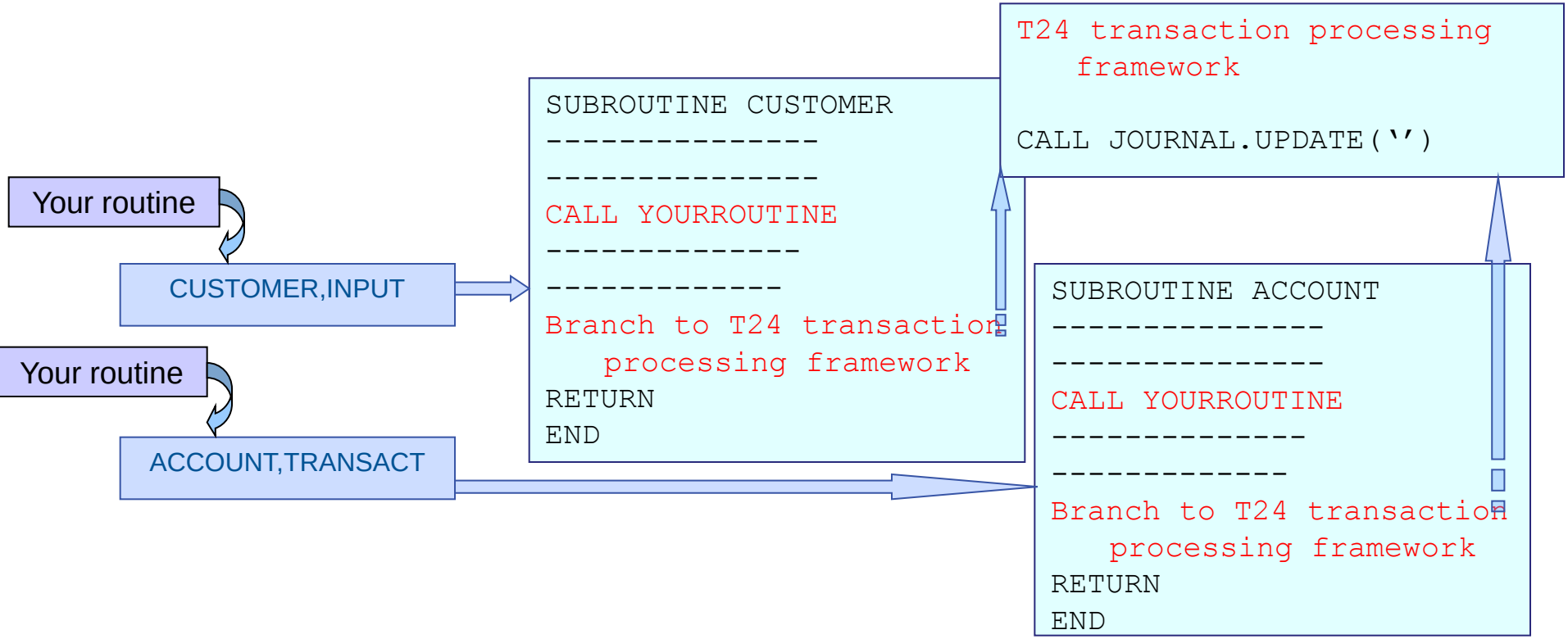
```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.ACCOUNT
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
FN.ACC='F.ACCOUNT';F.ACC='';Y.ACC.ID=13935
R.ACC='';Y.ACC.ERR=''
RETURN
OPENFILES:
CALL OPF(FN.ACC,F.ACC)
RETURN
PROCESS:
CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'')
CRT "RECORD DETAILS BEFORE WRITE"
CRT R.ACC
R.ACC<AC.ACCOUNT.TITLE.2>="FROM TRAINING";
CALL F.WRITE(FN.ACC,Y.ACC.ID,R.ACC)
CALL JOURNAL.UPDATE('Y.ACC.ID')
CRT "RECORD DETAILS AFTER WRITE"
CRT R.ACC
RETURN
END
```

More on JOURNAL.UPDATE

- Do I have to call JOURNAL.UPDATE for every routine that I write?

NO

- The routines that you are writing now are mainline routines
  - Standalone routines
  - Executed from the T24 command line
- Normally routines are written and attached to various applications in T24. T24 applications will call JOURNAL.UPDATE



Back to the question – When do locks get released?

