

*By V.Kazimirchik*

# Contents

1	Preface	5
2	Model Bank – installation	6
3	Does it work straight away?	8
4	What’s next?	17
5	Shortcuts, applications basics	18
6	Application data, LIVE and NAU files, record status, audit trail, LIST basics, INT applications	20
7	FIN and CUS applications, history file, “F” VOC entries, COMPANY, EVAL in LIST	25
8	SELECT and SELECT lists, COUNT, “SAVING” and “UNIQUE” in SELECT	32
9	More navigation in Classic	36
10	Introduction into programs and subroutines, conclusion for applications	38
11	Introduction into OFS, more about functions, setup OFS.SOURCE, tSS, simple enquiry output	40

12 OFS – inputting an application record: VERSION creation, “VALIDATE” option, couple of tests, STANDARD.SELECTION check	46
13 Writing a simple T24 subroutine	49
14 Getting application information from a routine	55
15 OFS – application record creation – continued, overrides, fields GTS.CONTROL and NO.OF.AUTH	56
16 VERSION routines – AUT.NEW.CONTENT, R.NEW, application insert file	61
17 OFS.REQUEST.DETAIL	67
18 Manual transaction input in comparison with OFS, GTSACTIVE variable	69
19 Browser client – jboss, jBASE agent, logging in	73
20 Transaction input under Browser, debugging	78
21 TODAY variable, date format in T24, edit mode in Classic, API for dates manipulation	82
22 Global variables again – their lifetime, writing a PROGRAM, CRT	85

23 CHECK.REC.RTN – error raising, other VERSION routines – notes	87
24 Programming language overview, writing a simple game	89
25 Local applications, code rating, enrichment, AUTO.ID.START, SEARCH, jBASE file types, jstat	100
26 File I/O	111
27 Changing structure of a local application – amending field type	119
28 Records locking	122
29 Programming for enquiries	123
30 Performance, code analysis, local fields, DYNAMIC.TEXT	128
31 I-descriptors	143
32 COB programming	147
33 Linux	167
34 Java and jRemote	171
35 Conclusions	181

# 1 Preface

**S**uddenly it came to my mind that it's already 10 years that I work with Temenos product T24. Of course it wasn't T24 all those years... It was Globus previously and I've started with G10 release of Globus. Now it's R10 release of T24.

So I thought it's a good idea to share some things that came to my attention during this time (because if I don't I'll surely forget it all myself). Being a technical person, I address this book mainly to techies, though business guys might also find something for them.

The phrase "This is how Globus works" is well-known to anyone who worked with Globus and I decided that it's a very appropriate title.

It will mostly be a case study (all cases real, some of them too real) – far from what regular training provides. Probably it can be seen as an addition to regular training, sometimes expanding beyond "pure" T24 – sort of a collection of tips, tricks, hand-ons, how-tos etc.

I've noticed that for a beginner it's not easy to understand what belongs where – like "where user access is supported – at DB or application level?" etc. And for a person who knows jBASE or Universe well it's a bit embarrassing to learn that some thing absolutely valid in "pure" jBASE doesn't work in T24 environment (like adding something to dictionary manually – it lives only until *STANDARD.SELECTION* record is rebuilt next time). So I'll try to close this gap (and hopefully some others).

*Note: all views, opinions etc are mine, not of my employer. Solutions and code lines (mostly) are mine as well, unless stated otherwise. You also agree that you use the samples provided with full understanding what you're doing – i.e. at your own risk. T24 is unpredictable in its changes and what worked perfectly well in one release will not work in another – or, worse, might harm your system.*

*It's a good idea to try all this on a virtual machine first...*

*Many things described here you can find in some manuals, many are not described anywhere to my knowledge. Hope you'll find something useful. You'll see me digressing a lot but it's all intentional.*

*To my experienced colleagues: don't consider it as something like "T24 for dummies". Feel free to skip what you already know (though you might find something new even there). Be sure that you'll find many places that will be a surprise for you.*

*Enjoy!*

All trademarks are the property of their respective owners.

Ok, where do we start?

## 2 Model Bank – installation

**Y**ou need to have some T24 environment. Preferrably Model Bank. The samples will be mostly based on R10, however older releases will do as well.

Firstly let's put things in their place. Create a directory at your C: drive (we'll cover Linux as well but for time being it's Windows). At my laptop this directory is called `temenos`. Then – create R10 subdirectory in it (we might need other releases as well, and, for example, TAFC for R10 is different from TAFC for R09). So it's better to have all the things in one “package” that you'll be able to move to another laptop or PC just copying it, if needs be.

Depending on what you've got from distribution you'll only need to extract the contents to appropriate subdirectories of `c:\temenos` (which is IMHO preferable) or, maybe, to proceed with TAFC installation first. In both cases you end up with the following components:

a) TAFC – new name for jBASE. I still prefer to call it “jBASE”. TAFC

is very similar to jBASE 5.x. (in `c:\temenos\R10\T AFC`) – note that once you’ve installed T AFC you can move it to another location, but keep in mind `telnetd` issue:

You’ll need `telnetd.exe` and a terminal emulation program (like Reflection or PuTTY) to run so-called “Clobus Classic” which in my opinion is a must for a T24 techie. Of course you can run it from Windows shell but in most cases the output will be ugly (no terminal command sequences supported, at least under XP) and terminal emulators also allow to map certain keys to help you to navigate in Globus Classic.

So where to find `telnetd.exe`? In some T AFC installations it presents and you can activate it by the following commands (firstly deactivate an old one in case you need to move the whole T AFC directory or remove older T AFC altogether):

```
cd c:\temenos\R10\T AFC\bin
telnetd stop
telnetd remove
telnetd install
telnetd start
```

If you don’t have it in `bin` subdirectory – find an older jBASE 4.x or 5.x, install it and use `telnetd.exe` located there. Windows telnet service lacks the feature of auto-starting `remote.cmd` file (probably it can be catered to do that but I never needed it).

An alternate way is to install free ssh server for Windows and use ssh client like PuTTY.

b) Model Bank environment (`c:\temenos\R10\mb10`). There you extract contents of `bnk` directory that you find in a distribution package. I usually prefer not to retain `bnk` directory itself so we end up with folders like `c:\temenos\R10\mb10\bnk.data`, `c:\temenos\R10\mb10\bnk.run` etc.

c) C compiler. We’ll definitely need it in many cases. MS Visual Studio .Net 2003 will be enough. To avoid a path with space in it (like `c:\Program`

Files etc), it makes sense if you copy its `bin`, `lib` and `include` directories into `c:\temenos\compile`. We'll see later how to set up things to be able to compile your source code.

d) Java SDK, preferably 1.6. I have mine at `c:\temenos\jdk1.6.0_17` (since it's common thing I have it not under `c:\temenos\R10` but one level up).


e) jboss 4.2.3 GA in `c:\temenos\R10\jboss`. We'll need it later when we experiment with Browser client.

f) Batch files to start different components necessary for Browser in `c:\temenos\R10\BATfiles`.

g) Windows user to log in via telnet (again, if you prefer running `remote.cmd` under windows shell, you don't need it). People usually assign administrative rights to this user but it's better not to do it. Instead, create a group (I call it `t24users`), assign full control rights to directory `c:\temenos` for this group and then create your user (`mb10` is the name used here but it of course can be any). Then assign `mb10` user to the group `t24users`. Don't forget to put the path to your `bnk.run` (e.g. `c:\temenos\R10\mb10\bnk.run`) to "Local Path" field of "Profile" tab.

If you copy some files into `c:\temenos` directory, your group rights are OK. If you move files there or if you install something right there, reassign the group rights for new directories and files, otherwise you'll end up with error messages like "Error creating user object" (whatever that might mean).

### 3 Does it work straight away?

 irstly correct `remote.cmd` file. What to look at (corrected is in [blue colour](#)):



```
set HOME=c:\temenos\R10\mb10\bnk.run
set TAFC_HOME=c:\temenos\R10\TAFC
set JBC_CCOMPILER_PATH=c:\temenos\compile
set JAVA_HOME=c:\temenos\jdk1.6.0_17
set PATH=%TAFC_HOME%\bin;%HOME%\bin;%HOME%\t24bin;
      C:\windows\system32;%JAVA_HOME%\bin
```

Make sure that path to TAFC bin directory goes before Java bin folder.  
Hint: there are 2 different jstat.exe programs in these 2 directories and some jBASE utilities implicitly run jstat.exe.

```
set JBASE_LOCALE=en_US
```

Make sure that it's en\_US otherwise you most probably wouldn't be able to log in on a system with non-English locale (day and month will be swapped).

```
set JBCSPOOLERDIR=%HOME%\jspooler
```

We'll see later how to initialise the spooler so it wouldn't spit errors like `'** Error [ NO_PRINTER ] **'`.

Now, let's start Reflection, input user name and password...

```
START GLOBUS Y/N=
```

Press Enter...

```
<Your 'TELNET' connection has terminated>
```

Something is wrong... What?

Firstly let's see the actual error message. Edit the file `remote.cmd` in `bnk.run` directory. At the very end of it comment the line containing

‘‘jprofile.bat’’ and put there cmd instead. Log in again. You’ll see Windows shell prompt:

```
c:\temenos\R10\mb10\bnk.run>
```

Type jsh

Output – something like:

```
The system cannot execute the specified program.
```

To see what it needs you can use a tool like ”Dependency Walker” to open the file libTAFcfmwrk.dll in c:\temenos\R10\T AFC\bin. What we see is the error message:

```
Error: The Side-by-Side configuration information
for "c:\temenos\R10\T AFC\bin\LIBTAFCFRMWRK.DLL"
contains errors. This application has failed to start
because the application configuration is incorrect.
Reinstalling the application may fix this problem
(14001).
```

Don’t try to reinstall application – this will not fix the error.

After some googling it’s evident that we need to install so-called “Visual C++ 2005 Redistributable package” from Microsoft. If you installed T AFC rather than extracted it, you might have noticed that “vcredist” installation was triggered as well... Then it was wrong package included to this particular T AFC installation.

How to find out which package we need? Take a look into libTAFcfmwrk.dll. Search for the following text: Microsoft.VC80.CRT. Ok, here. After that text goes something like: version=‘ ‘8.0.50727.4053’ ’ – that’s all we need.

Search for the package, download it, install. Log in again.

Type `jdiag` to see if all is OK and what parameters we have in our jBASE installation (I said I call it jBASE but I'm not alone – `jdiag` claims to be a “jBASE diagnostic”):

```

jdiag - jBASE diagnostic '$Revision: 1.15 $'
System Information =====
System : WinNT QWERTY 5.1 i386
OS Release : Windows XP Pro, Build 2600, Service Pack 3
NT User : mb10
Time : Wed Oct 06 15:35:07 2010

Environment =====
JBCPORTNO : Not Set
TAFC_HOME : 'C:\temenos\R10\TAFC'
JBCGLOBALDIR : 'C:\temenos\R10\TAFC'
WARNING: JBCDATADIR is not set,
Default 'C:\temenos\R10\TAFC\jbase.data'
WARNING: JBCDATADIR is subdirectory of JBCGLOBALDIR
HOME : 'c:\temenos\R10\mb10\bnk.run'
JEDIFILEPATH : 'c:\temenos\R10\mb10\bnk.run'
JEDIFILENAME_MD : 'VOC'
JEDIFILENAME_SYSTEM : 'C:\temenos\R10\TAFC\src\SYSTEM'
SYSTEM File is (DICT) :
'C:\temenos\R10\TAFC\src\SYSTEMJD'
RELEASE Information : Major 10.0 , Minor 0.2 , Patch
(Change 89685)
Spooler dir (JBCPOOLERDIR) :
'c:\temenos\R10\mb10\bnk.run\jspooler'
JBCEMULATE : 'prime'
TEMP file path : 'C:\WINDOWS\TEMP\'
Object path (JBCOBJECTLIST) :
'c:\temenos\R10\mb10\bnk.run\lib;
c:\temenos\R10\mb10\bnk.run\t24lib'
WARNING: From checking the registry, It appears that VC++ is
not loaded
WARNING: JBC_CCOMPILER_PATH is set to 'c:\temenos\compiler'
jBASE Compiler Run-time :
'C:\temenos\R10\TAFC\config\system.properties'
Program dir (JBCDEV_BIN) : 'c:\temenos\R10\mb10\bnk.run \bin'
Subroutine dir (JBCDEV_LIB) : 'c:\temenos\R10\mb10
\bnk.run\lib'

```

You can ignore warnings. Pay attention to “RELEASE Information” –

you might need it when reporting a bug.

Now type `jsh`.


```
ERROR! Maximum concurrent licensed user limit exceeded
```

What now? Take a look into the following file:

`c:\temenos\R10\T AFC\config\system.properties` – look at the very end. The last line of it most probably looks like:

```
jruntime.license = qP2XhdjqbdbiqzvfDBgPisP3ITiDfhyT
```

After that line you have to add the licence – you need to obtain it from Temenos. Let's assume you have it. Put the licence line after the last one, save the file and try to log in again. Then type `jsh`. Now we're finally there! At last...

```
c:\temenos\R10\mb10\bnk.run>jsh   
jsh mb10 ~-->
```

Let's return `jprofile.bat` call to `remote.cmd` and log in once again to double-check. You'll have `jsh` prompt right away.

Before we continue, it's necessary to initialise jBASE spooler. If directory `jspooler` in `bnk.run` doesn't exist, create it. Then type:

```
SP-NEWTAB
```

The screen looks like:

This operation deletes ALL print jobs and print queues, and re-creates a new spooler with the following criteria:

- 1) Spooler directory = c:\temenos\R10\mb10\bnk.run\jspooler
- 2) Default security of form queues = CREATOR
- 3) Default security of print jobs = CREATOR
- 4) Other owners of spooler =

Enter options 1) to 4) to modify, C to continue or Q to quit :

Type C and press Enter:

Enter options 1) to 4) to modify,

C to continue or Q to quit : C

[ 417 ] File c:\temenos\R10\mb10\bnk.run\jspooler\jobs]D created, type = UD

[ 417 ] File c:\temenos\R10\mb10\bnk.run\jspooler\jobs created, type = UD

[ 417 ] File c:\temenos\R10\mb10\bnk.run\jspooler\jspool\_log]D created, type = J4

[ 417 ] File c:\temenos\R10\mb10\bnk.run\jspooler\jspool\_log created, type = J4

Spooler RESTART completed

Let's try to log in into T24 Classic. To do that type:

EBS.TERMINAL.SELECT EBS-JBASE

or:

ETS

As a result, you see the output:

```
Terminal type set to EBS-JBASE
```

Stop, stop... where “ETS” comes from? Is it an internal command?

No. Now it’s time to look at `VOC` table that is located in the current directory (which is ‘`bnk.run`’).


Type: `CT VOC ETS`

The output:

```
ETS  
001 PA  
002 EBS.TERMINAL.SELECT EBS-JBASE
```

So now you know that in `VOC` there are records with type “PA” where you can store a command (or several commands).

To log in to T24 type `EX`, input T24 user name and password (you can find it in `passwords.txt` if you have Model Bank package or use some well-known standard login as `INPUTT/123456`) . If you’re not allowed to log in, probably `END.DATE.PROFILE` field in `USER` application contains a date in the past, e.g.:

```
jsh mb10 ~-->LIST F.USER 'INPUTTER'   
LIST F.USER 'INPUTTER' PAGE 1 12:08:53 07 OCT 2010  
@ID..... INPUTTER  
@ID..... INPUTTER  
USER.ID..... INPUTTER  
USER.NAME..... INPUTTER  
SIGN.ON.NAME..... INPUTT  
CLASSIFICATION..... INT  
LANGUAGE..... 1  
COMPANY.CODE..... GB0010001 EU0010001 SG0010001  
GB0010002 GB0010003 GB0010004  
DEPARTMENT.CODE..... 1  
PASSWORD.VALIDITY... 20090801M0601  
START.DATE.PROFILE.. 20080122  
END.DATE.PROFILE.... 20091202  
START.TIME..... 1
```

Good news is that you can edit this field (that's why in production system no user shall be granted the access to `jsh`). How to do this – we'll see later.

Other good news is that even in R10 you still can work in terminal environment. Of course some modern things like composite screens will not work but regular input screens (called *VERSIONS*) or inquiries (*ENQUIRY*) most probably will.

If you get the following error, get new maintenance code (again from Temenos):

```
Maintenance license expired, enter new code (20100316  
EURGBTMNS000)
```

If all is OK, we find ourselves logged in. See the screen:



```
R10 Model Bank SELECT APPLICATION
```

---

```
- LAST SIGN.ON, DATE: 30 JUN 2010 TIME: 11:53 ATTEMPTS: 0 -  
07 OCT 2010 12:14:19 USER (05 JAN) VLADIMIR.K [1103,IN]  
ACTION  
AWAITING APPLICATION
```

## 4 What's next?

**T**hroughout this book we'll try to understand:

- What's inside.
- How it works.
- How it can be set to work if it doesn't.

## 5 Shortcuts, applications basics

**L**og out from T24 for time being. To do that type BK or PGM.BREAK at AWAITING APPLICATION prompt. You might notice that we again have a shortcut for a command. This time it's not VOC entry but the application...

What's an application after all in T24 context? The answer is: it's a combination of at least one subroutine containing table structure and business logic (based on so-called "template"), one or several data files (read – jBASE tables), dictionary file and several setup records. You'll be surprised to know that setup records and data records actually are stored the same way – setup records in setup applications and data records in data applications (both types of applications doesn't differ at all).

Are you still there? Let's try to illustrate. The shortcut "BK" is stored in application *ABBREVIATION* as a record and BK is its ID (ID is often referred to as @ID). Let's see it. Type *ABBREVIATION* at AWAITING APPLICATION prompt. Then – S at AWAITING FUNCTION prompt. Then – BK at AWAITING ID prompt.

Note: in this book all that is an application (in T24 context) is written in slanting text like this: *ABBREVIATION*, field names will be in small caps (e.g. END.DATE.PROFILE); everything else that is to stand out of the text in order to catch your eye (commands, file names etc) is like **THAT**.

Could we access the record faster? Yes, typing *ABBREVIATION* S BK at AWAITING APPLICATION prompt. So, in both cases we see the following screen:

```
R10 Model Bank Abbreviation SEE
ABBREVIATION.CODE. BK
-----
1 ORIGINAL.TEXT..... PGM.BREAK
3 CURR.NO..... 1
4. 1 INPUTTER..... 46_AUTHORISER
5. 1 DATE.TIME..... 14 MAR 09 16:29
6 AUTHORISER..... 46_INPUTTER
7 CO.CODE..... GB-001-0001 R10 Model Bank
8 DEPT.CODE..... 1 Implementation

-----
12 OCT 2010 12:56:43 USER (05 JAN) VLADIMIR.K [9187,INPAGE 1
ACTION
AWAITING PAGE INSTRUCTIONS
```

So every time you type BK at Awaiting Application prompt it's the same as if you've typed PGM.BREAK.

Now we need to get back to Awaiting Application prompt. If you have your keys mapped, just press **F1** twice. If they are not – you press:

```
Ctrl -U ↵
```

After first iteration you go back to Awaiting ID prompt, after the second – at Awaiting Application prompt. Type BK now and we're back at jsh:

```
jsh mb10 ~-->
```

## 6 Application data, LIVE and NAU files, record status, audit trail, LIST basics, INT applications

**Y**ou're already familiar with application *ABBREVIATION*. Let's see which jBASE table (or tables) held data for this application.

In jsh type LIST F.ABBREVIATION.

The screen is:


```

LIST F.ABBREVIATION PAGE 1 13:35:41 12 OCT 2010
@ID..... AGC
@ID..... AGC
ABBREVIATION.CODE. AGC
ORIGINAL.TEXT..... ACCT.GEN.CONDITION
RECORD.STATUS.....
CURR.NO..... 1
INPUTTER..... 1_ATB.DIM.MBPARAM-200812.003
DATE.TIME..... 0904100108
AUTHORISER..... 3_INPUTTER
CO.CODE..... GB0010001
DEPT.CODE..... 1
AUDITOR.CODE.....
AUDIT.DATE.TIME...

@ID..... COMM
@ID..... COMM
ABBREVIATION.CODE. COMM
ORIGINAL.TEXT..... FT.COMMISSION.TYPE
RECORD.STATUS.....
CURR.NO..... 1
INPUTTER..... 1_ATB.DIM.MBPARAM-200812.003
DATE.TIME..... 0904100108
...etc

```

Long output? Press Q to stop it.

If we want our particular record, we can use the following command:  
LIST F.ABBREVIATION 'BK'. To save typing, use key  on your keyboard  
to return the previous command and edit it.

```
LIST F.ABBREVIATION 'BK' PAGE 1 13:42:22 12 OCT 2010
@ID..... BK
@ID..... BK
ABBREVIATION.CODE. BK
ORIGINAL.TEXT.... PGM.BREAK
RECORD.STATUS....
CURR.NO..... 1
INPUTTER..... 46_AUTHORISER
DATE.TIME..... 0903141629
AUTHORISER..... 46_INPUTTER
CO.CODE..... GB0010001
DEPT.CODE..... 1
AUDITOR.CODE....
AUDIT.DATE.TIME...


1 Records Listed
```

Why “F.” prefix? In this case prefix “F.” is attached to application name to form the name of “LIVE” file.


More questions from this point:

- What are other cases?
- What’s “LIVE” file?

In T24 there are several types of applications. *ABBREVIATION* has the type ‘‘INT’’ and we can see it in its record in *FILE.CONTROL* (which is also an application). So use *LIST* command again (with slight change to see not all fields but particular one):

```
jsh mb10 ~-->LIST F.FILE.CONTROL 'ABBREVIATION'
CLASSIFICATION 
```

Please note that the command line is wrapped – this can happen not only because of page width limitation here but in real life too. Put attention


to key  to see where command ends.

Output:

```
LIST F.FILE.CONTROL 'ABBREVIATION' CLASSIFICATION PAGE 1
14:05:26 12 OCT 2010
@ID..... CLASSIFICATION
ABBREVIATION INT
1 Records Listed
```


‘‘INT’’ here means that this file has prefix ‘‘F.’’ and is used throughout all the environments (i.e. in all companies). We’ll see later what a company is in T24 context.

Another field of *FILE.CONTROL* application is of interest to us. It’s SUFFIXES.

```
jsh mb10 ~-->LIST F.FILE.CONTROL 'ABBREVIATION'
SUFFIXES 
LIST F.FILE.CONTROL 'ABBREVIATION' SUFFIXES PAGE 1
14:12:50 12 OCT 2010
@ID..... SUFFIXES
ABBREVIATION $NAU
1 Records Listed
```

It means that this application, besides ‘‘LIVE’’ (read – regular) file, has another file for unauthorised records. When you input and commit a record in T24, it doesn’t necessarily mean that this record goes directly to ‘‘LIVE’’ file. Normally, another person is to access this record as well and ‘‘authorise’’ it. Of course, it can be yourself logged in as another user, or you can use a ‘‘zero-authorisation’’ input screen (or ‘‘VERSION’’ – not very relevant name for input screen but that’s how it is in T24).

“NAU-file” has its name the same as “LIVE” with addition of \$NAU to the end:

```
jsh mb10 ~-->LIST F.ABBREVIATION$NAU 
```

```
LIST F.ABBREVIATION$NAU PAGE 1 14:22:28 12 OCT 2010
@ID..... QWERTY
@ID..... QWERTY
ABBREVIATION.CODE. QWERTY
ORIGINAL.TEXT.... ASDFG4
RECORD.STATUS.... INAU
CURR.NO..... 1
INPUTTER..... 5_VLADIMIR.K_OFS_BROWSERTC
DATE.TIME..... 1006240943
AUTHORISER.....
CO.CODE..... GB0010001
DEPT.CODE..... 1
AUDITOR.CODE.....
AUDIT.DATE.TIME...

1 Records Listed
```

Now put your attention to the field RECORD.STATUS. Here it means that to get into “LIVE” file this record is to be authorised. Other widely used status codes are “empty” (i.e. this field doesn’t contain a value) – for authorised records, IHLD or CHLD – for records in status “HOLD”, REVE – for reversed records.

How (and when) this record got here? Let’s see fields INPUTTER and DATE.TIME. The former is user ID (read – ID of this user in application *USER*), the latter – date and time of input. “1006240943” represents 09:43, June 24, 2010. Fields from RECORD.STATUS to AUDIT.DATE.TIME are called “audit trail”.




## 7 FIN and CUS applications, history file, “F” VOC entries, COMPANY, EVAL in LIST


**T**hese are applications that might have a separate set of files for each company. Company in T24 context – large branch that has a separate set of accounts, deals etc. FIN application (e.g. *ACCOUNT*) is always separate, CUS application behaves the same way like *CUSTOMER* – if 2 branches have common customer list, *CUSTOMER* and all CUS-type applications reside in the same files for these branches. For branches with separate customer list CUS-type applications are also in separate files.

How then file names are formed?

Let’s see companies list. Guess where? Correct, application *COMPANY*.

```
jsh mb10 ~-->LIST F.COMPANY MNEMONIC   
  
@ID..... MNEMONIC  
  
SG0010001    SG1  
GB0010003    MF2  
EU0010001    EU1  
GB0010002    MF1  
GB0010004    MF3  
GB0010001    BNK  
  
6 Records Listed
```

In general mnemonic in T24 is the thing that you can easily memorize and use instead of ID. In this particular case, mnemonic is also the thing that is attached to file name prefix. E.g. for company GB0010001 “history” file name for *ACCOUNT* application will be **FBNK.ACCOUNT\$HIS**:


```
jsh mb10 ~-->LIST FBNK.ACCOUNT$HIS 
```

Note that history records have a compound ID which contains the sequential number:

```
LIST FBNK.ACCOUNT$HIS PAGE 1 12:24:05 13 OCT 2010
@ID..... 38288;10
@ID..... 38288;10
ACCOUNT.NUMBER..... 38288;10
CUSTOMER..... 1000006
CATEGORY..... 6001
PRODCAT..... 6001
ACCOUNT.TITLE.1.... ROBERT USD
ACCOUNT.TITLE.2....
SHORT.TITLE..... ROBERTUSD
MNEMONIC..... ROBERTUSD
POSITION.TYPE..... TR
CURRENCY..... USD
CURRENCY.MARKET.... 1
LIMIT.REF.....
ACCOUNT.OFFICER.... 2
OTHER.OFFICER.....
POSTING.RESTRICT...
RECONCILE.ACCT....
INTEREST.LIQU.ACCT.
INTEREST.COMP.ACCT.
INT.NO.BOOKING.....
REFERAL.CODE.....
```

Remember where we can see if particular application has a history file?

```


jsh mb10 ~-->LIST F.FILE.CONTROL 'ACCOUNT' SUFFIXES 
@ID..... SUFFIXES
ACCOUNT          $HIS
                  $NAU

1 Records Listed

```

To see all history records for this particular account we can add selection criteria to LIST or SELECT command. Let's see only IDs (note ONLY keyword:

```

jsh mb10 ~-->LIST FBNK.ACCOUNT$HIS LIKE "'38288'..."
ONLY 
@ID.....


38288;10
38288;5
38288;13
38288;4
38288;1
38288;2
38288;8
38288;3
38288;14
38288;9
38288;12
38288;6
38288;7
38288;11

14 Records Listed

```

Each of these files – “LIVE”, “NAU”, “HIS” – has “VOC entry”, i.e. a corresponding record in VOC file. We already took a look into VOC (see ‘ ‘PA’ ’ VOC entry above). These entries are of type “F”:


```

jsh mb10 ~-->CT VOC FBNK.ACCOUNT$HIS F.ABBREVIATION 
FBNK.ACCOUNT$HIS
001 F
002 ..\bnk.data\ac\FBNK_ACCOUNT#HIS
003 ..\bnk.dict\F_ACCOUNT]D
F.ABBREVIATION
001 F
002 ..\bnk.data\eb\F_ABBREVIATION
003 ..\bnk.dict\F_ABBREVIATION]D

```

Note that in field 2 there's name of actual physical file which represents the data portion of jBASE table and in field 3 there's physical file which represents the dictionary portion. That's how jBASE finds files that are not in the directory (or directories) mentioned in environment variable JEDIFILEPATH. Let's see this variable:

```

jsh mb10 ~-->echo %JEDIFILEPATH% 
c:\temenos\R10\mb10\bnk.run

```


Historically T24 files are addressed via VOC entries; they reside in multiple subdirectories of `c:\temenos\R10\mb10\bnk.data` so JEDIFILEPATH has not much use in T24.

In jBASE it's also possible to address a file directly using its full path:

```
jsh mb10 ~-->LIST ..\bnk.data\ac\FBNK_ACCOUNT#HIS   
38288;10  
17736;1  
38008;17  
38008;21  
38288;5  
38008;5  
USD112100001;1  
38296;9  
37559;20  
37947;4  
37559;19  
37559;25  
37338;5  
37947;7  
38288;13  
38008;12  
USD180050001;1  
37559;26  
37947;2  
37559;33
```

Why we see only IDs? The answer is that jBASE in this case by default looks into the same directory for a dictionary file with the name which is the same as data file name plus suffix "JD". And we don't have such file there. Now we add dictionary to our command:


```

LIST ..\bnk.data\ac\FBNK_ACCOUNT#HIS USING DICT
..\bnk.dict\F_ACCOUNT]D 
@ID..... 38288;10
ACCOUNT.NUMBER..... 38288;10
CUSTOMER..... 1000006
CATEGORY..... 6001
PRODCAT..... 6001
ACCOUNT.TITLE.1..... ROBERT USD
ACCOUNT.TITLE.2.....
SHORT.TITLE..... ROBERTUSD
MNEMONIC..... ROBERTUSD
POSITION.TYPE..... TR
CURRENCY..... USD
CURRENCY.MARKET..... 1
LIMIT.REF.....
ACCOUNT.OFFICER..... 2
OTHER.OFFICER.....
POSTING.RESTRICT.....
RECONCILE.ACCT.....
INTEREST.LIQU.ACCT.....
INTEREST.COMP.ACCT.....
INT.NO.BOOKING.....

```

Now it's time for "EVAL" keyword in LIST. With it we can output calculated things – and not only ones related to jBASE table. See how to output only the sequential history number which is part of ID:


```

jsh mb10 ~-->LIST FBK.ACCOUNT$HIS EVAL
"FIELD(@ID,',';',';2)" 
@ID..... FIELD(@ID,"";",2)
38288;10      10
17736;1       1
38008;17      17
38008;21      21
38288;5       5
38008;5       5
USD112100001;1  1
38296;9       9
37559;20      20
37947;4       4
37559;19      19
37559;25      25
37338;5       5
37947;7       7
38288;13      13
38008;12      12
USD180050001;1  1
37559;26      26
37947;2       2
37559;33      33

```


Another trick – let’s use LIST command as a calculator. We use the table with 1 record to keep the output tidy:


```

jsh mb10 ~-->LIST F.SPF EVAL "2*2" 
@ID... 2*2.....
SYSTEM 4
1 Records Listed

```


We can also check some jBASE stuff:



```
jsh mb10 ~-->LIST F.SPF EVAL "OCONV(TIME(), 'MT')"   
@ID... OCONV(TIME(), "MT")  
SYSTEM 14:11  
1 Records Listed
```

```
jsh mb10 ~-->LIST F.SPF EVAL "INDEX('QWERTY', 'RTY', 1)"  
  
@ID... INDEX('QWERTY', 'RTY', 1)  
SYSTEM 4  
1 Records Listed
```

Ok, let's move further...



## 8 SELECT and SELECT lists, COUNT, “SAVING” and “UNIQUE” in SELECT

ow we want not to display some records on the screen, but to select them for further processing:

```
jsh mb10 ~-->SELECT FBNK.ACCOUNT$HIS LIKE "'38288'..."  
  
14 Records selected  
>SAVE.LIST MY.LIST   
14 record(s) saved to list 'MY.LIST'
```




Where can we find this list? In the folder that is set in environment variable JBCLISTFILE:

```
jsh mb10 ~-->echo %JBCLISTFILE%   
c:\temenos\R10\mb10\bnk.run\&SAVEDLISTS&  
  
jsh mb10 ~-->CT &SAVEDLISTS& MY.LIST   
'MY.LIST' record not found
```



Well... if record is not there it most probably means that the way the environment variable was specified in `remote.cmd` is not much compatible with Windows (I mean usage of ampersands). Is this list lost for us then? Let's see.

```
jsh mb10 ~-->GET.LIST MY.LIST   
14 Records selected  
  
>
```



Where it was stored then? Firstly input the command `CLEARSELECT` to clear default `SELECT` list, otherwise our further commands will be affected by it. Then:

```
jsh mb10 ~-->CT . MY.LIST   
MY.LIST  
001 38288;10  
002 38288;5  
003 38288;13  
004 38288;4  
005 38288;1  
006 38288;2  
007 38288;8  
008 38288;3  
009 38288;14  
010 38288;9  
011 38288;12  
012 38288;6  
013 38288;7  
014 38288;11
```




So it was saved in current directory (`c:\temenos\R10\mb10\bnk.run`). To keep it tidy let's amend environment variable `JBCLISTFILE` to something more compatible – e.g. `SAVEDLISTS`, create that directory in `bnk.run` and log in once again. When we repeat previous `SELECT` and `SAVE.LIST` commands, we see that there is the file `MY.LIST` stored in `SAVEDLISTS` subdirectory:

```
jsh mb10 ~-->cmd   
c:\temenos\R10\mb10\bnk.run>dir SAVEDLISTS   
15.07.2010 11:33 131 MY.LIST  
c:\temenos\R10\mb10\bnk.run>
```

Let's go back to the history file of application `ACCOUNT`. How many records are there? Firstly we return to `jsh` prompt and then use `COUNT` command:



```
c:\temenos\R10\mb10\bnk.run>exit   
jsh mb10 ~-->COUNT FBNK.ACCOUNT$HIS   
228 Records counted
```

As we saw earlier, there might be several records for the same account reflecting all its history. But how can we see how many accounts at all have history? It's possible with **SELECT** command using **UNIQUE** clause. We'll also see that the result of **SELECT** not necessarily will be ID but anything else as well (see **SAVING** clause); we'll also sort the records by ID using **SSELECT**:

```
jsh mb10 ~-->SSELECT FBNK.ACCOUNT$HIS SAVING UNIQUE  
EVAL "FIELD(@ID;',' ,1)"   
63 Records selected  
  
>SAVE.LIST AC.UNIQ   
63 record(s) saved to list 'AC.UNIQ'  
  
jsh mb10 ~-->CT SAVEDLISTS AC.UNIQ   
  
AC.UNIQ  
001 10596  
002 10944  
003 10979  
004 11495  
005 11525  
...  
060 USD180100001  
061 USD180200001  
062 USD180300001  
063 USD195050001
```



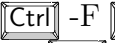


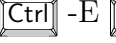

You probably noticed that when we have an active **SELECT** list, the **jsh** prompt changes to “>” sign. Knowing that you have an active **SELECT**

list is a good thing. Let's try to use it to find out, for example, how many accounts that are in history do not present in "LIVE" file (i.e. were reversed or closed):

```
jsh mb10 ~-->GET.LIST AC.UNIQ   
  
63 Records selected  
  
>SELECT FBNK.ACCOUNT   
  
** Error [ 202 ] **  
Record '37567' is not on file.  
** Error [ 202 ] **  
Record '45705' is not on file.  
  
61 Records selected  
  
>
```

Now we know these 2 records.

## 9 More navigation in Classic

et's log in to T24 Classic (before that don't forget to clear the active SELECT list). We'll take a look at *ACCOUNT* record – one of these 2 that don't present in "LIVE" file. Type **ACCOUNT S 37567** at **AWAITING APPLICATION** prompt, then navigate down using  or  ; either you can go directly to the end of this record using  or  .

```

R10 Model Bank ACCOUNT SEE
                                USD Savings Account
ACCOUNT.NUMBER.... 37567 ; 3    DAVEUSD1
-----
94 CHARGE.MKT..... 1          Currency Market
95 INTEREST.CCY..... USD      US Dollar
96 INTEREST.MKT..... 1          Currency Market
99. 1 ALT.ACCT.TYPE.. LEGACY
108 ALLOW.NETTING.... NO
142 SINGLE.LIMIT.... Y
163 CLOSED.ONLINE.... Y
167 DATE.LAST.UPDATE.. 09 DEC 2009
202 RECORD.STATUS.... CLOSED    Account Closed
203 CURR.NO..... 3
204. 1 INPUTTER..... 2038_SEAT.AUTH
205. 1 DATE.TIME..... 06 FEB 10 20:48
206 AUTHORISER..... SY_CLOSURE
207 CO.CODE..... GB-001-0001    R10 Model Bank
208 DEPT.CODE..... 1            Implementation
-----
15 OCT 2010 09:30:20 USER (05 JAN) VLADIMIR.K [7323,INPAGE 3
ACTION
AWAITING PAGE INSTRUCTIONS

```


We see that RECORD.STATUS field value is CLOSED and ID contains ';' – this means that this record is no more in “LIVE” file.

If we knew page number we also could go directly here using command “P3” at AWAITING PAGE INSTRUCTIONS prompt.


At this stage you’re probably tired of pressing key combinations like **Ctrl** -F **↵**. It’s time to map your keys – in case your terminal emulation program allows it. So map:


**Ctrl** -U **↵** to **F1** – “Up”


**Ctrl** -B **↵** to **F2** – “Back”

**Ctrl**-F  to **F3** – “Forward”

**Ctrl**-E  to **F4** – “End”


**Ctrl**-V  to **F5** – “Verify, i.e. commit”

**Ctrl**-W  to **F6** – “double-V, so **W** – commit and move to next record”

**Ctrl**-T  to **F7** – “Text edit”

## 10 Introduction into programs and subroutines, conclusion for applications

**W**hat actually happens when you type **ACCOUNT** at **AWAITING APPLICATION** prompt? Logoff from T24. At **jsh** type:

```
jsh mb10 ~-->jshow -c ACCOUNT   
Subroutine:  c:\temenos\R10\mb10\bnk.run\t24lib  
             \acm_accountopening\lib0.dll  
jBC ACCOUNT version 10.0 Tue Feb 16 17:19:11 2010  
jBC ACCOUNT source file source/R10.000/win32_TAFCR10.000
```

So when you type **ACCOUNT** at **AWAITING APPLICATION** prompt a subroutine with the same name is launched. Actually almost all in T24 is subroutine. To log in to T24 in Classic mode we type **EX** at **jsh**. “EX” is a program and all that works further is a subroutine. See:

```

jsh mb10 ~-->jshow -c EX ↩
Executable: c:\temenos\R10\mb10\bnk.run\t24bin\eb_api
\EX.dll
jBC main() version 10.0 Tue Feb 16 18:33:05 2010
jBC main() source file source/R10.000/win32_TAFCR10.000

jsh mb10 ~-->jshow -c ABBREVIATION ↩
Subroutine: c:\temenos\R10\mb10\bnk.run\t24lib
\eb_systemtables\lib0.dll
jBC ABBREVIATION version 10.0 Wed Feb 17 09:36:01 2010
jBC ABBREVIATION source file source
/R10.000/win32_TAFCR10.000

```

So, to end up applications theme: we saw different parts of application – data files, dictionary, subroutine with business logic, *FILE.CONTROL* record. What else? See also corresponding records (i.e. with ID = application name) in *PGM.FILE* (field TYPE) and *STANDARD.SELECTION*.