- A lock on a record is released when
  - An F.WRITE is executed on the record that has been locked
  - If within a transaction, then, F.WRITE will not release the lock. Once the transaction is complete, the lock gets released.

    When TRANSEND is called by JOURNAL.UPDATE, all locks get released
  - Internally calls the jBC command RELEASE
  - RELEASE <filename> releases all locks on the given file held by current session

Use this with caution

- Used to release locks. Locks are released at the end of the transaction.
- Use it only if you have locked a record using F.READU but haven't written the record back using F.WRITE

```
*Routine to emphasise the working of F.RELEASE
SUBROUTINE TRG.TEST2
---------------
         CALL F.READU(FN.ACC,11967,R.ACC,F.ACC,Y.ACC.ERR,'') ; *Read and lock record
          CALL F.READU(FN.ACC,11956,R.ACC,F.ACC,Y.ACC.ERR,'') ; *Read and lock record
    CALL F.WRITE(FN.ACC,11956,R.ACC) ; *Write record to file
    CALL F.RELEASE(FN.ACC,11967,F.ACC)
--------------
RETURN
END
```

- If no record key is specified – All locks on the file name specified are released
- If no file name is specified, all locks are released (Online only)

Keep in mind

Sensitive routines listed below should not be used, as only the T24 transaction processing framework is allowed to call it

- JOURNAL.UPDATE
- F.RELEASE
- EB.TRANS
- CACHE.OFF – variable to denote whether caching is on or off

Workshop 2

- Write a routine to update the field TEXT in the CUSTOMER application with a value “This is from training”. Use any customer number of your choice.
- Note : TEXT is a multi value field. Write code in such a way that data is always appended to the field and not overwritten

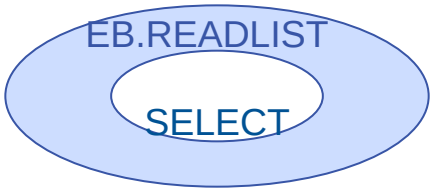
Task

- Write a routine to display the ID, category and currency of all accounts

Algorithm

Task	T24 API to be used
Select all account IDs	EB.READLIST NEW!
Start loop	jBC command - LOOP
Read each account record	F.READ
Extract category and currency value	Variable=dynamicarray<position>
Display ID, category and currency	CRT variable
Loop back	jBC command - REPEAT

T24 API – EB.READLIST



- EB.READLIST – Execute a SELECT statement
- Syntax

```
CALL
EB.READLIST(Selectcommand,Selectedlist,'',NoOfRecordsSelected,
ReturnCode)
```

- Example

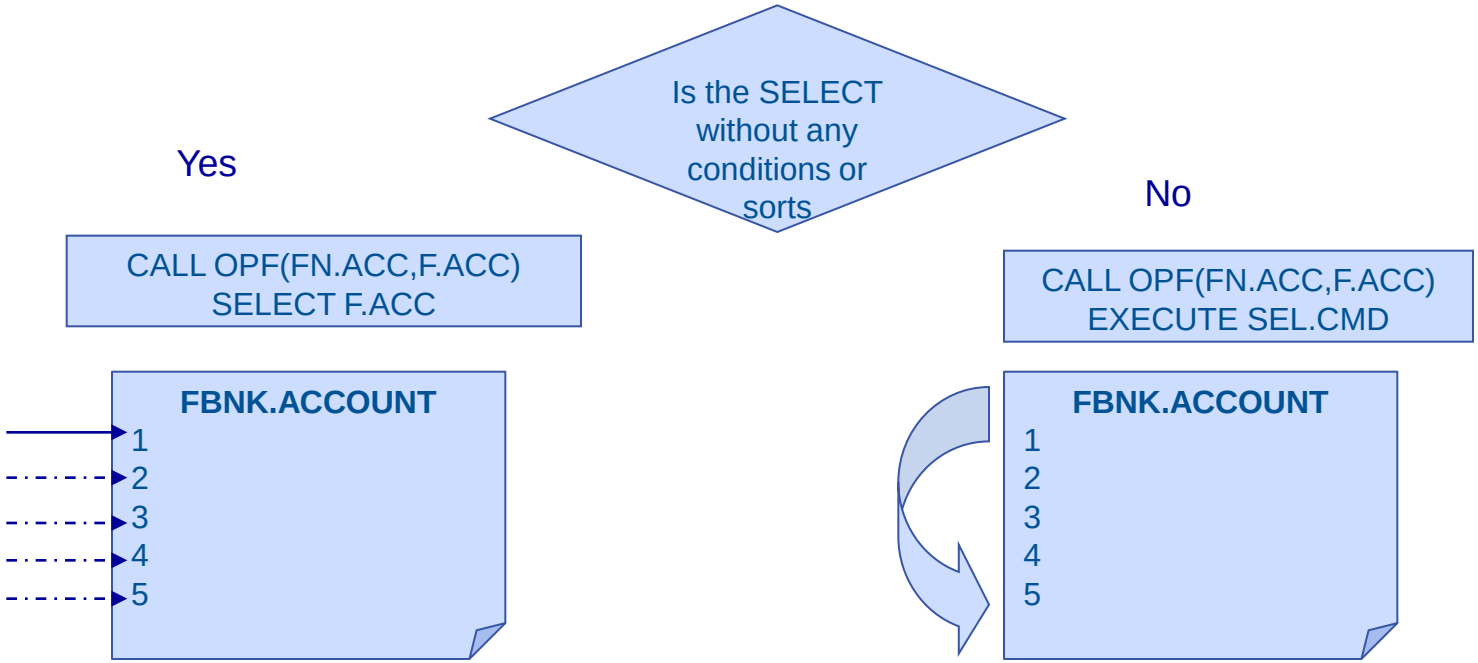
```
SEL.CMD = "SELECT ":FN.ACC
CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
```