The main thing you need to understand: if business logic lies in application subroutine (also called “template”), it’s not possible to update or create a record in T24 application just editing it in front-end or issuing SQL-like commands. Of course technically it’s possible but it will not create a correct record with all necessary rules applied and other linked tables updated. You might ask: how to do it then – only to input manually inside T24? The answer is: OFS.

## 11 Introduction into OFS, more about functions, setup OFS.SOURCE, tSS, simple enquiry output

**OFS** stands for “Open financial service” though it’s not a service – just a script language (though without programming capabilities) to manipulate records in T24 application. And it’s a must for a techie to know. Manuals give quite comprehensive explanation of OFS syntax so I will not go deeper into it here. Instead let’s try to use it to create or update T24 applications.

Firstly let’s create an appropriate record in *OFS.SOURCE* application. Log in into T24 and type **OFS.SOURCE, I TEST** at **AWAITING APPLICATION** prompt. “I” is the function used to input a record.

What I can say now about functions? In brief the rules are:

- Use function I in “pure” application or in regular *VERSION* to input an unauthorised record (use **F3** after I to have ID assigned by T24 automatically).
- Use function I in “zero-authorisation” *VERSION* to input an authorised record.
- Unauthorised record can be deleted (function D), authorised (function A) or (depending on application) put to “HOLD” (type IHLD to do it).
- Authorised record can (depending on application) be amended (again function I; not all fields can be amended – there are so-called “NOCHANGE” ones) – with or without further authorisation – or reversed (function R) – again authorisation might or might not be required. Reversed records go to “history” file – for those applications which support history (see above *FILE.CONTROL*, field SUFFIXES). To history file also go “LIVE” records after each change (we saw it above with *ACCOUNT*) or after their retention period expires (e.g. for *FUNDS.TRANSFER* it happens every day after close



of business – COB).

- Use function **S** to see a record. If you're not going to edit a record, better use this function instead of **I** to avoid locks but keep in mind that all fields that are empty will not be shown.

- All records can be printed (function **P**) though it is mainly used for output to flat file.

- Use function **C** to copy a record (wherever the application allows that).

- Authorised records can be listed using function **L**. “**L L**” is used to search authorised file, “**L -something**” allows to jump to a record starting with **something**, e.g. “**EB.ERROR L -D**” allows to list *EB.ERROR* records starting from **DC-999.NOT.ALLOWED**.

- Unauthorised records can be listed using function **E**. “**E E**” is used to search unauthorised file, “**E -something**” also works the same way as “**L -something**”.

So – command is “**OFS.SOURCE, I TEST**”. Why the comma after application name? Answer – to create the record in one step. *VERSION* (read – input screen) name consists of name of application followed by comma and a suffix. This suffix may be absent so we get just application name and a comma. Don't think that comma itself means zero-authorisation – it's an internal agreement that “comma-versions” are simple *VERSION* records that are setup for zero authorisation with no additional features like selective fields display, tabbed view and so on:

R10 Model Bank VERSION SEE

```
PGM.NAME.VERSION..      OFS.SOURCE,
-----
2 RECORDS.PER.PAGE..    1
3 FIELDS.PER.LINE...   1
46 NO.OF.AUTH.....    0
55.  1.  1 VAL.ASSOC... LOGIN.ID
55.  1.  2 VAL.ASSOC... EB.PHANT.ID
57 LOCAL.REF.FIELD...  LOCAL.REF
65 REPORT.LOCKS.....   YES
103 CURR.NO.....       1
104.  1 INPUTTER.....  6_WOODY-00_OFS_BROWSERTC
105.  1 DATE.TIME..... 02 APR 10 17:40
106 AUTHORISER.....    6_WOODY-00_OFS_BROWSERTC
107 CO.CODE.....       GB-001-0001      R10 Model Bank
108 DEPT.CODE.....     4      Retail Banking User 4
```

```
-----
15 OCT 2010 13:29:32 USER (05 JAN) VLADIMIR.K PAGE 1
ACTION
AWAITING PAGE INSTRUCTIONS
```

It's a good idea never to amend any comma version and leave them for the purpose they were designed for.

Under "appropriate" *OFS.SOURCE* record I mean the one with the following field values:

```
R10 Model Bank      OFS SOURCE, INPUT

SOURCE.NAME.....  TEST
-----
1 DESCRIPTION.....  TEST ONE
2 SOURCE.TYPE.....  TELNET
3.  1 LOGIN.ID..... any
4.  1 EB.PHANT.ID...
5 MAX.CONNECTIONS...
6 RESTRICT.LINK....
7 INITIAL.ROUTINE...
8 CLOSE.ROUTINE....
9 IN.MSG.RTN.....
10 OUT.MSG.RTN.....
11 MSG.PRE.RTN.....
12 MSG.POST.RTN....
13 LOG.FILE.DIR....
14 LOG.DETAIL.LEVEL. NONE
15 OFFLINE.QUEUE...
16 MAINT.MSG.DETS...
-----
16 OCT 2010 09:12:16 USER (05 JAN) VLAD.K PAGE 1 >>>4>>>
ACTION
```

Note that field LOG.DETAIL.LEVEL was automatically set to “NONE” – this is business logic we were talking about earlier.

```

R10 Model Bank      OFS SOURCE, INPUT

SOURCE.NAME.....  TEST
-----
17 DET.PREFIX.....
18 IN.QUEUE.DIR....
19 IN.QUEUE.NAME....
20 OUT.QUEUE.DIR....
21 OUT.QUEUE.NAME....
22 QUEUE.INIT.RTN...
23 QUEUE.CLOSE.RTN...
24 SYNTAX.TYPE..... OFS
25.  1 LOCAL.REF....
26 GENERIC.USER..... INPUTTER
27 IN.DIR.RTN.....
28 VERSION.....
29 IB.USER.CHECK....
30 EOD.VALIDATE.....
31 FIELD.VAL.....
32.  1 ATTRIBUTES....
-----
16 OCT 2010 09:12:16 USER (05 JAN) VLAD.K PAGE 2 >>>4>>>
ACTION


```

To jump to particular field just type its number. E.g. after typing “any” on the first page you can go directly to the second one using **F4** **F3** and then type 24 to jump directly to the field SYNTAX.TYPE. Note that you can’t jump to so-called “NOINPUT” fields – try, for example, to type 25.1.


Having populated fields SYNTAX.TYPE and GENERIC.USER, use **F5** to commit the record.

This is the required minimum we can proceed with; we’ll add then some additional features to see how they work.

Log out of T24 to **jsh**. Then launch the program that represents another point of entry into T24 using our *OFS.SOURCE* record ID as a parameter:

```
jsh mb10 ~-->tSS TEST   
<tSS version="1.1"><t24version>R10.000</t24version>  
<t24pid>556</t24pid><t24ofssource>TEST</t24ofssource>  
<clientIP/></tSS>
```

Then type an OFS message. The simplest one looks like:

```
ENQUIRY.SELECT, , INPUTT/123456,%USER   
  
,@ID::@ID/USER.NAME::USER.NAME/SIGN.ON.NAME  
::SIGN.ON.NAME/LANGUAGE::LANGUAGE/DEPARTMENT.CODE  
::DEPARTMENT.CODE,"ACCTEXEC " " Account Executive "  
"ACCTEXEC1 " "1 " "1 ", "ARCUSER " "TFOFFICER " "TFOFFICER  
" "1 " "1 ", "AUTHORISER " "AUTHORISER " "AUTHOR " "1 " "1  
", "BTOOLSUSER " "BUSINESS TOOLS USER " "BTOOLS " "1 " "1  
", "BUO  
  
etc
```

This was output from so-called “percent enquiry”. These enquiries are called whenever you use L function for an application. So it’s a good idea never to amend such enquiries (exactly the same advice as for comma versions – see above).

To exit from tSS type EXIT.

Note: though tSS is a part of older connectivity scheme used in R08 and below, it’s still around and useful for such purposes as one described above.

## 12 OFS – inputting an application record: VERSION creation, “VALIDATE” option, couple of tests, STANDARD.SELECTION check

Let's start tSS again and try to input an application record. Let's try to create a *FUNDS.TRANSFER* record. Why *FUNDS.TRANSFER*? Because it's one of the main financial applications which is quite complex, has a lot of dependencies; playing with simple things like *ABBREVIATION* will be too boring.

But firstly we need our own *VERSION* since using comma version wouldn't allow us to try many things that are worth trying. An appropriate name looks like *FUNDS.TRANSFER,TEST* (remember rules for version names?) Let's create it (only necessary fields will be shown):

R10 Model Bank	VERSION, INPUT
PGM.NAME.VERSION..	FUNDS.TRANSFER,TEST
-----	
2 RECORDS.PER.PAGE..	1
3 FIELDS.PER.LINE...	1
...	
46 NO.OF.AUTH.....	1
...	
73 EXC.INC.RTN.....	NO
-----	

“NO” in the last field allows us to have really “pure” application (otherwise if there is any routine mentioned in *VERSION.CONTROL>FUNDS.TRANSFER* record, it will be triggered while we experiment).

Note the method of specifying a record: *APPLICATION>RECORD@ID*. We'll use it later.

We'll start form "VALIDATE" mode to gradually get the correct record contents. Start tSS, type the following OFS message:

```
jsh mb10 ~-->tSS TEST 
<tSS version="1.1"><t24version>R10.000</t24version>
<t24pid>556</t24pid><t24ofssource>TEST</t24ofssource>
<clientIP/></tSS>

FT,TEST/I/VALIDATE,INPUTT/123456, 

FT10005CKNDK//-1/NO,TRANSACTION.TYPE:1:1=INPUT MISSING,
TRANSACTION.TYPE:1:1=INPUT MISSING
```

We haven't supplied any values for any fields and now see that some fields are mandatory. Note that we haven't specified an ID and it was assigned by T24 automatically (FT10005CKNDK).

To correct this error let's put some value to the field TRANSACTION.TYPE:

```
FT,TEST/I/VALIDATE,INPUTT/123456,,TRANSACTION.TYPE:=1
FT10005NNVBH//-1/NO,TRANSACTION.TYPE:1:1=TOO FEW CHARACTERS
```

Too few... OK, make it long:

```
FT,TEST/I/VALIDATE,INPUTT/123456,,TRANSACTION.TYPE:=12345
FT1000588XQN//-1/NO,TRANSACTION.TYPE:1:1=TOO MANY CHARACTERS
```

Another example of business logic. How long this field actually should be? Full answer could be found in the source code of this application template in case we had it. Sometimes the answer to this question can be found in application which was already mentioned – *STANDARD.SELECTION*.

Log in to T24, our command at AWAITING APPLICATION prompt is quite short since it contains 2 abbreviations with a function between them:

SS S FT

Then navigate to the second page:

R10 Model Bank	STANDARD SELECTION FIELDS SEE
FILE.NAME.....	FUNDS.TRANSFER
-----	
11. 2 SYS.LANG.FIELD.	N
12. 2 SYS.GENERATED.	Y
1. 3 SYS.FIELD.NAME.	TRANSACTION.TYPE
2. 3 SYS.TYPE.....	D
3. 3. 1 SYS.FIELD.NO	1
4. 3. 1 SYS.VAL.PROG	IN2A&&NOCHANGE
6. 3 SYS.DISPLAY.FMT	4L
7. 3 SYS.ALT.INDEX..	N
10. 3 SYS.SINGLE.MULT	S
11. 3 SYS.LANG.FIELD.	N
12. 3 SYS.GENERATED..	Y
14. 3. 1 SYS.REL.FILE	FT.TXN.TYPE.CONDITION Defines the default conditions f
1. 4 SYS.FIELD.NAME.	DEBIT.ACCT.NO
2. 4 SYS.TYPE.....	D
3. 4. 1 SYS.FIELD.NO	2
4. 4. 1 SYS.VAL.PROG	IN2.ALLACCVL&&NOCHANGE
-----	

Fields from 1.3 to 14.3.1 are so-called “associated multi-value block”. If you’re still not very comfortable with multi- and sub-values, we’ll later have some examples. But for now all you need to know that these fields form a group and we’re interested in three fields from this group.

- 1.3 – this is our *FT* field, TRANSACTION.TYPE.
- 6.3 – it says that this field is displayed as 4 left-justified symbols. It probably means that up to 4 characters can be input (I use “probably” because it’s not stipulated here – it’s just formatting instruction.).



- 14.3.1 – it says that into this field can be input only a value that presents as a record in application *FT.TXN.TYPE.CONDITION*.


A look into *SS* record for that application says us that *SYS.DISPLAY.FMT* field for *ID* has formatting instructions as *4R*. So *- 4* is most probably the maximum size, what's the minimum? We of course can experiment with that (since *4* gives us not many iterations to try) but what if maximum length is, say, *100*? Next chapters will help us to do that.


## 13 Writing a simple T24 subroutine

**W**hy not program? Actually, in our case a subroutine is better than a program because a subroutine can work indside T24 and therefore can have access to T24 global variables pool.

First thing to do is to create a directory for our subroutines and programs. The names of such directories are usually in uppercase and end with “.BP”. That suffix is handy when you need to transfer the source code to another environment (such names are allowed in *DL.DEFINE* application which is used for this purpose).


A directory of course can be created with operation system command “*mkdir*” but we’d better do it from *jsh* prompt. A hint: folders with files are treated like tables with records in *jBASE*. So we can use *jBASE* command *CREATE-FILE*. To skip creation of directory portion we use *DATA* clause, *TYPE* clause makes it a directory. Other *jBASE* commands will work as well:

```
jsh mb10 ~-->CREATE-FILE DATA ETC.BP TYPE=UD 
[ 417 ] File ETC.BP created , type = UD

jsh mb10 ~-->LIST ETC.BP 

No Records Listed
```

This directory was created in our home directory `bnk.run`. To create a subroutine we firstly need to invent a name for it. This name is to be meaningful. Moreover, this name has to be unique to our T24 environment. How do we know that? For example, we like the name “TEST.RTN”.

```
jsh mb10 ~-->jshow -c TEST.RTN   
Subroutine:  c:\temenos\R10\mb10\bnk.run\t24lib  
\se_test\lib2.dll  
jBC TEST.RTN version 10.0 Tue Feb 16 20:00:48 2010  
jBC TEST.RTN source file source/R10.000/win32_TAFCR10.000
```

No, it’s busy already. But what will happen if you occasionally choose such a name? Let’s see. Edit the source file using JED editor (or `notepad.exe`) to have there the following text:

```
SUBROUTINE TEST.RTN  
$INSERT I_COMMON  
$INSERT I_EQUATE  
  
TEXT = 'THIS IS A TEST'  
CALL REM  
  
RETURN  
END
```

We’ll discuss the contents later on. Note only that the name of subroutine or program should be the same as source code file name (including case – jBASE is case-sensitive though it’s not so evident under Windows). For now – let’s compile it. The utility `EB.COMPILE` that was used in earlier T24 releases is no longer available so we simply use jBASE commands `BASIC` and `CATALOG`. Note option `-IGLOBUS.BP` to let compiler know where to look for `I_COMMON` and `I_EQUATE` – so-called “insert” files that are mandatory for all subroutines that run under T24 (though can be omitted for very simple ones that don’t use T24 global variables):

```

jsh mb10 ~-->BASIC -IGLOBUS.BP ETC.BP TEST.RTN ↵
[jpp error 1] line 2:
      I.COMMON : File not located - suggest option -I
<pathname>
"TEST.RTN":
fatal pre-processor error - compilation abort

1 error was found
jbbcom -f -d -aETC.BP -IGLOBUS.BP BASIC.1.b failed , command
returned a code of 1
jcompile: Returned an error code of 8
** Unable to compile source TEST.RTN **

```

Quite unexpectedly... Though you might find a new release lacking something you're used to... In this case it's VOC entry GLOBUS.BP:

```

jsh mb10 ~-->CT VOC GLOBUS.BP ↵
'GLOBUS.BP' record not found

```

Funny that there are still Universe object files in RG.B0.0 subdirectory of bnk.data/eb that nobody cares to remove but the vital things like that are disappearing... not a big problem, let's create it. In jsh type JED VOC GLOBUS.BP, fill necessary fields, press **Esc** to get to command line, then type FI to save the record:


```

NEW *File VOC , Record 'GLOBUS.BP' Insert 13:06:10
Command-> FI
0001 F
0002 ../T24.BP
0003
----- End Of Record -----


```

Now we can compile and CATALOG this routine:

```

jsh mb10 ~-->BASIC -IGLOBUS.BP ETC.BP TEST.RTN 
TEST.RTN
BASIC_1.c


Source file TEST.RTN compiled successfully

jsh mb10 ~-->CATALOG ETC.BP TEST.RTN 
TEST.RTN Object TEST.RTN cataloged successfully
mt -nologo -manifest c:\temenos\R10\mb10\bnk.run\lib
\lib2.dll.manifest -outputresource:
c:\temenos\R10\mb10\bnk.run\lib\lib2.dll;2 failed , command
returned a code of -1
Library c:\temenos\R10\mb10\bnk.run\lib\lib2.dll rebuild
okay

```


Don't pay attention to "failed" message. Check what we have now for this routine:

```





jsh mb10 ~-->jshow -c TEST.RTN 
Subroutine:  c:\temenos\R10\mb10\bnk.run\lib\lib2.dll
    jBC TEST.RTN version 10.0 Tue Oct 19 16:11:20 2010
    jBC TEST.RTN source file ETC.BP
Subroutine (DUP!!):  c:\temenos\R10\mb10\bnk.run\t24lib\
    se_test\lib2.dll
    jBC TEST.RTN version 10.0 Tue Feb 16 20:00:48 2010
    jBC TEST.RTN source file source
    /R10.000/win32.TAFCR10.000

```

So now we have 2 routines with same name registered (or CATALOG'ued) in the system. And whenever TEST.RTN is called, a local one will be loaded rather than a core one – because environment variable is set accordingly:

```
jsh mb10 ~-->echo %JBCOBJECTLIST%   
c:\temenos\R10\mb10\bnk.run\lib;  
c:\temenos\R10\mb10\bnk.run\t24lib
```

A good idea is either to put `t24lib` first or to check all local routines if such name was already registered. For this particular routine we'll DECATALOG it, delete object file (one that starts from "\$") and rename to some name that is not used (the latter will be done from `jsh` to illustrate more `jBASE` commands):

```
jsh mb10 ~-->DECATALOG ETC.BP TEST.RTN   
Object TEST.RTN decataloged successfully  
...  
jsh mb10 ~-->DELETE ETC.BP $TEST.RTN   
1 record(s) deleted.  
  
jsh mb10 ~-->jshow -c TEST2.RTN   
jsh mb10 ~-->COPY FROM ETC.BP TEST.RTN,TEST2.RTN  
1 records copied  
  
jsh mb10 ~-->DELETE ETC.BP TEST.RTN   
1 record(s) deleted.
```

Now let's compile `TEST2.RTN` – use `BASIC` and `CATALOG` commands as before, use `jshow` to check if this subroutine is properly registered. Don't forget to change `SUBROUTINE TEST.RTN` to `SUBROUTINE TEST2.RTN` in the source code before compilation. To run this subroutine as a standalone one inside T24 it's necessary to create a record in application *PGM.FILE*:

```

R10 Model Bank      PROGRAM FILE, INPUT
PROGRAM            TEST2.RTN
-----
1 TYPE..... M
...
5 PRODUCT..... EB          CORE
-----

```

Type “M” means “mainline routine”. Though this concept is obsolete in browser, you might need such type of routine to test some things (actually we’re doing this right now).