Note the space

You may use SSELECT instead of SELECT if you want data in sorted order

Internal working of EB.READLIST

```
SEL.CMD = "SELECT ":FN.ACC
CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
```



Subroutine with EB.READLIST incorporated

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.ACCOUNT
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
FN.ACC='F.ACCOUNT`;F.ACC='`;Y.ACC.ID=13935;R.ACC='`;Y.ACC.ERR=''
RETURN
OPENFILES:
CALL OPF(FN.ACC,F.ACC)
RETURN
PROCESS:
SEL.CMD="SELECT ":FN.ACC
CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
LOOP
REMOVE Y.ACC.ID FROM SEL.LIST SETTING POS
WHILE Y.ACC.ID:POS
CALL F.READU(FN.ACC,Y.ACC.ID,R.ACC,F.ACC,Y.ACC.ERR,'')
CRT"ID-":Y.ACC.ID :"CATEGORY-":R.ACC<AC.CATEGORY>:"CURRENCY":R.ACC<AC.CURRENCY>
REPEAT
RETURN
END
```

Workshop 3

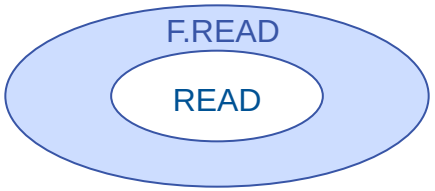
- Display the mnemonic and nationality of all customers

Welcome to CACHE.READ

- You know any record read using F.READ is available only as long as the transaction lasts
- What if you wish to read and cache records of some static or parameter applications such as SECTOR, CATEGORY etc?
- What if you want this cache last for a specific period of time instead of vanishing off after a transaction completes

- Answer : Use CACHE.READ

T24 API – CACHE.READ



- CACHE.READ – Read a record from a hashed file
- Syntax

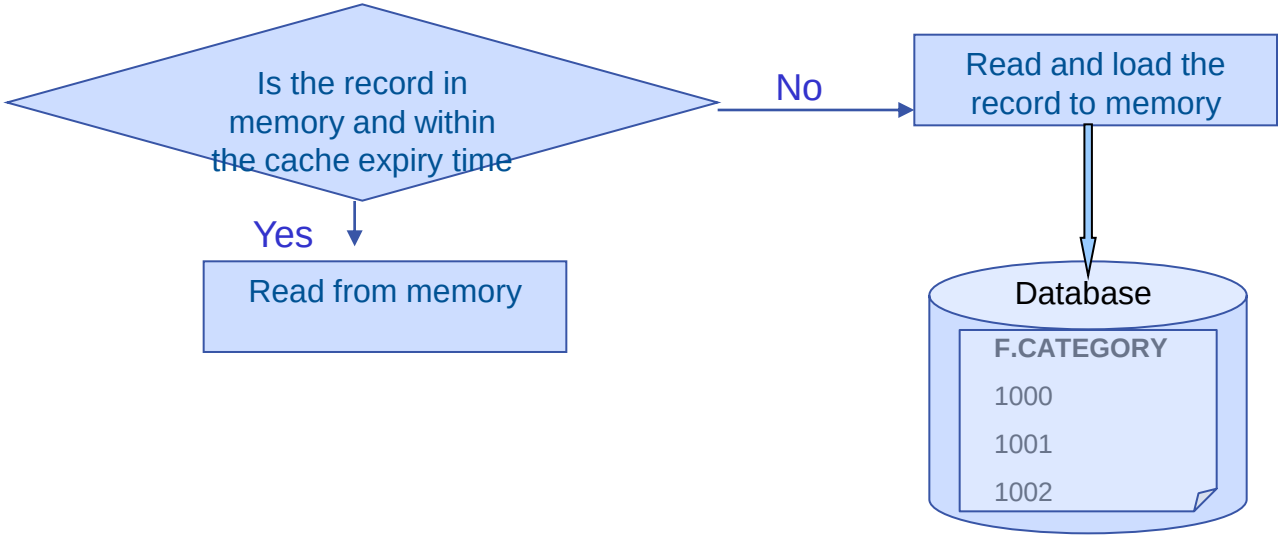
```
CALL CACHE.READ(FileNameWithoutMne,Key,Record,Error variable)
```

- Example