Type TEST2.RTN at AWAITING APPLICATION prompt:

```

R10 Model Bank      TEST2.RTN
-----

-----

19 OCT 2010 13:53:38 USER (05 JAN) VLADIMIR.K [8788,IN]
ACTION
CONTINUE (Y)          THIS IS A TEST

```

Type Y to return to AWAITING APPLICATION prompt.

Looking into the source code of TEST2.RTN, it becomes evident that contents of variable TEXT were output as a message by subroutine REM. Variable TEXT is a global one, it is declared in I\_COMMON insert file. All is quite simple. So far.

## 14 Getting application information from a routine

**A** mend the routine TEST2.RTN. Make it look like:

```
TEST2.RTN
001 SUBROUTINE TEST2.RTN
002 $INSERT I_COMMON
003 $INSERT I_EQUATE
004
005 V$FUNCTION = 'TEST'
006 CALL FT.TXN.TYPE.CONDITION
007
008 DEBUG
009
010 RETURN
011 END
```

Here the hint: we can call application template with function length greater than 1 character and get its full structure. Main arrays that hold this structure have names F – name, N – size and T – type. For ID there are respective variables ID.F, ID.N and ID.T (See I\_COMMON and I\_RULES in ../T24\_BP for more information).

When we compile and run this subroutine, we can check these arrays and variables in debugger. The one we need to see possible length of field

TRANSACTION.TYPE in *FT* is ID.T:

```
ACTION DEBUG statement seen
0008 DEBUG
jBASE debugger->V ID.N
COMMON variables
  ID.N                : 4.2
```

It means that ID of application *FT.TXN.TYPE.CONDITION* and therefore field TRANSACTION.TYPE in *FT* can have length from 2 to 4... of what?

```
jBASE debugger->V ID.T
COMMON variables
  ID.T                : A
```

...from 2 to 4 alphanumeric characters. T24 manual "In2 routines" says: "IN2A: allows alphanumeric input. The character set is defined in *ASCII.VALUES* and *ASCII.VAL.TABLE*".

So far we'll stop exploring this thing further. Type C to continue subroutine execution or Q to quit it. You can check 2 applications mentioned in the manual by yourself; note only that "A" value in the variable ID.T corresponds to "IN2A" routine; this is a rule that is true for all other types of fields. Let's return to creating *FT* record using OFS.

## 15 OFS – application record creation – continued, overrides, fields GTS.CONTROL and NO.OF.AUTH



**W**e've stopped at the moment when we've tried to put a value to TRANSACTION.TYPE field. We now know that we can put there only a value that corresponds to ID in application *FT.TXN.TYPE.CONDITION*. Let's



choose a value. Log in to T24, type FT.TXN.TYPE.CONDITION L at AWAITING APPLICATION prompt:

R10 Model Bank FT.TXN.TYPE.CONDITION - Default List			
ID	DESCRIPTION	SHORT.DESCR	TXN.CODE.FUNCT.
1	AC Account Transfer	Acct Transfer	213
2	ACCLAZ Loan Preclosure	AZ LN Preclose	483
...			

OK, AC looks as what we need. And we'll need less fields to populate than if we have chosen something else.

Some more navigation hint: to go into this record type 1  , then function (e.g. S)  :


```

R10 Model Bank          FT.TRANSACTION.TYPE.CONDITION SEE
                          TRANSACTION.TYPE.. AC
-----
1.  1 GB DESCRIPTION. Account Transfer
2.  1 GB SHORT.DESCR. Acct Transfer
3 TXN.CODE.CR..... 213 Transfer
4 TXN.CODE.DR..... 213 Transfer
5 STO.TXN.CODE.CR... 214 Standing Order
6 STO.TXN.CODE.DR... 214 Standing Order
7 DR.CHARGE.TXN.CODE 234 Account Transfer Charges
8 DR.CHEQUE.TXN.CODE 201 Outward Cheque - Dr
11 FORW.VALUE.MAXIMUM +05W
12 BACK.VALUE.MAXIMUM -05W
13.  1 PAYMENT.TYPE... ALL
14.  1 PAYMENT.VALUE.. Y
15.  1 CUSTOMER.FLOAT. 0
16.  1 SAME.CUST.FLOAT 0
17 DR.ADVICE.REQD.Y.N N
18 CR.ADVICE.REQD.Y.N N
-----

```

Back to OFS input. Type tSS TEST at jsh prompt. Then:


```

FT,TEST/I/VALIDATE,INPUTT/123456,,TRANSACTION.TYPE::=AC 
FT10005H1YX7// -1/NO,CREDIT.AMOUNT:1:1=TRF AMOUNT MUST BE
INPUT IN #6 OR #14

```

After some time we finally get to more or less correct message which is:

```


FT,TEST/I/VALIDATE,INPUTT/123456,,TRANSACTION.TYPE::=AC,
CREDIT.AMOUNT::=100.00,CREDIT.CURRENCY::=EUR,
DEBIT.ACCT.NO::=14637,CREDIT.ACCT.NO::=10715 

FT10005QWFC//1,TRANSACTION.TYPE:1:1=AC,
DEBIT.ACCT.NO:1:1=14637,CURRENCY.MKT.DR:1:1=1,
DEBIT.CURRENCY:1:1=EUR,DEBIT.VALUE.DATE:1:1=20100105,
CREDIT.ACCT.NO:1:1=10715,CURRENCY.MKT.CR:1:1=1,
CREDIT.CURRENCY:1:1=EUR,CREDIT.AMOUNT:1:1=100.00,
CREDIT.VALUE.DATE:1:1=20100105,PROCESSING.DATE:1:1=20100105,
CHARGE.COM.DISPLAY:1:1=NO,COMMISSION.CODE:1:1=DEBIT
PLUS CHARGES,CHARGE.CODE:1:1=DEBIT PLUS CHARGES,
PROFIT.CENTRE.CUST:1:1=100283,RETURN.TO.DEPT:1:1=NO,
FED.FUNDS:1:1=NO,POSITION.TYPE:1:1=TR,
AMOUNT.DEBITED:1:1=EUR100.00,AMOUNT.CREDITED:1:1=EUR100.00,
CREDIT.COMP.CODE:1:1=GB0010001,DEBIT.COMP.CODE:1:1=GB0010001,
LOC.AMT.DEBITED:1:1=145.00,LOC.AMT.CREDITED:1:1=145.00,
CUST.GROUP.LEVEL:1:1=99,DEBIT.CUSTOMER:1:1=100283,
CREDIT.CUSTOMER:1:1=100318,DR.ADVISE.REQD.Y.N:1:1=N,
CR.ADVISE.REQD.Y.N:1:1=N,CHARGED.CUSTOMER:1:1=100318,
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,
TOT.REC.CHG.CRCCY:1:1=0,TOT.SND.CHG.CRCCY:1:1=0,
OVERRIDE:1:1=WITHDRAWL.LT.MIN.BAL}WITHDRAWL MAKES A/C BAL
LESS THAN MIN BAL, OVERRIDE:2:1=ACCT.UNAUTH.OD}Unauthorised
overdraft of & & on account&.{EUR}1607353.17}14637{EUR
{1607353.17{14637{100283{213{{

```

I said “more or less” because though all necessary fields are present, we see here several so-called “overrides” (in red font). Let’s try to input this record. To do that we replace “VALIDATE” option with “PROCESS” one:

```

FT,TEST/I/PROCESS,INPUTT/123456,,TRANSACTION.TYPE:=AC,
CREDIT.AMOUNT:=100.00,CREDIT.CURRENCY:=EUR,
DEBIT.ACCT.NO:=14637,CREDIT.ACCT.NO:=10715 

FT10005SVD9T//1,TRANSACTION.TYPE:1:1=AC,
DEBIT.ACCT.NO:1:1=14637, CURRENCY.MKT.DR:1:1=1,
DEBIT.CURRENCY:1:1=EUR, DEBIT.VALUE.DATE:1:1=...

```

Now we have the record FT10005SVD9T with status INAU. This status was set because we had number of authorisations in our *VERSION* set to 1. We haven't set GTS.CONTROL field in our *VERSION* so by default the record was accepted. Help for that field says: "Null: (Reject errors / Approve overrides)". This behaviour can be overridden by setting this field otherwise; we also can override this value in OFS message itself ("2" in blue font means the value of GTS.CONTROL):

```

FT,TEST/I/PROCESS/2,INPUTT/123456,,TRANSACTION.TYPE:=AC,
CREDIT.AMOUNT:=100.00,CREDIT.CURRENCY:=EUR,
DEBIT.ACCT.NO:=14637,CREDIT.ACCT.NO:=10715

FT10005K4V1H//2/NO,HOLD - OVERRIDE WITHDRAWAL MAKES A/C BAL
LESS THAN MIN BAL

```

So the record FT10005K4V1H has status IHL. But what if we'd like to create the record in one step regardless of overrides? We can set the field NO.OF.AUTH in *VERSION*>FUNDS.TRANSFER,TEST to 0 or set it in OFS message:

```

FT,TEST/I/PROCESS/1/0,INPUTT/123456,,TRANSACTION.TYPE:=AC,
CREDIT.AMOUNT:=100.00,CREDIT.CURRENCY:=EUR,
DEBIT.ACCT.NO:=14637,CREDIT.ACCT.NO:=10715

FT100055LJR1//1,TRANSACTION.TYPE:1:1=AC...
...AUTHORISER:1:1=349_INPUTTER_OFS_TEST...

```

## 16 **VERSION routines – AUT.NEW.CONTENT, R.NEW, application insert file**

**T** here are many types of routines (also called “hooks” that can be attached to a *VERSION* to provide additional processing. Let’s try some of them.

Create a routine in *ETC.BP*. Name can be *ANC.TEST*; the following contents:

```
001 SUBROUTINE ANC.TEST
002 $INSERT I_COMMON
003 $INSERT I_EQUATE
004
005 DEBUG
006
007 RETURN
008 END
009
```

Compile it, then attach to user screen *VERSION>FUNDS.TRANSFER,TEST* as *AUT.NEW.CONTENT*. Main purpose of such routine – auto-population of some fields in the record being input.

But before we can do that we have to create a record in *PGM.FILE* application. Type of this record will be “S”:

R10 Model Bank	PROGRAM FILE, INPUT	
PROGRAM	ANC.TEST	
-----		
1	TYPE.....	S
2.	1 GB SCREEN.TITLE	
3	ADDITIONAL.INFO...	
4.	1 BATCH.JOB.....	
5	PRODUCT.....	EB CORE
6	SUB.PRODUCT.....	
7.	1 DESCRIPTION...	
8.	1 APPL.FOR.SUBR.	FUNDS.TRANSFER FUNDS.TRANSFER
9	ACTIVATION.FILE...	
10	MT.KEY.COMPONENT.	
11	MT.KEY.FILE.....	
12	REC.VERIFY.....	
13	BYPASS.SEL.....	
14	BULK.NO.....	
15	JOB.RATING.....	
16	RESERVED.9.....	
-----		

Funny but for AUT.NEW.CONTENT routines you have to create *PGM.FILE* records, for all other “*VERSION*” routines it’s necessary to create a record in application *EB.API*. We’ll see it a bit later.


Let’s attach the routine. We can attach it to any field – we can amend any other field anyway (within reason of course):

```

R10 Model Bank VERSION, INPUT

PGM.NAME.VERSION..      FUNDS.TRANSFER,TEST
-----
49.  1 REKEY.FIELD.NO.
50.  1 AUTOM.FIELD.NO. TRANSACTION.TYPE
51.  1 AUT.OLD.CONTENT
52.  1 AUT.NEW.CONTENT @ANC.TEST
...
-----

```


This is another example of “multi-value block”. Imagine that we want to add another AUT.NEW.CONTENT routine. To expand this block of fields position the cursor at the field 50.1 and type < (less than) . See the result:

```

R10 Model Bank VERSION, INPUT

PGM.NAME.VERSION..      FUNDS.TRANSFER,TEST
-----
49.  1 REKEY.FIELD.NO.
50.  1 AUTOM.FIELD.NO.
51.  1 AUT.OLD.CONTENT
52.  1 AUT.NEW.CONTENT
50.  2 AUTOM.FIELD.NO. TRANSACTION.TYPE
51.  2 AUT.OLD.CONTENT
52.  2 AUT.NEW.CONTENT @ANC.TEST
...
-----

```

To remove this block, position the cursor to any of its fields and type - (hyphen) .

Again we are in *FT* record creation via OFS. Since we put `DEBUG` statement into `ANC.TEST` routine, the processing stops. To populate a field in new record from our routine we need to update global array `R.NEW`. But let’s see it first in debugger (set up your terminal to show at least 600 last lines since `R.NEW` has 500 items in it). Input the following OFS message in `tSS`:

```

FT,TEST/I/PROCESS,INPUTT/123456,,TRANSACTION.TYPE:=AC,
CREDIT.AMOUNT:=100.00,CREDIT.CURRENCY:=EUR,
DEBIT.ACCT.NO:=14637,CREDIT.ACCT.NO:=10715 ↵
DEBUG statement seen

Source changed to c:\temenos\R10\mb10\bnk.run\ETC.BP\ANC.TEST
0005 DEBUG

jBASE debugger->V R.NEW ↵

COMMON variables
R.NEW(0) :
R.NEW(1) : AC
R.NEW(2) : 14637
R.NEW(3) :
R.NEW(4) :
R.NEW(5) :
R.NEW(6) :
R.NEW(7) :
R.NEW(8) :
R.NEW(9) :
R.NEW(10) :
R.NEW(11) : 10715
R.NEW(12) :
R.NEW(13) : EUR
R.NEW(14) : 100.00
R.NEW(15) :
...

```

So if we want to assign something to *FT* field 9 (*DEBIT.THEIR.REF*) we have to assign a value to *R.NEW(9)*. But before we do that – think: what if field numbers will change in later releases of T24? Will we have to go through all local code (and our routine belongs to local code – i.e. all code that is written outside the core) to see if field numbers are still in sync?

The answer is: no. To make things compatible with future releases we'll use another “insert“ file – this time it's one where all fields of our application are defined. The name of such file consists of “*I.F.*” plus application name:



```
001 SUBROUTINE ANC.TEST
002 $INSERT I_COMMON
003 $INSERT I_EQUATE
004 $INSERT I_F.FUNDS.TRANSFER
005
006 DEBUG
007
008 R.NEW(FT.DEBIT.THEIR.REF) = 'OUR REFERENCE'
009
010 RETURN
011 END
012
```

In `I_F.FUNDS.TRANSFER` `FT.DEBIT.THEIR.REF` is defined as 9. This allows us to move to a new release with only recompiling all local source. Now compile the source and launch `tSS` again. At debugger prompt type `W` to see where we are. If the source that you see is an older one – type `P ETC.BP` and then again `W`:

```

FT,TEST/I/PROCESS,INPUTT/123456,,TRANSACTION.TYPE:=AC,
CREDIT.AMOUNT:=100.00,CREDIT.CURRENCY:=EUR,
DEBIT.ACCT.NO:=14637,CREDIT.ACCT.NO:=10715 
DEBUG statement seen

Source changed to c:\temenos\R10\mb10\bnk.run\ETC.BP\ANC.TEST
0005 DEBUG
jBASE debugger->W 
0002 $INSERT I.COMMON
0003 $INSERT I.EQUATE
0004
0005 DEBUG
0006
0007 RETURN
0008 END
0009
jBASE debugger->P ETC.BP 
Source path: ETC.BP
jBASE debugger->W 
0002 $INSERT I.COMMON
0003 $INSERT I.EQUATE
0004 $INSERT I.F.FUNDS.TRANSFER
0005
0006 DEBUG
0007
0008 R.NEW(FT.DEBIT.THEIR.REF) = 'OUR REFERENCE'
0009
0010 RETURN

```

Type S  2 times to proceed with 2 program steps. Then see what's there in R.NEW(9):

```


jBASE debugger->V R.NEW(9) 
COMMON variables
R.NEW(9) : OUR REFERENCE

```

Then – let it continue (C ) and see the result:


```
FT10005Z96PZ//1,TRANSACTION.TYPE:1:1=AC...
DEBIT.THEIR.REF:1:1=OUR REFERENCE,...
```

## 17 OFS.REQUEST.DETAIL

o log all OFS activities we can use application *OFS.REQUEST.DETAIL*. All setup that is necessary is to populate 2 additional fields in our *OFS.SOURCE* record:

```
R10 Model Bank      OFS SOURCE, INPUT
SOURCE.NAME.....  TEST
-----
...
16 MAINT.MSG.DETS... Y
17 DET.PREFIX.....  TEST
-----
```

Now feed OFS string to tSS again and see the protocol:

```
jsh mb10 ~-->LIST F.OFS.REQUEST.DETAIL LIKE TEST... 
@ID.....          TEST100050550329057.01
@ID.....          TEST100050550329057.01
MESSAGE.KEY.....  TEST100050550329057.01
APPLICATION.....  FT
VERSION.....      TEST
FUNCTION.....      I
TRANS.REFERENCE.  FT100054MYD8
...
```

```

USER.NAME..... INPUTT
COMPANY..... GB0010001
DATE.TIME.RECD.. 08:04:17:260 22 OCT 2010
DATE.TIME.QUEUE.
DATE.TIME.PROC.. 08:04:23:541 22 OCT 2010
STATUS..... PROCESSED
MSG.IN..... FT,TEST/I/PROCESS,INPUTT/*****,,
                TRANSACTION.TYPE:=AC,CREDIT.AMOUNT:=100.00,
                CREDIT.CURRENCY:=EUR,DEBIT.ACCT.NO:=14637,
                CREDIT.ACCT.NO:=10715

MSG.OUT..... FT100054MYD8/TEST100050550329057.01/1,
                TRANSACTION.TYPE:1:1=AC,DEBIT.ACCT.NO:1:1=14637,
                CURRENCY.MKT.DR:1:1=1,DEBIT.CURRENCY:1:1=EUR,

                DEBIT.VALUE.DATE:1:1=20100105,
                DEBIT.THEIR.REF:1:1=OUR REFERENCE,
                CREDIT.ACCT.NO:1:1=10715,CURRENCY.MKT.CR:1:1=1,
                CREDIT.CURRENCY:1:1=EUR,CREDIT.AMOUNT:1:1=100.00,
                CREDIT.VALUE.DATE:1:1=20100105,
                PROCESSING.DATE:1:1=20100105,
                CHARGE.COM.DISPLAY:1:1=NO,

...
                OVERRIDE:1:1=WITHDRAWL.LT.MIN.BAL}WITHDRAWL MAKES
                A/C BAL LESS THAN MIN BAL,OVERRIDE:2:1=
                ACCT.UNAUTH.OD}Unauthorised overdraft of & &
                on account&.{EUR}1607753.17}14637{EUR{1607753.17
                {14637{100283{213{{,RECORD.STATUS:1:1=INAU,
                CURR.NO:1:1=1,INPUTTER:1:1=5503_INPUTTER_OFS_TEST,
                DATE.TIME:1:1=1007220804,CO.CODE:1:1=GB0010001,
                DEPT.CODE:1:1=1
                ACTION.....
                GTS.CONTROL.....
                NO.OF.AUTH.....

```

## 18 Manual transaction input in comparison with OFS, GTSACTIVE variable

**L**et's try to input *FT* record manually using the same *VERSION*. But before that comment the *DEBUG* statement in routine *ANC.TEST* (don't forget to recompile routine after that):

```
001 SUBROUTINE ANC.TEST
002 $INSERT I_COMMON
003 $INSERT I_EQUATE
004 $INSERT I_F.FUNDS.TRANSFER
005
006 *DEBUG
007
008 R.NEW(FT.DEBIT.THEIR.REF) = 'OUR REFERENCE'
009
010 RETURN
011 END
012
```

Now log in to T24, type *FT,TEST I F3* at *AWAITING APPLICATION* prompt – yes, “F3” at command end is another way to get ID automatically:

```
R10 Model Bank FUNDS.TRANSFER,TEST INPUT REF FT10005000F3
```

```
-----  
1 TRANSACTION.TYPE..  
2 DEBIT.ACCT.NO.....  
3 IN.DEBIT.ACCT.NO..  
4 CURRENCY.MKT.DR... 1      Currency Market  
5 DEBIT.CURRENCY....  
6 DEBIT.AMOUNT.....  
7 DEBIT.VALUE.DATE..  
8 IN.DEBIT.VDATE....  
9 DEBIT.THEIR.REF... OUR REFERENCE  
10 CREDIT.THEIR.REF..  
11 CREDIT.ACCT.NO....  
12 CURRENCY.MKT.CR... 1      Currency Market  
13 CREDIT.CURRENCY...  
14 CREDIT.AMOUNT.....  
15 CREDIT.VALUE.DATE..  
16 TREASURY.RATE.....  
-----
```

What we can see here?

- There are core defaults visible (fields 4 and 12) – again business logic.
- Our local default in field 9 is here so OFS processing is basically the same as manual input.

How our routine knows if we're using OFS or manual input? We can analyse core variable `GTSACTIVE` that is set in `I_GTS.COMMON`:

```

001 SUBROUTINE ANC.TEST
002 $INSERT I.COMMON
003 $INSERT I.EQUATE
004 $INSERT I.F.FUNDS.TRANSFER
005 $INSERT I.GTS.COMMON
006
007 *DEBUG
008
009 R.NEW(FT.DEBIT.THEIR.REF) = 'OUR REFERENCE'
010
011 IF GTSACTIVE THEN
012     R.NEW(FT.CREDIT.THEIR.REF) = 'OFS INPUT'
013 END ELSE
014     R.NEW(FT.CREDIT.THEIR.REF) = 'MANUAL INPUT'
015 END
016
017 RETURN
018 END
019

```


Compile, input 2 transactions – manually and using OFS, check results:

```

R10 Model Bank FUNDS.TRANSFER,TEST INPUT FT10005MFRL3
-----
...
9 DEBIT.THEIR.REF.. OUR REFERENCE
10 CREDIT.THEIR.REF. MANUAL INPUT
...
-----

```

```

jsh mb10 ~-->LIST FBNK.FUNDS.TRANSFER$NAU
'FT100059MKYC' CREDIT.THEIR.REF 

@ID..... CREDIT.THEIR.REF.....


                FT100059MKYC  OFS INPUT

1 Records Listed

```

Why ID is right-justified in the output and CREDIT.THEIR.REF is left-justified? The answer is in dictionary:

```

jsh mb10 ~-->CT DICT FBNK.FUNDS.TRANSFER$NAU @ID
CREDIT.THEIR.REF 

@ID
001 D
002 0
003
004 @ID
005 25R
006 S

CREDIT.THEIR.REF
001 D
002 10
003
004 CREDIT.THEIR.REF
005 27L
006 S
007
...
```

And dictionary is built on the base of *STANDARD.SELECTION* record which we've examined earlier. The problem for a person who is familiar with MV (multivalue) world but is new to T24 is that normally you can add your dictionary entries (e.g. I-descriptors; see later what's that) manually



to dictionary file. In T24 you add such things to *STANDARD.SELECTION* record and then – upon its authorisation (depending of your changes or if the field *REBUILD.SYS.FIELDS* is set to Y) – dictionary is rebuilt (erasing all things added manually). Remember about that.

## 19 Browser client – jboss, jBASE agent, logging in

**P**robably it's time to start using this beast. To access T24 via browser you'll need (except browser itself – IE or Firefox) to start **jboss** application server and so-called “jBASE agent”. I've written at the very start of this book that in your R10 package you probably already have a directory called something like BATfiles. Though you might not.

If you got **jboss** not from Temenos but simply downloaded it from the Internet then you'll need to put there so-called “TOCF-EE” package. I'd rather restrain from more explanations – things change all the time and you better see the latest setup document provided by Temenos.

Then see file `t24-ds.xml` in **jboss** deploy directory which in our case is: `c:\temenos\R10\jboss\server\default\deploy` (yes we again can use CT or JED to see or edit it; rule is the same – directory is a “table”, file is a “record”):

```

jsh mb10 ~-->CT
c:\temenos\R10\jboss\server\default\deploy t24-ds.xml
↩
...
038 <jndi-name>jca/t24ConnectionFactoryR10</jndi-name>
...
043 <config-property name="host"
type="java.lang.String">127.0.0.1</config-property>
044 <config-property name="port"
type="int">20001</config-property>
045
046 <config-property name="allowInput"
type="java.lang.Boolean">true</config-property>
047 <config-property name="env" type="java.lang.String">
OFS_SOURCE=BROWSERTC</config-property>

```

OK, here we are. To connect to T24 we'll need to start both **jboss** and **jBASE** agent with the latter running on a local host using port 20001. Agent will use *OFS.SOURCE* record **BROWSERTC** to connect to T24; while "allowInput" = "true" will let us debug our routines if needs be.

But let's first see if that *OFS.SOURCE* record is OK. It has to have type "SESSION" and syntax type "OFS":

```

jsh mb10 ~-->LIST F.OFS.SOURCE 'BROWSERTC' SOURCE.TYPE
SYNTAX.TYPE ↩

```

@ID.....	SOURCE.TYPE	SYNTAX.TYPE
BROWSERTC	SESSION	OFS


```

1 Records Listed

```

Type "SESSION" also allows us to connect to it using **tSS**:

```

jsh mb10 ~-->tSS BROWSERTC 
<tSS version="1.1"><t24version>R10.000</t24version><t24pid>
2404</t24pid><t24ofssource>BROWSERTC</t24ofssource>
<clientIP/></tSS>

ENQUIRY.SELECT,,INPUTT/123456,%SPF

,@ID::No of /RUN.DATE::Run date/SITE.NAME::Site
name/OP.MODE::Op mode/OP.CONSOLE::Op console,"SYSTEM" "20
JAN 2010" "R10 Model Bank " " 0" " "

```

For the second time we use *SPF* application without yet knowing what it is. Answer – this is the main T24 setup table. And there is always only one record (SYSTEM) so we could use it in “EVAL” exercise described above.

Now start jBASE agent using a bat file you already have (or you can use `remote.cmd`, copy it to `agent.cmd` and amend the last line):

```

rem jprofile.bat
%JBCRELEASEDIR%\bin\jsh - -c "jbase_agent -p 20001"

```

Note port number (20001) that was set up in `t24-ds.xml`. Start agent. Then start `jboss` (actually the order doesn’t matter). The following batch file will do:

```

set JAVA_HOME=c:\temenos\jdk1.6.0.17
cd c:\temenos\R10\jboss\bin
run.bat -b 0.0.0.0

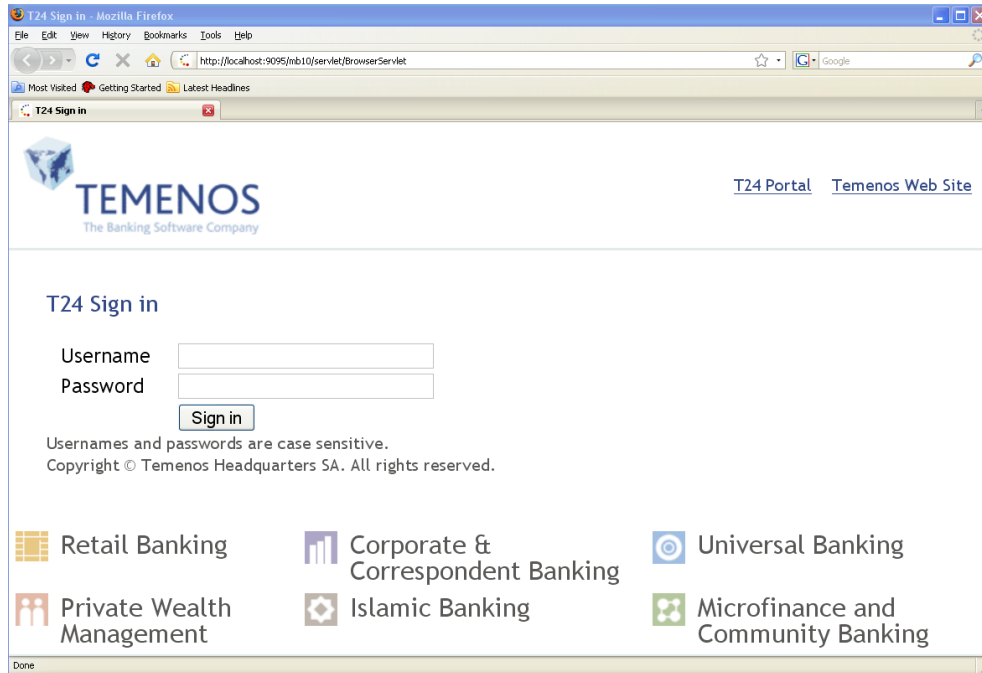
```

Now go to browser and try to connect to:

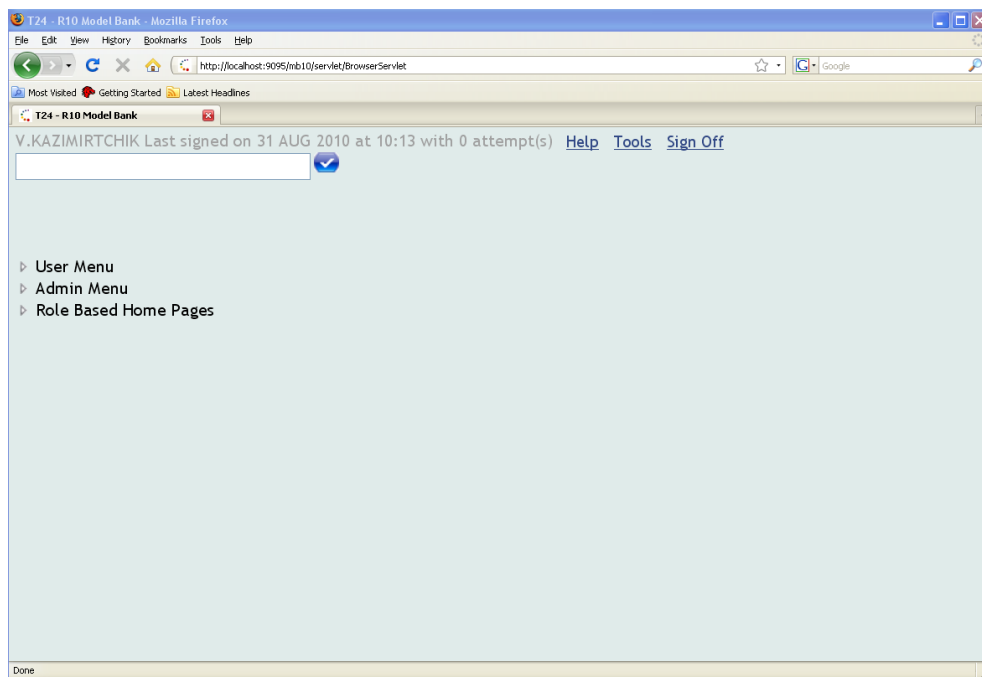
`http://localhost:9095/BrowserWeb`

I usually rename `BrowserWeb.war` file (it can either come as a directory with the same name) in `jboss` deploy directory to something more relevant (e.g. `mb10.war`) so the following screen will have different URL (note that

“/servlet/BrowserServlet” at the end is added automatically so you needn’t type it):




Input your user name and password... we're there!



We have several jBASE sessions started now. See WHERE command output:

```

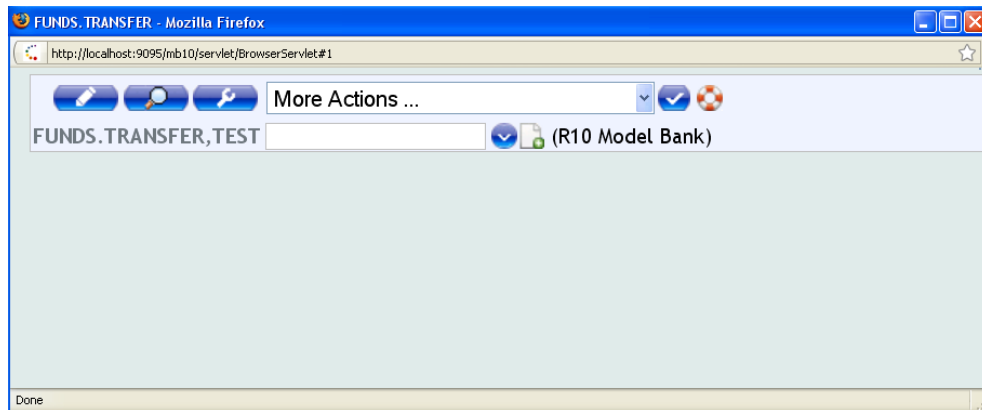
jsh mb10 ~-->WHERE 
Port   Device  Account      PID    Command
*1     VT220   telnet       2404   c:\temenos\R10\T AFC\bin\jsh -
                                   WHERE
  2     ntcon   vkazimirtchi 2872   c:\temenos\R10\T AFC\bin\jsh - -c jbase_agent -p 20001
  4     ntcon   vkazimirtchi 2768   jbase_agent

```

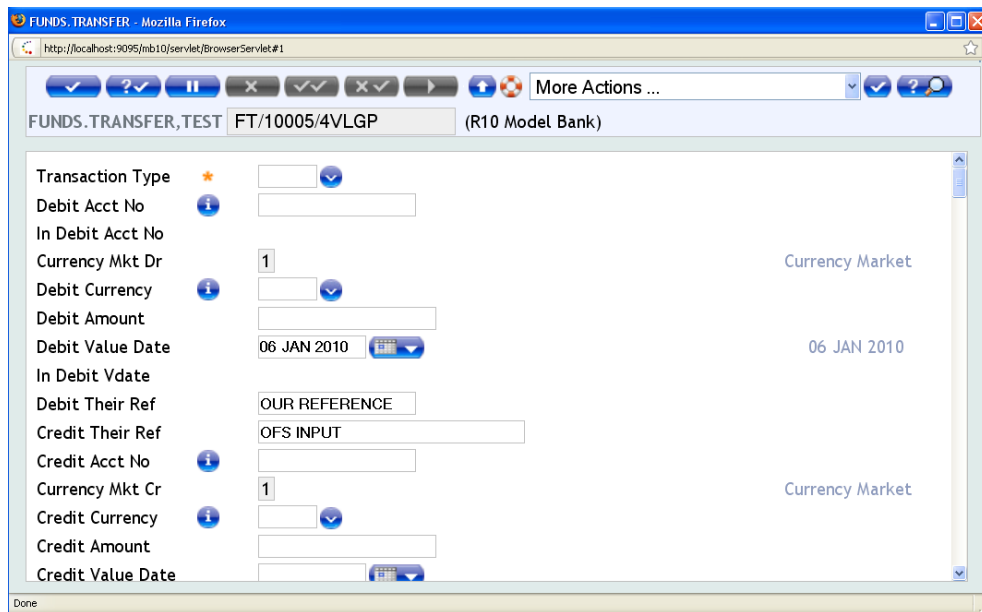
Here we see jBASE agent on the port 20001 and another agent instance that was forked when we logged in using browser.

## 20 Transaction input under Browser, debugging

**L**et's use our *VERSION* to input a new *FT* record. Input a command *FT,TEST* at browser client command line.



To input a new record click to the icon which is located to the left of “R10 Model Bank” words.



What we can see here?

- AUT.NEW.CONTENT routine defaulted the field DEBIT.THEIR.REF the same way as it did before.
- Routine thinks it's an OFS input.

Actually, browser client interacts with T24 via OFS (using *OFS.SOURCE* record *BROWSERTC*, as we saw above). To make our routine more intelligent we amend it the following way:

```

001 SUBROUTINE ANC.TEST
002 $INSERT I.COMMON
003 $INSERT I.EQUATE
004 $INSERT I.F.FUNDS.TRANSFER
005 $INSERT I.GTS.COMMON
006 $INSERT I.F.OFS.SOURCE
007
008 *DEBUG
009
010 R.NEW(FT.DEBIT.THEIR.REF) = 'OUR REFERENCE'
011
012 IF GTSACTIVE THEN
013     IF OFS$SOURCE.REC<OFS.SRC.SOURCE.TYPE> EQ 'SESSION'
014     THEN
015         R.NEW(FT.CREDIT.THEIR.REF) = 'INPUT VIA BROWSER'
016     END ELSE
017         R.NEW(FT.CREDIT.THEIR.REF) = 'OFS INPUT'
018     END
019 END ELSE
020 R.NEW(FT.CREDIT.THEIR.REF) = 'INPUT VIA TERMINAL'
021 END
022 RETURN
023 END
024

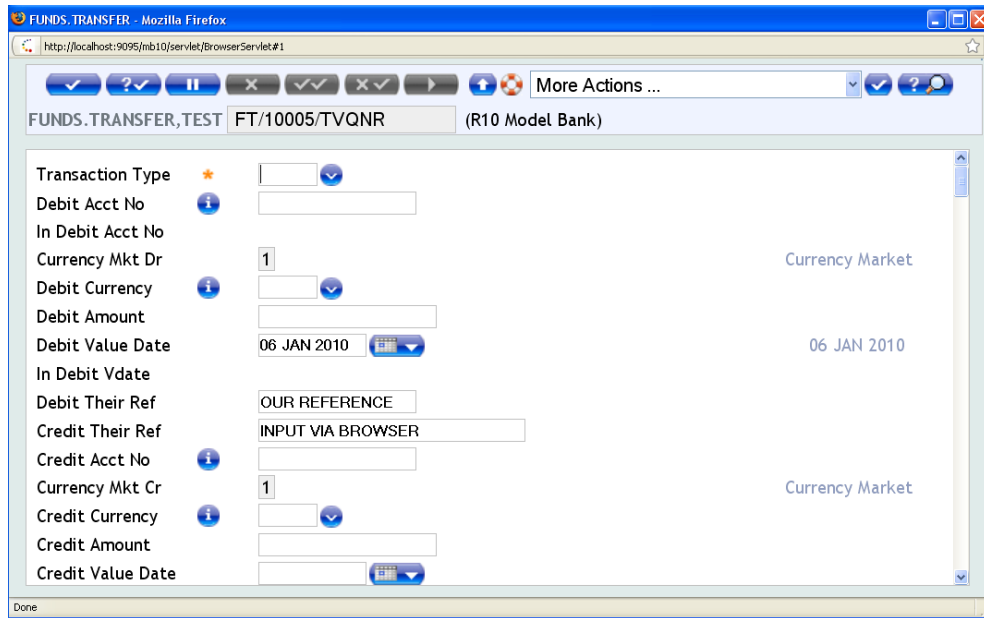
```

You see that we don't read a record from application *OFS.SOURCE*. We already have it read and put into global variable *OFS\$SOURCE.REC*. Use this technique wherever possible to reduce I/O.

After compilation of amended routine it might happen that system behaviour hasn't changed. It's because the executable code is cached, so log out and log in again. Sometimes you'll have even to restart jBASE agent to see the changes since a forked process (which we saw earlier using *WHERE* command) is reused (so we'll use one that has old program in cache). In case of restarting the agent it's not necessary to log in again.