```
CALL CACHE.READ(F.CATEGORY,"1000",R.CATEGORY,Y.CAT.ERR)
```

Internal working of CACHE.READ

```
CALL CACHE.READ(F.CATEGORY,"1000",R.CATEGORY,Y.CAT.ERR)
```



Set the CACHE.EXPIRY time in SPF. Field to be used : CACHE.EXPIRY.

Default value for this field is 60 (Seconds)

Take a look at this sample routine

```
SUBROUTINE TRG.TEST1
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.SECTOR
FN.SEC='F.SECTOR'
F.SEC=''
Y.SEC.ID='1001'
R.SEC1=''
R.SEC2=''
Y.SEC.ERR=''
CALL OPF(FN.SEC,F.SEC)
CALL CACHE.READ(FN.SEC,'1001',R.SEC2,Y.SEC.ERR)
CRT "CACHE.READ OUTPUT"
CRT R.SEC2
RETURN
END
```

Points to remember about CACHE.READ

- Use this routine to read data from applications that contain frequently used but static data
- Never use this to read financial data (Eg: ACCOUNT)

- Time : 10.30 AM
  - User 1 logs in
  - Executes routine TRG.TEST2 in Session 1
  - Will the record be fetched from the database or will it be fetched from memory?

- Time : 10.32 AM
  - User 1 executes the routine TRG.TEST2 in Session 1
  - Will the record be fetched from the database or will it be fetched from memory?

Some more APIs

- EB.READ.PARAMETER
  - To read parameter record of any module
  - Returns both dynamic and dimension array
  - Can indicate whether to read the record with a lock or not

- F.DELETE
  - Deletes the record from cache
  - During cob if write cache is not enabled delete the record immediately

Some more APIs

- LOG.WRITE
  - Routine to write all logs in T24
  - Can be used to write to disk directly if transaction management is not set
  - Takes in four parameters

Discussion

- Form 5 groups
- Topics for each group
  - Group 1 : OPF
  - Group 2 : F.READ and CACHE.READ
  - Group 3 : F.READU
  - Group 4 : F.WRITE
  - Group 5 : JOURNAL.UPDATE and F.RELEASE

- Discuss and understand the working of the T24 API given to you (10 minutes)
- Form questions to ask the other groups – 5 questions (10 minutes)
- Groups ask questions to each other
- Note the group that is the highest scorer

Authoring Contributions:

Alagammai(First Edition, 2001) – Temenos India Private Ltd.

Alagammai(Second Edition, 2004) – Temenos India Private Ltd.

Alagammai(ThirdEdition, 2008) – Temenos India Private Ltd.

Kalaiselvi (Fourth Edition, 2010) - Temenos India Private Ltd

Edited for XIB by:

Gerard Thomas (Mar 2011)

Thankful Acknowledgements:

Temenos Corporate Training Team