See the changed logic in action:





To debug a routine simply put `DEBUG` statement into it (we can uncomment the one we had commented earlier). When it's being reached you'll see debugger screen in jboss window:

```

16:27:04,806 INFO [STDOUT] (OFS.INITIALISE.SOURCE) :
BROWSERTC
16:27:05,181 INFO [STDOUT] Total processing time from start
to end of OFS.SESSION.MANAGER *** Time = 359.36108397
16:27:05,338 WARN [Parameters] Parameters: Invalid chunk
ignored.
16:27:05,353 WARN [Parameters] Parameters: Invalid chunk
ignored.
16:27:06,338 INFO [STDOUT] (OFS.INITIALISE.SOURCE) :
BROWSERTC
16:27:06,400 INFO [STDOUT] Total processing time from start
to end of OFS.SESSION.MANAGER *** Time = 46.8730468696064
16:27:07,994 INFO [STDOUT] Total processing time from start
to end of OFS.SESSION.MANAGER *** Time = 2593.65039063
16:27:25,915 INFO [STDOUT] Total processing time from start
to end of OFS.SESSION.MANAGER *** Time = 234.3659668
16:27:28,196 INFO [STDOUT] DEBUG statement seen
16:27:28,196 INFO [STDOUT] Source changed to
c:\temenos\R10\mb10\bnk.run\ETC.BP\ANC.TEST
16:27:28,196 INFO [STDOUT] 0008
16:27:28,196 INFO [STDOUT] DEBUG
16:27:28,196 INFO [STDOUT] jBASE debugger->

```

After you finish your debugging and use its “C” command to continue execution, the application screen will appear (provided that you haven’t exceeded timeouts which in this case probably are to be increased).

## 21 TODAY variable, date format in T24, edit mode in Classic, API for dates manipulation

**L**et’s expand a little the functionality of our AUT.NEW.CONTENT routine. There is a field DEBIT.VALUE.DATE that we’d like to auto-populate with current bank date. How to do that? System date will not do because the

date of current bank day might be different.

Answer: there is a variable called TODAY in I\_COMMON insert file. So we can include the following piece of code to our routine:

```
021
022 R.NEW(FT.DEBIT.VALUE.DATE) = TODAY
023
```

Let's go back to terminal client (the code works the same way – you saw it; we'll use Browser client only for Browser-specific issues). Input the deal manually:

```
R10 Model Bank FUNDS.TRANSFER,TEST INPUT REF FT10005C7BOK








-----
1 TRANSACTION.TYPE..
2 DEBIT.ACCT.NO....
3 IN.DEBIT.ACCT.NO..
4 CURRENCY.MKT.DR... 1           Currency Market
5 DEBIT.CURRENCY...
6 DEBIT.AMOUNT.....
7 DEBIT.VALUE.DATE.. 05 JAN 2010
8 IN.DEBIT.VDATE...
9 DEBIT.THEIR.REF... OUR REFERENCE
10 CREDIT.THEIR.REF.. INPUT VIA TERMINAL
11 CREDIT.ACCT.NO....
12 CURRENCY.MKT.CR.. 1           Currency Market
13 CREDIT.CURRENCY...
14 CREDIT.AMOUNT.....
15 CREDIT.VALUE.DATE.
16 TREASURY.RATE....
-----
```

Is today's date stored as "05 JAN 2010" ? We could use debugger to see it but the other way is: go to the field DEBIT.VALUE.DATE using **F3** several times and – when you're on that field – press **F7** (or **Ctrl T** **←** if your keys

are still not mapped):

```
7 DEBIT.VALUE.DATE.. 20100105 |
```

Here we are at “Edit” mode which represents the true format of data. “05 JAN 2010” is its external representation. In this mode functional keys also work:

-  – exit.
-  – move one character backward.
-  – move one character forward.
-  – move to the end.
-  – delete a character.
-  – toggle insert/overwrite mode.
-  – commit changes and exit.

What if we wanted to put to our field not the current bank day but the next one? Firstly, it’s evident that we can’t add 1 to the value of TODAY, otherwise we might end up with, say, 32<sup>nd</sup> of July (I swear I saw that in local developments). Secondly, we need to know if tomorrow is a work day or either a weekend or a holiday.

Public API subroutine CDT (which is described in “Subroutine Guide.pdf”) can be used to calculate the necessary date for us:

```
021
022 V.DATE = TODAY
023 CALL CDT(' ', V.DATE, '+1W')
024
025 R.NEW(FT.DEBIT.VALUE.DATE) = V.DATE
026
```

Where “+1W” means “forward 1 working day”. For more options see the manual mentioned above; answer for the evident question “where weekends and holidays are stored” is “application *HOLIDAY*”.

But why didn’t we call `CDT` with `TODAY` as a parameter? Because the second parameter for `CDT` is both an input and output one. It changes after that call. So `TODAY` would be changed as well resulting in total mess in everything that this particular session might have done after that. Even after logging out since global variables are preserved until the session is over.

After you’ve checked the changes and have seen “06 JAN 2010” in the field `DEBIT.VALUE.DATE` of a new deal let’s see how global variables do behave.

## 22 Global variables again – their lifetime, writing a PROGRAM, CRT

**N**ow let’s write a program (since we’re going to run it from `jsh`). The code is very simple:

```
001 PROGRAM PROG.TEST
002 $INSERT I_COMMON
003 $INSERT I_EQUATE
004
005 CRT TODAY
006
007 STOP
008 END
009
```


(Note “STOP” instead of “RETURN”.)


Of course before writing it check if its name is available (`jshow -c`). Compile it (again ignore “failed” messages, the one to look at is: “Object PROG.TEST cataloged successfully”); but if you have any doubts – run `jshow` again – and ignore “DUP” warning).

During the compilation beware of messages like:

```
Warning: Variable TODAY is never assigned!
```

In our case this could happen if we’d forgotten to include `I_COMMON` in our source. In that case variable `TODAY` would be addressed without being assigned. Such errors are also quite common when you get a typo in variable name.

OK, open a fresh telnet session and at `jsh` type `PROG.TEST` . Then log in to T24, log out and run `PROG.TEST` again. See the difference?

On the first run `TODAY` variable is not initialised so the output is “0”. On the second it is so the output is “20100105”. And sometimes it hurts... Don’t be surprised if you test some OFS with `tss`, then try to log in to T24 and see the system in an infinite loop... In this case press  C and – when you’re at debugger prompt – quit it, then close the session.

About “CRT” – you’ll never see its output under Browser, so you might be surprised how many unexpected screen output happens in T24 running

in Classic mode. Sometimes it helps to find the reason of an error that you were desperately seeking under Browser.

## 23 CHECK.REC.RTN – error raising, other VERSION routines – notes

**I**t's always a good idea to use a hook routine for the purpose for which it was designed. For example, to stop the processing before a record can be accessed (or a user can see a record) it's no use to try to raise an error in an AUT.NEW.CONTENT routine – for this purpose it's better to use routine of type CHECK.REC.RTN. Let's see an example of a routine that forbids the user to enter *FT* record that wasn't input by this user. Here we'll start to comment our code – the purpose of the routine in the header and also some comments in the code:

```
001 SUBROUTINE CHKREC.TEST
002 *-----
003 * This routine allows either to input a new record
004 * or enter a record which belongs to the same user.
005 * Type of routine: CHECK.REC.ROUTINE.
006 *-----
007 $INSERT I_COMMON
008 $INSERT I_EQUATE
009 $INSERT I_F_FUNDS.TRANSFER
010
011 V.INPT = R.NEW(FT.INPUTTER)
012 IF V.INPT EQ '' THEN RETURN ;* new record
013
014 IF FIELD(V.INPT, '_ ', 2) NE OPERATOR THEN
015     E = 'OPERATOR NOT THE SAME'
016 END
017
018 RETURN
019 END
020
```

Before we can attach this routine to our *VERSION*, we have to create *EB.API* record for it:

```
R10 Model Bank EB.API, INPUT

  KEY..... CHKREC.TEST
-----
1 DESCRIPTION.....
2 PROTECTION.LEVEL.. FULL
3 SOURCE.TYPE..... BASIC
4 JAVA.METHOD.....
5 JAVA.CLASS.....
6 JAVA.PA
7 RESERVED27.....
8 RESERVED26.....
9 RESERVED25.....
10 RESERVED24.....
11 RESERVED23.....
12 RESERVED22.....
13 RESERVED21.....
14 RESERVED20.....
15 RESERVED19.....
16 RESERVED18.....
-----
```

Then attach our routine to our *VERSION* (field *CHECK.REC.RTN*; it's a multivalued field so we can have several such routines if needs be). Run *FT,TEST* at *AWAITING APPLICATION* prompt and try to:

- Input a new record.
- Enter a record that you've created before (for authorised records you can only use function *S*).
- Enter a record that somebody else created.

It's not that I recommend to use such technique for access control; there's whole T24 module called *SMS* available for that purpose. Most such things can be achieved without programming, though it's still required in some



cases.

It's time to mention that not all routines are triggered in a particular situation. For example, `S` function doesn't trigger `AUT.NEW.CONTENT` routines. (Actually, it doesn't make sense to auto-populate a record that you're opening in "See" mode.) `VALIDATION.RTN` is triggered twice – when user inputs a value into a field (that field is to be assigned as "`HOT.FIELD`" or "`HOT.VALIDATE`" for it to work in Browser) and before system cross-validation at commit time.

Also keep in mind that to raise an error you'll have to apply different methods. We saw how to raise an error in `CHECK.REC.RTN`; to do it in a "`VALIDATION`" routine you need to assign global variable `ETEXT`. In "`INPUT`" routine the following code is necessary:

```
AF = FT.DEBIT.THEIR.REF ;* number of field where this error
                        ;* will appear
ETEXT = 'THIS IS THE ERROR'
CALL STORE.END.ERROR
```

You can find more about *VERSION* routines in helptext (unfortunately, T24 manuals still describe only 4 earliest types of them – `AUT.NEW.CONTENT`, `VALIDATION.RTN`, `INPUT.ROUTINE` and `AUTH.ROUTINE`. The only thing that is left to say here about `VALIDATION.RTN` is that it's not recommended nowadays since when it's triggered after field input, under Browser the whole record is to be transferred to the server and it's considered not so good for overall system performance.

## 24 Programming language overview, writing a simple game

**A**s we saw, there is a programming language that is supplied with jBASE. This language was earlier called "jBASE Basic" but then was renamed to

“jBC” (possibly from marketing point of view). But I still call it “Basic”.

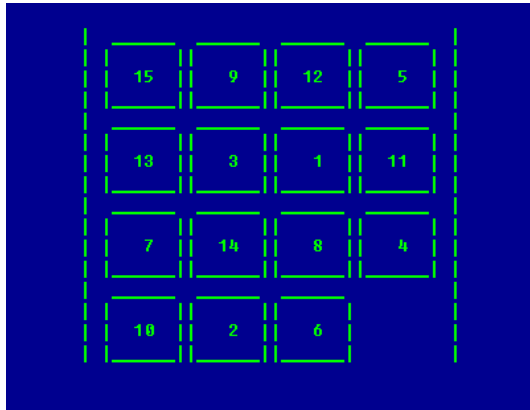
It looks quite simple and readable and here is the dangerous part. Typical boss usually thinks that it’s sufficient to hire anyone with Basic knowledge and that’s it. Don’t think that it’s simple to program for T24. The truth is that T24 local developer (or supporter) uses only tiny part of the language and the most steep part of learning curve is to understand API, core global variables, hook types and many other things. Real knowledge starts to come only after about 6 months of real work.

What T24 local developer actually doesn’t need:

- Screen I/O (see CRT note above).
- File I/O as it described in “jBASE BASIC.pdf” – only wrappers like F.READ etc.
- File creation, clearing etc – all done via wrappers again.
- Writing a PROGRAM, so no STOP, ABORT etc. The only place where PROGRAM can still exist is *PRINTER.ID* hook routine (though last time I’ve tried it was around R04 times).
- Many other things.

But to understand the language a little better it could be a good exercise to write something that uses some these things. Let this exercise be a game. Am I kidding? No. A classic “fifteen” game using text mode of a terminal session. It was written by me a long time ago and still works perfectly being compiled under R10.

The screen of this game looks like:



Besides things that as a T24 person you wouldn't need we'll learn some useful stuff as well: arrays, for example.

Let's see how to do that. Here the start of this program:

```
001 PROGRAM FIFTEEN
002 *=====
003 * BY V.KAZIMIRCHIK (KZM), 200712
004 * 15 Puzzle.
005 * Feel free to distribute provided that you fully
006 * retain this header.
007 * Author bears no responsibility if you run it during
008 * work hours :-))
009 * ENJOY!
010 *=====
```

Here we have typical header that I recommend to use. The more comments, the better.

```
011 GOSUB INIT
012
```

It's a good idea to have subsections rather than top-down code. Here's section INIT at the very bottom of our program:

```

149 *-----
150 INIT:
151 ECHO OFF
152
153 CRT @(-1)
154
155 * Generate the field
156
157 V.BOARD = ''
158 V.FINI = ''
159
160 FOR V.I = 1 TO 15
161
162     LOOP
163         V.NUMBER = RND(15) + 1
164         FIND V.NUMBER IN V.BOARD SETTING V.DUMMY ELSE
165             V.BOARD<-1> = V.NUMBER
166             BREAK
167     END
168 REPEAT
169
170     V.FINI<V.I> = V.I ;* solved array for comparison
171
172 NEXT V.I
173
174 V.BOARD<16> = 0
175 V.FINI<16> = 0
176
177 GOSUB DRAW.FIELD
178
179 RETURN
180
181 END

```

Note:

- “V.” prefix used for variables (even for a loop counter) – it’s a good idea to have some standard prefix which saves a developer from occasionally choosing a name which is used for a global T24 variable (and the latter can

be even so simple as “A”, “C”, “E” – see `I.COMMON`). Some people use “LOC.” being influenced by my very old document (and I probably was influenced by somebody else writing it) but nowadays I find it too long. Some people use “Y.”... It’s up to you.

- Variables `V.BOARD` and `V.FINI` which are used to store so-called “dynamic array”. Actually dynamic array is a string with delimiters – `@FM`, `@VM` and `@SM` which I hope you already know. Even in `I.COMMON` they are declared as `FM`, `VM` and `SM` accordingly (and assigned in `I.EQUATE`). So in `T24` subroutine you can use just “FM” while in a standalone program you’ll have to prefix it with “@”.

- `CRT` usage to clear the screen, `FOR . . . NEXT` loop, `LOOP . . . REPEAT` loop.
- `RND()` function to get a random number in the range from 1 to 15.
- `FIND` statement to find a value in dynamic array.

Back to dynamic arrays. I wouldn’t go into a discussion when to use dynamic array and when to use a dimensioned one (we saw a dimensioned array earlier – it’s `R.NEW`). In the code we see that dynamic array elements are addressed using angle brackets (“-1” means “the new element”).

Another `GOSUB` here. Where it leads us?

```

120 *=====
121 DRAW.FIELD:
122
123 GOSUB CLEAR.FIELD
124 GOSUB DRAW.BORDER
125
126 FOR V.PIECE = 1 TO 16
127     GOSUB DRAW.PIECE
128 NEXT V.PIECE
129
130 CRT @(25,0):SYSTEM(40):@(-4):
131 CRT @(8,22):
132
133 RETURN

```

OK, what's new here? Some positioned screen output using CRT, 3 more GOSUBs. See them one by one:

```

134
135 *=====
136 CLEAR.FIELD:
137
138 FOR V.I = 2 TO 19
139
140     IF V.I NE 3 THEN
141         CRT @(0, V.I):
142         CRT @(-4):
143     END
144
145 NEXT V.I
146
147 RETURN
148

```

```

087 *=====
088 DRAW.BORDER:
089
090 FOR V.I = 4 TO 19
091     CRT @(22,V.I):'|':
092     CRT @(57,V.I):'|':
093 NEXT V.I
094
095 RETURN
096

```

```

097 *=====
098 DRAW.PIECE:
099
100 V.X = MOD(V.PIECE+3, 4) * 8 + 24
101 V.Y = (INT((V.PIECE-1) / 4) + 1) * 4
102
103 V.NUMBER = V.BOARD<V.PIECE>
104 IF V.NUMBER EQ 0 THEN
105     CRT @(V.X+1,V.Y):'_____' :
106     CRT @(V.X,V.Y+1):'_____' :
107     CRT @(V.X,V.Y+2):'_____' :
108     CRT @(V.X,V.Y+3):'_____' :
109 END ELSE
110     CRT @(V.X+1,V.Y):'_____' :
111     CRT @(V.X,V.Y+1):'|_____|' :
112     CRT @(V.X,V.Y+2):'|__':FMT(V.NUMBER,"2R"): '__|' :
113     CRT @(V.X,V.Y+3):'|_____|' :
114 END
115
116 CRT @(8,22):
117
118 RETURN
119

```

I've made spaces visible intentionally in lines 105-108, 111 and 112.

By the way, it's a common mistake even experienced guys sometimes do

– if you forget a `RETURN` at the end of a section, execution continues to the next one, providing unexpected program flow.

At last we've returned to the main section. Let's continue:

```
013 LOOP
014
015     IF V.BOARD EQ V.FINI THEN
016         V.TEXT = 'YOU WIN'
017         PRINT V.TEXT
018         INPUT V.DUMMY
019         BREAK
020     END
021
```

The main loop starts here and ends almost where main section does. Note `BREAK` statement to exit from the main loop when the puzzle is solved. We saw earlier that dynamic array `V.BOARD` contains tiles' numbers in their shuffled state (initiated using random number function) and array `V.FINI` contains numbers in the sorted (i.e. solved) mode. Since dynamic arrays are strings they can be simply compared using `'EQ'`. Instead of `'EQ'` symbol `'='` could be used.

Again, I'm not pressing on you but for me it's tidier to use `'='` for assignment and `'EQ'`, `'LT'`, `'GE'` etc for comparison. Again, it's up to you.

Have you noticed text formatting with empty lines between blocks of code and offsets inside `IFs`, `WHILEs` etc? Readability of code is crucial. Even if you return back to your own code in one year time, the more this code is readable, the better. In the examples here I don't show the leftmost code offset of 6 characters that I'm accustomed to (to save some space). Automatic offsets can be achieved by command `"BI"` in `JED` editor, though defaults are not my favourite and I set it up with an option to `JED`:

```
jsh mb10 ~-->JED ETC.BP FIFTEEN (B3,2)
```



Wher have we stopped?

```
022 V.KEY = KEYIN()
023
024 BEGIN CASE
025
```

Here we wait for keyboard input and start a CASE processing the key that was pressed.

```
026 CASE V.KEY EQ CHAR(27) ;* Esc
027
028     V.KEY.2 = KEYIN() ;* ]
029     V.KEY.3 = KEYIN()
030
031     FIND 0 IN V.BOARD SETTING V.ZERO.POSN ELSE
032         TEXT = 'FATAL ERROR 1'
033         PRINT TEXT
034         BREAK
035     END
036
037     BEGIN CASE
```

If it was ESC then most probably it's a starting point to the combination of symbols that is resulted from pressing one of arrow keys. Without much of an analysis we take the second character (which in this case is “[”) and then the third one that tells us if one key of that four which are of interest to us was pressed.

Then we look where do we have an empty square at the puzzle. (With handling the situation when we don't have one – quite unlikely but who knows?)

And only then we're ready to proceed all arrow keys (of course if this is really the case):

```

038 CASE V.KEY.3 EQ 'C' AND MOD(V.ZERO.POSN, 4) NE 1 ;*
right
039
040     V.POSN.ZERO.NEW = V.ZERO.POSN - 1
041     GOSUB SWAP.PIECES
042
043 CASE V.KEY.3 EQ 'B' AND V.ZERO.POSN GE 5 ;* down
044
045     V.POSN.ZERO.NEW = V.ZERO.POSN - 4
046     GOSUB SWAP.PIECES
047
048 CASE V.KEY.3 EQ 'D' AND MOD(V.ZERO.POSN, 4) NE 0 ;* left
049
050     V.POSN.ZERO.NEW = V.ZERO.POSN + 1
051     GOSUB SWAP.PIECES
052
053 CASE V.KEY.3 EQ 'A' AND V.ZERO.POSN LE 12 ;* up
054
055     V.POSN.ZERO.NEW = V.ZERO.POSN + 4
056     GOSUB SWAP.PIECES
057
058 END CASE

```

Here you can see that we not only analyse a key being pressed, but also check if the empty space on the board allows an adjacent tile to move in requested direction to close it. To move a tile we use the section SWAP.PIECES which in turn uses section DRAW.PIECE to draw tiles:

```

075 *=====
076 SWAP.PIECES:
077
078 V.BOARD<V.ZERO.POSN> = V.BOARD<V.POSN.ZERO.NEW>
079 V.BOARD<V.POSN.ZERO.NEW> = 0
080 V.PIECE = V.POSN.ZERO.NEW
081 GOSUB DRAW.PIECE
082 V.PIECE = V.ZERO.POSN
083 GOSUB DRAW.PIECE
084
085 RETURN
086

```

What if some other key was pressed? See here:

```

059
060
061     CASE 1
062
063         V.KEY = UPCASE(V.KEY)
064         IF V.KEY EQ 'Q' THEN BREAK
065
066     END CASE
067
068 REPEAT

```

We understand only “Q” and quit the game if it was pressed (regardless the case). All other keys are ignored and we start the main loop again.

The last piece of this jigsaw puzzle:

```

069
070 CRT @(-1)
071 ECHO ON
072 STOP
073
074

```

I'd recommend to collect all the source above and try to compile it, make some improvements etc.

Don't forget to use "BI" command of JED if you do that so you'll not end up with something like:

```
013 LOOP
014
015 IF V.BOARD EQ V.FINI THEN
016     TEXT = 'YOU WIN'
017     PRINT TEXT
018 INPUT DUMMY
019 BREAK
020     END
021
022 V.KEY = KEYIN()
023
024     BEGIN CASE
025
026 CASE V.KEY EQ CHAR(27) ;* Esc
```

## 25 Local applications, code rating, enrichment, AUTO.ID.START, SEARCH, jBASE file types, jstat

**B**ack to T24 programming. In T24 you can create your own application using a set of templates provided by Temenos. Though we are on R10 here with new and a bit sophisticated templates, let's start from an old one (which however still works). The simplest one is "L-type" which refers to the value of *PGM.FILE* field *TYPE*.

Copy the file *TEMPLATE.L* from *T24.BP* to *ETC.BP* directory, rename it to your chosen name. (Here I don't even remind you that you have to choose a meaningful name and this name shouldn't be already in use – remember

“jshow -c” command?)

Firstly we need to think of a plausible reason to use a local application. Earlier we saw how *FT* input method could be checked (OFS, terminal, Browser). Now imagine that we want to store this method in a local application rather than trash the core field *CREDIT.THEIR.REF*.

To proceed with data storage we use “L-type” application because it doesn’t allow user input. Imagine that we write to an “H-type” application. Not only we need to be sure that nobody edits the record we’re trying to write into, we also need to think about history record creation, audit fields etc. Of course there is relatively new API subroutine *F.LIVE.WRITE* which handles both history and audit trail, but if we’re not going to input records manually or via OFS, “L-type” application is the best choice.

OK, then the name chosen will be, say, *FT.CREATION.METHOD*.

After the file *TEMPLATE.L* is renamed to *FT.CREATION.METHOD*, it’s necessary to rename the subroutine itself:

```
File ETC.BP , Record 'FT.CREATION.METHOD' Insert 06:53:42
Command->

0001 * Version 6 22/05/01 GLOBUS Release No. 200511
31/10/05
0002 *-----
0003 * <Rating>308</Rating>
0004 *-----
0005 SUBROUTINE FT.CREATION.METHOD
```

You might ask – what’s the “Rating” tag is? When no more available (for the client) *EB.COMPILE* tool is used to compile the local source, so-called “Rating” is calculated. The less this rating is (it could be below zero), the better. Does it reflect the quality of the code? In my opinion, only to some extent. It greets comments, for example. But how it can be sure a comment isn’t “bla-bla-bla”? Another example – it doesn’t like long routines without breaking to subsections not regarding the task. OK, I agree with this rating

that “GOTO” is evil but see the header of the standard template – it says the rating is 308 which is not very good. Once I’ve tried to compile a core routine `SEC.TRADE` that came to my possession. The purpose was to proceed with some debugging. Its rating was about 6000.

Anyway we’re going to use `BASIC` and `CATALOG` commands so this rating will not be applied.

Back to template editing. At the very bottom there is subsection called `DEFINE.PARAMETERS`. We need to develop the table structure suitable for our task and put it into respective arrays (`F`, `N`, `T` etc – we saw all them before).

We need only one field to store `FT` input method. We have 3 methods so far (see `ANC.TEST` routine) – “INPUT VIA BROWSER”, “INPUT VIA TERMINAL” and “OFS INPUT”. Instead of storing text it’s a good idea to assign codes to these descriptions – 1, 2 and 3 accordingly. How to see the text near the code – we’ll see later (it’s called “enrichment”).

So let’s name our field and put its size and data type into the respective arrays:

```
File ETC.BP , Record 'FT.CREATION.METHOD' Insert 06:57:12
Command->

0136 Z+=1 ; F(Z) = 'INPUT.METHOD' ; N(Z) = '1.1' ; T(Z) = ''
```

Though it’s not recommended by Temenos to put several lines of code into one line in the source file, these lines are short and belong to one field definition, so be it.

Our application doesn’t allow manual input, so `N` and `T` arrays don’t really do any work here, but let’s keep things tidy from the very beginning. In addition, `ID` is also described using the same definitions and `ID` will be input “manually” (even if we choose it from a list rather than type):

```
File ETC.BP , Record 'FT.CREATION.METHOD' Insert 06:59:45
Command->


0129 ID.F = 'FT.ID' ; ID.N = "___" ; ID.T = "-"
```

In this example the simplest solution for ID is to keep it the same as *FT* ID. How do we know the length and data type of *FT* ID?

*STANDARD.SELECTION* record doesn't give us the ultimate answer:

```
1.  1 SYS.FIELD.NAME.  @ID
2.  1 SYS.TYPE.....  D
3.  1.  1 SYS.FIELD.NO  0
4.  1.  1 SYS.VAL.PROG  IN2A&&&L##/#####/#####
6.  1 SYS.DISPLAY.FMT  25R
```

The ID is displayed in 25-character field being right-justified, but how many characters can we input when *FT* ID is required?

It's about time to look into *AUTO.ID.START* application. It's used to generate next ID for applications. Remember that we've used  or omitted ID in OFS message to get a new ID generated for us?

In old times there was one generic scheme to get next ID number for the deals – every transaction-related application has a prefix (e.g. FT, LD, MM etc), then goes the year (2 last digits), then current day number in a year (which gives us 3 more digits in the range of 001-366), then – 5 digits of sequential number for the current day. You can still see it in *MM*-related record:

```
R10 Model Bank      Auto ID Start SEE
KEY.....          MONEY.MARKET
-----
1.  1 DESCRIPTION.... Money Market
2.  1 APPLICATION.... MM.MONEY.MARKET
3.  1 ID.START.....  MM9820400001
```

But what to do if there is more than 99,999 deals in a day?

First attempts were to increase *FT* ID span by including there branch ID and increasing the sequential number, but then there was another problem: process of getting a sequential number assumed that there is a counter somewhere that is to be locked and subsequently updated by every process running in parallel. That counter is stored in *LOCKING* application:

```
FBNK.MM.MONEY.MARKET
001 MM1000500059
002 MM9820400001
```

To get rid of locking (which affects performance) it was decided to have an option of non-sequential IDs (which, in addition, could contain not only digits but also alphabetic symbols). See *FT*-related record of *AUTO.ID.START*:

```
R10 Model Bank      Auto ID Start SEE
KEY.....           FUNDS.TRANSFER
-----
1.  1 DESCRIPTION... FUNDS TRANSFER UNIQUE ID
2.  1 APPLICATION... FUNDS.TRANSFER
3.  1 ID.START..... FT
4.  1 UNIQUE.NO..... YES
5.  1 BASE.TABLE.....
6.  1 ID.LENGTH.....
```

Allowed characters and variable portion length can be defined in fields *BASE.TABLE* and *ID.LENGTH*, by default *BASE.TABLE* consists of uppercase Latin characters (excluding vowels) and digits from 0 to 9; for generating a random ID suffix there is 5 digits and/or letters. We saw these IDs before – when we’ve tried to create a new *FT* record – *FT10005QWCFC*, *FT10005SVD9T*, *FT10005K4V1H* etc.

Why vowels are excluded? For not getting foul words being generated. Yes, I’m not kidding.

So we put the following values to provide ID definition of our application:



```
File ETC.BP , Record 'FT.CREATION.METHOD' Insert 06:53:42
Command->

0129 ID.F = 'FT.ID' ; ID.N = "12.12" ; ID.T = "A"
```

Where “12.12” means “maximum 12, minimum 12” accordingly. “A” in ID.T refers to IN2A system check routine which was mentioned earlier (alphanumeric input).


Last but not least – put the comment into routine header. Comment needs to contain your name, current date and the purpose of this development:

```
File ETC.BP , Record 'FT.CREATION.METHOD' Insert 19:23:22
Command->

0005 SUBROUTINE FT.CREATION.METHOD
0006 *****
0007 *
0008 * V.Kazimirchik, 1/Oct/2010.  New application to store
0009 * FT deal input method.
0010 *
0011 *****
```

Now let’s compile this template. Then we need *FILE.CONTROL* and *PGM.FILE* records for our application.

*FILE.CONTROL* record can be copied from the record “FUNDS.TRANSFER” – thus we make sure that both these applications have the same type. Since there’s no C function for *FILE.CONTROL*, we just copy it from back-end and then amend in JED. (T24 function I for an existing record also doesn’t work for *FILE.CONTROL*.)

```
jsh mb10 ~-->COPY FROM F.FILE.CONTROL
FUNDS.TRANSFER,FT.CREATION.METHOD 

1 records copied

jsh mb10 ~-->JED F.FILE.CONTROL FT.CREATION.METHOD
```

Correct definition in field 1 and contents of field 3 (since we don't need \$NAU and \$HIS files) and (if you feel like it) also correct audit trail:

```
0001 FT creation method
0002 FT
0003
0004 2
0005 5
0006 FIN
0007 TRANSACTION
0008 Y
0009 N
0010 BIN
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020 1
0021 1_VLADIMIRK
0022 1008302252
```

Here's a new *PGM.FILE* record:

```

R10 Model Bank          PROGRAM FILE, INPUT

PROGRAM                FT.CREATION.METHOD
-----
1 TYPE.....          L
2.  1 GB SCREEN.TITLE FT CREATION METHOD
3 ADDITIONAL.INFO...
4.  1 BATCH.JOB.....
5 PRODUCT.....       FT          FUNDS TRANSFER

```

Now it's time to create insert file for our application. Yes, we have only one field at the moment but who knows what business guys might ask us to squeeze into that application in the future?

To do that firstly log in into T24, then log out (yes, we need some core variables in common area, otherwise it blows up). Then:

```

jsh mb10 ~-->FILE.LAYOUT 

Use Select List or Input Individually either :
List of FILE names from which to build insert modules
via their dictionarys

List of PROGRAM names from which to build insert modules
through calling.

Enter Program/File(s) : FT.CREATION.METHOD 
Enter Program/File(s) : 
Build Insert from File (D)ictionarys or (P)rograms or
(Q)uit
<CR = (P)rograms> : 

-----
Program is FT.CREATION.METHOD

Enter output Name - <CR> = I.F.Entryname : 

Enter PREFIX or <CR> = NONE :

```

Here we need to create a prefix which is attached to field names in our application. We need to have it more or less unique. Why more or less? Because if in another application we have similar or same prefix, full field names with prefix might occur to be the same. But it hurts only if we include both insert files in a routine. Let's see an example:

If we have field with name `AMT` and our prefix is `FT.TAX`, in summary it makes `FT.TAX.AMT` which is field 69 of `FUNDS.TRANSFER` which has prefix `FT` and field name `TAX.AMT`.

By the way, the simpler the prefix is, the older is core application.

It's an impossible task to see all core insert files for their prefixes, that's why I use the term "more or less". If you know which inserts you're going to use in your development, go through them to be sure that resulting field names won't be the same. But here's the better way – try to invent something that looks a bit unusual, search for it in `GLOBUS.BP`, then – if not found – use it.

E.g., for `FT.CREATION.METHOD` we intend to have the prefix "F.C.M.":

```
jsh mb10 ~-->SEARCH GLOBUS.BP 
String :F.C.M. 
String : 
Record Keys : * 
No Records selected
```

Now we have really unique prefix. Back to the screen of `FILE.LAYOUT`:

```
Enter PREFIX or <CR> = NONE : F.C.M. 
Enter SUFFIX or <CR> = NONE : 
Processed 1 matrix entries for FT.CREATION.METHOD program
```

What's in there?

```
jsh mb10 ~-->CT BP I_F.FT.CREATION.METHOD 
      I_F.FT.CREATION.METHOD
001 * File Layout for FT.CREATION.METHOD Created 3 OCT 10
      at 10:18AM by telnet
002 * PREFIX[F.C.M.] SUFFIX[]
003 EQU F.C.M.INPUT.METHOD TO 1, FtCreationMethod_InputMethod
      TO 1
```

Here we can see that there is a recent addition of field reference constructed without prefix (`FtCreationMethod_InputMethod`). To my knowledge, this functionality appeared around R08. I still prefer the old notation. (You never know what T24 release will be there on your next assignment, also there were many things declared and then discontinued, like VB scripting in Globus Desktop.)

Yes, `FILE.LAYOUT` created insert file in `BP` subdirectory of `bnk.run`. Since we need it in `ETC.BP` (under no circumstances we're going to put it to `GLOBUS.BP`) we can use `jBASE` commands `COPY` and `DELETE` or move the file using `OS` commands.


Next step in creation of a local application – creation of data and dictionary files. To do it login to T24 and type `CREATE.FILES` from `AWAITING APPLICATION` prompt, for “COMPANY CODE” input `BNK`, for “LIST NAME” just press , for “FILE NAME” input `FT.CREATION.METHOD`, for the same question being repeated just press , then `Y` to continue.

After scrolling up your terminal window you can see the following:

```
[ 417 ] File ..\bnk.dict\F_FT_CREATION_METHOD]D created ,
type = J4
[ 417 ] File ..\bnk.data\ft\FBNK_FT_CREATION_METHOD created ,
type = JR
```

Note about file types listed above: by default data files are now created

with type “JR” which means “resilient”. You can look into jBASE manuals for more explanations but the general idea is: a resilient file wouldn’t corrupt, neither it needs resizing. Let’s see the status of the new file:

```
jsh mb10 ~-->jstat -v FBNK.FT.CREATION.METHOD 
File Path = ..\bnk.data\ft\FBNK.FT.CREATION.METHOD
File Type = JR, Hash method = 5, Created = Sun Oct 3
11:05:18 2010
Frame size = 4096, OOG Threshold = 2048
File size = 8192, Freespace = 0 frames
Internal Modulo = 3/7/19, External Modulo = 31
Inode no. = 393420, Device no. = 17308
Accessed = Wed Sep 01 12:40:17 2010, Modified = Wed Sep 01
12:40:17 2010
Backup = YES, Log = YES, Rollback = YES, Secure updates =
YES
Deallocate pointers : NO Deallocate frames NO
Record Bytes = 0, Record Count = 0
Bytes/Record = 0, Bytes/Group = 0
Data Frames = 0, Ptr Frames = 0
OOG Bytes = 0, OOG Frames = 0
Sum Squares = 0, Std Dev Mean = 1
```

Yes, secure updates are on, so this file won’t become corrupted... at least that’s what manuals say. To have such file type support you need “jBASE non-stop” licence.

General note: properly sized J4 file is faster than JR file so I wouldn’t recommend to have all your files in JR format.

Since file type isn’t “INT”, we need to run CREATE.FILES for all companies. Results look like:

```
[ 417 ] File ..\bnk.data\ft\FEU1.FT.CREATION.METHOD created ,
type = JR
```

Dictionary already exists so only data file is created. Note company mnemonic (`FEU1`) in file name. Now you know that whenever you see the constant `FBNK` in the source code, you are looking into something that wouldn't work in multi-company environment.

Having done that, now we're facing the last step – creation of `STANDARD.SELECTION` record. At `AWAITING APPLICATION` prompt type `SS, I FT. CREATION.METHOD`. You'll see record which is so far empty. Go to field `REBUILD.SYS.FIELDS`, input there `Y` and commit the record. Return to `AWAITING APPLICATION` prompt and repeat the last command (or use `F7` twice) to see that now we have populated `STANDARD.SELECTION` record.

Where the core took field definitions? From the compiled source of our application. We used this technique before to retrieve structure arrays from `FT.TXN.TYPE.CONDITION`.

There are different tools which can help you to build all components of your local application. There is a core application `EB.TABLE.DEFINITION` which can be used to create local applications without programming, but I wanted you to pass all the way in the low level at least once for better understanding.

In next section we'll see how to update our new local application from a `VERSION` routine. For this task we'll use one attached to `VERSION` field `INPUT.ROUTINE`.

## 26 File I/O

`F`ile I/O in T24 – as I've written above – is done via wrappers. They are described in “Subroutine Guide.pdf” so I wouldn't list all of them, just present some examples.

Let's take the logic from our routine `ANC.TEST` – copy its source to file `INPUT.TEST` and modify the following way:

```
001 SUBROUTINE INPUT.TEST
002 * V.Kazimirchik, 3/Oct/2010. Store FT input method
003 * in application FT.CREATION.METHOD.
004 * Input routine for FT version.
005
```

This is the header – subroutine name, comments. It’s good to see in the source which type of routine we have here.

```
006 $INSERT I.COMMON
007 $INSERT I.EQUATE
008 $INSERT I.GTS.COMMON
009 $INSERT I.F.OFS.SOURCE
010 $INSERT I.F.FT.CREATION.METHOD
011
```

Inserts that we need for our work. Our new application insert is also here.

```
012 * Open file
013
014 FN.FCM = 'F.FT.CREATION.METHOD' ; F.FCM = ''
015 CALL OPF(FN.FCM, F.FCM)
016
```

Here we’re opening our local application using OPF wrapper subroutine. It supports caching of opened files so if this file is already opened by another subroutine, whether a core or a local one, file wouldn’t be reopened and performance wouldn’t suffer.

After the file has been opened the variable FN.FCM contains full file name. (It’s FBNK.FT.CREATION.METHOD in case we’re working in the main company with mnemonic “BNK” used for financial files.) In fact, sometimes OPF is used only to obtain full file name, e.g. for subsequent SELECT.



```

017 * Check input method
018
019 V.METHOD = ''
020
021 IF GTSACTIVE THEN
022     IF OFS$SOURCE.REC<OFS.SRC.SOURCE.TYPE> EQ 'SESSION'
THEN
023         V.METHOD = 1 ;* 'INPUT VIA BROWSER'
024     END ELSE
025         V.METHOD = 3 ;* 'OFS INPUT'
026     END
027 END ELSE
028     V.METHOD = 2 ;* 'INPUT VIA TERMINAL'
029 END
030

```

Here's our main logic transferred from ANC.TEST. Also – comment at the same line with code is introduced.

```

031 * Compose a record
032
033 R.FCM = ''
034 R.FCM<F.C.M.INPUT.METHOD> = V.METHOD
035

```

The record which we're going to write to our local application is initialised and then populated. We address field via its definition in the insert file.

```

036 * Write a record
037
038 CALL F.WRITE(FN.FCM, ID.NEW, R.FCM)
039
040 RETURN
041 END
042

```

F.WRITE awaits a dynamic array as 3<sup>rd</sup> parameter, that's why we've prepared the record this way.

Now – compile INPUT.TEST, create *EB.API* record for it, then edit the *VERSION* record FUNDS.TRANSFER,TEST – remove ANC.TEST since we don't need it anymore and attach INPUT.TEST as INPUT.ROUTINE. Launch this *VERSION* in terminal, create *FT* record, then do it in Browser... hope you can do it all by yourself now?

See then what's there in application *FT.CREATION.METHOD*:

R10 Model Bank	FT CREATION METHOD SEE
FT.ID.....	FT10005HRTNY
-----	
1 INPUT.METHOD.....	2

R10 Model Bank	FT CREATION METHOD SEE
FT.ID.....	FT10005SG89D
-----	
1 INPUT.METHOD.....	1

Of course user won't have to memorize what "2" means etc. Earlier I've wrote that we'll see how so-called "enrichments" are displayed. In our case we have several methods to achieve that:

- Create another local application which contains records 1, 2 and 3 and the field (usually called *DESCRIPTION*) which contains... well, descriptions of these codes. These descriptions even can be made multi-lingual so user will see it in his native language (provided that it is set up in core application *LANGUAGE*). Then this application could be used as *CHECKFILE* for application *FT.CREATION.METHOD*. This is a preferred way and if you want, proceed with it yourself. Note that it has to be based on a template that allows input so you'll be able to input there your records and – if necessary – amend their descriptions later.

- Put descriptions into the source code of *FT.CREATION.METHOD*. Not the best way but acceptable assuming that we have only 3 codes. Let's try to proceed with it.

It's a bit tricky to assign the enrichment since *TEMPLATE.L* doesn't allow input and therefore has much less places where a user-defined logic can be added. The only place that is more or less adequate – section *CHECK.ID*. The minor inconvenience is that at that stage application record isn't yet available in global variable *R.NEW*. We only have *ID.NEW* to start with, so we need to read the record in question before the core does the same. We're causing no harm in so doing because the core then will take this record from cache so no I/O increase here.

We have our application table already opened by the core into global file variable *F.FILE*. We'll read application record using *F.READ* so the cache will be used afterwards.

Enrichments are stored in global array *T.ENRI* which we are able to modify. So the changes will be:

```
020 $INSERT I.F.FT.CREATION.METHOD ;* KZM
```

and

```
096 * KZM S
097 * Setting an enrichment
098
099 V.ERR = ''
100 CALL F.READ('F.' : APPLICATION, ID.NEW, R.APP, F.FILE,
V.ERR)
101
102 IF V.ERR THEN RETURN
103
```

Here we read the record (note global variable *APPLICATION*) and proceed with possible error. One might ask: there should be the record with this ID, what else might happen that triggers an error? Answer: anything, starting

from network error or a server error. Wherever an error might be caught, we need to provide the correct way of handling it. In our case we simply end the processing since enrichment display isn't that crucial. In other cases you might need to raise a fatal error or provide the error code to be returned to calling routine to be analysed there.

```
104 V.FLD = R.APP<F.C.M.INPUT.METHOD>
105 V.ENRI = ''
106
```

Here we extract field value from the record and initialise the local variable necessary for the result of analysis. By the way, local variables are visible only inside current source file (though in all subsections) so don't try to invent unique names for them.

However variable names are to be understandable.. see some excerpts from real code:

```
L.AM = S.1 + S.2 + S.3

or:

APPL = 'LD'
ARR.X = ARR.A
GOSUB AMEND.STMT.ARRAY
ARR.A = ARR.X

or:

IF L.CH THEN L.CH=L.CH
```

But there is another side in variables naming:

```

LOC.G.FN.B.TPU.RCSE.PROCESS.LIST= \
    'F.B.TPU.RCSE.PROCESS.LIST'

LOC.G.F.B.TPU.RCSE.PROCESS.LIST = ''

CALL OPF(LOC.G.FN.B.TPU.RCSE.PROCESS.LIST,
        LOC.G.F.B.TPU.RCSE.PROCESS.LIST)

CALL F.READU(LOC.G.FN.B.TPU.RCSE.PROCESS.LIST,
            LOC.ID.B.TPU.RCSE.PROCESS.LIST,
            LOC.R.B.TPU.RCSE.PROCESS.LIST,
            LOC.G.F.B.TPU.RCSE.PROCESS.LIST,
            LOC.ER.B.TPU.RCSE.PROCESS.LIST, 'E')

IF LOC.ER.B.TPU.RCSE.PROCESS.LIST NE "RECORD LOCKED" THEN
    LOC.R.B.TPU.RCSE.PROCESS.LIST= \
        LOC.ID.B.TPU.RCSE.PROCESS.LIST
    CALL F.WRITE(LOC.G.FN.B.TPU.RCSE.PROCESS.LIST,
                LOC.ID.B.TPU.RCSE.PROCESS.LIST,
                LOC.R.B.TPU.RCSE.PROCESS.LIST)
    CALL F.RELEASE(LOC.G.FN.B.TPU.RCSE.PROCESS.LIST,
                  LOC.ID.B.TPU.RCSE.PROCESS.LIST,
                  LOC.G.F.B.TPU.RCSE.PROCESS.LIST)

and so on for many pages...

```

Can you read this source? Probably no. One good thing with this code – here line continuations in jBASE are introduced. You can continue on the next line after comma or use backslash for this purpose.

OK, go further with our task.

```

107 BEGIN CASE
108
109 CASE V.FLD EQ 1
110     V.ENRI = 'INPUT VIA BROWSER'
111
112 CASE V.FLD EQ 2
113     V.ENRI = 'INPUT VIA TERMINAL'
114
115 CASE V.FLD EQ 3
116     V.ENRI = 'OFS INPUT'
117
118 END CASE
119

```

Here's the main logic...

```

120 T.ENRI<F.C.M.INPUT.METHOD> = V.ENRI
121
122 * KZM E

```

Finally, here's the assignment of enrichment. As you see, I've put comments of all changes to template code, marking the start and the end with "KZM S" and "KZM E" accordingly.

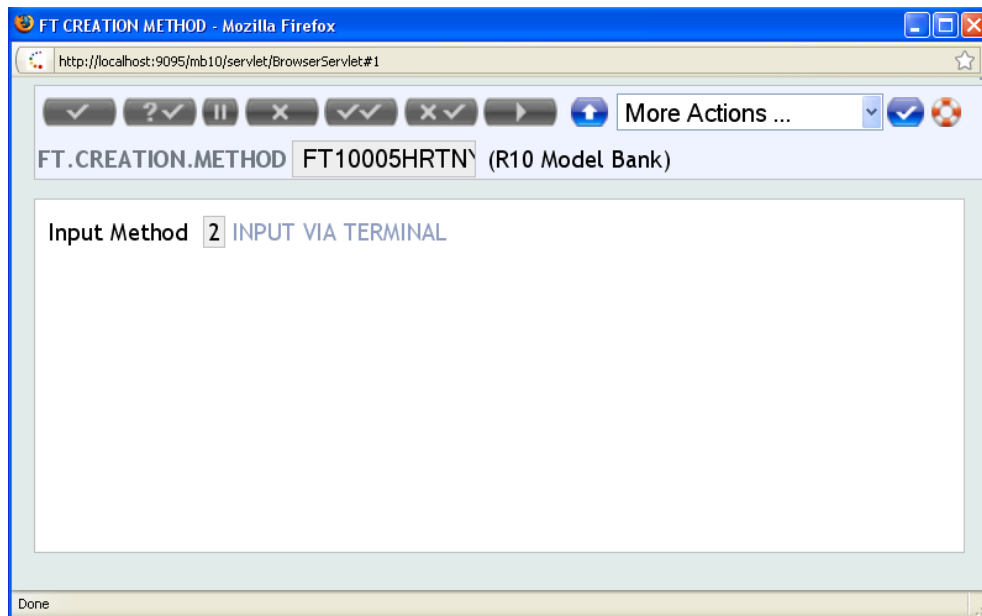
Result:

```

R10 Model Bank          FT CREATION METHOD SEE
      FT.ID..... FT10005HRTNY
-----
1 INPUT.METHOD..... 2          INPUT VIA TERMINAL

```

And – it works in Browser as well:



Global arrays and variables require extreme care. Here we knew what to do and how it works. Sometimes people try to amend, for example, T array on the fly, trying to make a field NOINPUT depending on other fields' values... It worked in Desktop, under Browser it doesn't. In case of getting such requirements probably it's a good idea to rethink the whole business process and not to try to rape the system.

## 27 Changing structure of a local application – amending field type

OK, but what if we want to check the input method when *FT* records are amended? In our *VERSION* we have number of authorisations set to 1, which means that *FT* record created using this *VERSION* can be later amended.

It's necessary to note that some T24 applications allow amendment of

records even after authorisation, but *FUNDS.TRANSFER* does not.

Since the field *INPUT.METHOD* used to store this information is single-valued we need to make it multi-valued. How?

Firstly amend the source code:

```
0166
0167 Z+=1 ; F(Z) = 'XX.INPUT.METHOD' ; N(Z) = '1.1' ; T(Z) =
', '
0168 *
```

Then compile the source and rebuild *STANDARD.SELECTION* record.  
The result:

```
1. 3 SYS.FIELD.NAME. INPUT.METHOD
2. 3 SYS.TYPE..... D
3. 3. 1 SYS.FIELD.NO 1
4. 3. 1 SYS.VAL.PROG IN2
5. 3 SYS.CONVERSION.
6. 3 SYS.DISPLAY.FMT 1R
7. 3 SYS.ALT.INDEX.. N
8. 3. 1 SYS.IDX.FILE
9. 3 SYS.INDEX.NULLS
10. 3 SYS.SINGLE.MULT M
11. 3 SYS.LANG.FIELD. N
12. 3 SYS.GENERATED.. Y
```

And we can also see it on the application screen:

```
R10 Model Bank          FT CREATION METHOD SEE
      FT.ID..... FT10005HRTNY
-----
1.1 INPUT.METHOD... 2          INPUT VIA TERMINAL
```

Now let's amend the routine *INPUT.TEST* which is responsible for popu-



lation of this field:

```
0004 * Input routine for FT version.
0005 *
0006 * 14/Sep/2010: field INPUT.METHOD is now multi-valued.
0007
```

Here – a comment in the header describing the nature of this change.

```
0033 * Read a record (even if it doesn't exist)
0034
0035 CALL F.READ(FN.FCM, ID.NEW, R.FCM, F.FCM, V.ERR)
0036
0037 * Compose or update a record
0038
0039 V.INP.METHOD = R.FCM<F.C.M.INPUT.METHOD>
0040 V.INP.METHOD<1,-1> = V.METHOD ;* add a new value
0041 R.FCM<F.C.M.INPUT.METHOD> = V.INP.METHOD
0042
```

Where “Compose a record” comment had been, we change the logic to read a record from local application and amend it (or create a new one if there was no such record before).

Let's try it (under Browser). Result:

```
R10 Model Bank          FT CREATION METHOD SEE
      FT.ID..... FT10005HRTNY
-----
      1.1 INPUT.METHOD... 2
      1.2 INPUT.METHOD... 1
```

We don't see enrichment anymore... Well, that's the dark side of hard-coding. Leave it be for now.

## 28 Records locking

**THE** question is: were we correct when we updated a record in a local application without record lock? Probably we had to use `F.READU` instead of `F.READ`?

In this particular case all is OK because:

- We update “L”-type application which users are not able to edit manually.
- ID of the record is the same as *FT* ID and it’s impossible to edit the same “master” *FT* record manually (or via OFS) in T24.

Actually, why it’s impossible?

When application code is invoked and the record is edited, T24 core applies a lock to a record in `$NAU` file of the application. Even if such record doesn’t exist, the lock shows the intention to update the record after editing.

In older releases of jBASE/T AFC we could use command `SHOW-ITEM-LOCKS` to see current locks applied in the system. Nowadays locks are maintained by using T24 application `RECORD.LOCK` to support complex multi-server architectures etc. If we use `READU` Basic statement, old scheme is applied. So for new releases use only `F.READU` which is a wrapper described in “Subroutine Guide.pdf”.

To test this just open any record for editing (I function) and then see that in application `RECORD.LOCK` a corresponding record appears – with ID like “`FBNK.FUNDS.TRANSFER$NAU.FT10005HRTNY`”. If you use terminal to edit the record, you’ll see the same effect plus good old `SHOW-ITEM-LOCKS` output:

PORT	PID	FILENAME	RECORDKEY	LOCK#	PORT/-PID
1	2156	..\bnk.data\ft\FB NK_FUNDS_TRANSFER #NAU	FT10005HRTNY	0x61998168,W	---

## 29 Programming for enquiries

**I'D** better not go into details how enquiries are created – most of the stuff is documented quite well. Where we can interfere with programming is:

- “Build” routine. Proceeds selection and may control and/or amend selection criteria. For example, you can add additional restriction if required (or build a work file – it’s what manuals insist on but I know no examples of that).
- “Conversion routine” to proceed a field in enquiry output. Include `I_ENQUIRY.COMMON` to your subroutine and manipulate `O.DATA` gloval variable.
- Routine for “no-file enquiry”. Mainly used when there are several files to take records from.

In old Desktop days I’ve used build routine to limit the number of records that are being output as a result of, say, clicking on a drop-down field. You had to wait a lot for enquiry core logic to build 200 pages (that’s the limit defined in *SPF*; though it can be exceeded if necessary using application *ENQUIRY.REPORT*). On early releases of jBASE in such situation you were lucky enough to get “Segmentation violation” fatal error.

Under Browser it doesn’t make much sense though, because it shows enquiry output by pages. However, let’s try this exercise.

Knowing that there are “special” enquiry names that are used for listing

a table, I've written a build routine to limit the default selection. In my case it was history file of application ... yes, again *FUNDS.TRANSFER*. I promise that my further examples will be based on some other application...

```
0001 SUBROUTINE ENQ.NARROW(P.ENQ)
0002
0003 *-----*
0004 * V.Kazimirchik (KZM), 2005-2010.
0005 * Build routine for enquiry to narrow the output.
0006 *-----*
0007
0008 $INSERT I.COMMON
0009 $INSERT I.EQUATE
```

Usual start...

```
0010
0011 IF P.ENQ<2,1> EQ '' THEN ;* no selections were input or
"list" menu item was clicked
0012
0013     V.DATE = TODAY
0014     CALL CDT('', V.DATE, '-1W')
0015
0016     V.JULDATE = ''
0017     CALL JULDATE(V.DATE, V.JULDATE)
0018
0019     P.ENQ<2,1> = '@ID'
0020     P.ENQ<3,1> = 'LK'
0021     P.ENQ<4,1,1> = 'FT' : V.JULDATE[5] : '...'
0022
0023 END
```

Main logic. Another T24 API subroutine (JULDATE) is introduced here.

```

0024
0025 *-----*
0026
0027 RETURN
0028 END

```


That's all. Now we need to attach this routine to the enquiry whose name starts prom % sign (so-called "percent enquiry"):

```

R10 Model Bank      ENQUIRY, INPUT
  ENQUIRY..... %FUNDS.TRANSFER$HIS
-----
1 PAGE.SIZE ..... 4,19
2 FILE.NAME..... FUNDS.TRANSFER$HIS
3.  1 FIXED.SELECTION
4.  1 FIXED.SORT....
5.  1 OPEN.BRACKET...
6.  1 SELECTION.FLDS.
7.  1.  1 GB SEL.LABEL
8.  1 SEL.FLD.OPER...
9.  1 REQUIRED.SEL...
10.  1 CLOSE.BRACKET..
11.  1 REL.NEXT.FIELD.
12.  1 BUILD.ROUTINE.. ENQ.NARROW
13.  1.  1 HEADER.....
14.  1 FIELD.NAME..... 0
15.  1.  1 OPERATION... @ID
16.  1 COLUMN..... 1
17.  1 LENGTH.MASK.... 25R
...
35.  1 SINGLE.MULTI... S
14.  2 FIELD.NAME..... 1
15.  2.  1 OPERATION... TRANSACTION.TYPE
16.  2 COLUMN..... 27
17.  2 LENGTH.MASK.... 4L

```

Now relogin to T24 (yes such things are cached) and type FT L ; L from AWAITING APPLICATION prompt (there are spaces surrounding ';' character).

Then –  to invoke the enquiry.

```
R10 Model Bank
```

```
-----  
FT09338BOP12;1 ACDI  
FT09338BDR4R;1 OT40  
FT09338BGT4J;1 AC  
FT09338CNFRT;1 AC  
-----
```

Apparently our build routine wasn't triggered (to double-check you can input `DEBUG` statement to it and try again). Let's try Browser (FT -> More Actions -> List History File):



Yes we see here only records that start from FT10004 – from the last work day.

Why it worked for Browser and did not under Classic... that's the question for Temenos Helpdesk.

Other enquiry-related types of subroutines are quite clearly documented in manuals so you can try them yourself.

## 30 Performance, code analysis, local fields, DYNAMIC.TEXT

**W**ell, probably – long-awaited topic – especially “performance”.

Usually it sounds like: “Our system is slow, do something” or “give us a set of universal rules which (after implementation) will keep our system fast forever”. In real life it’s not possible to find two T24 installations that are equal. Some of them are even less equal, if you know what I mean.

Well, there’s a lot of places where T24 performance can be tackled. I’ll try to provide typical causes and some advice on programming style.

If you’ve got poor performance, firstly check files sizing. There are tools to do that, I’ll not go deeper here, nor I intend to stop at obvious reasons like slow hardware/infrastructure etc.

It’s a good idea sometimes to give your code to some other person for a fresh look. (I even manage to get a look at my own code after a year or two and this look is really fresh as if I look at the code that is not mine.) The reason is obvious – when you program some task, it is never a static thing – something is always added, changed, removed etc. Finally after extensive amendments it looks that it’s much more efficient to rewrite the whole thing.

Here are some examples that don’t hit your eye at the beginning but there’s something in each of them to correct. Not necessarily these corrections improve performance – the result might be readability, better structuring, ease of maintenance etc.

You’ll see my answers in very small font (not to serve as a spoiler). All code is real.



Example 1:

```
* Read current COMPANY record
FN.COMPANY = 'F.COMPANY'
F.COMPANY = ''
CALL OPF(FN.COMPANY, F.COMPANY)
CALL F.READ(FN.COMPANY, 'US0010001', REC.COMPANY, F.COMPANY,
LOC.ERR)
```

What's wrong:

- There is a global variable ID.COMPANY which is to be used instead of 'US0010001'.
- We don't need to read company record since it's already available in global array R.COMPANY.

Example 2:

```
SUBROUTINE SAMPLE2(ENQ.DATA)
* This is a build routine for enquiry.
$INSERT I_COMMON
$INSERT I_EQUATE

      GOSUB OPENFILES
      IF Y.ERR NE '' THEN
          ENQ.ERROR = Y.ERR
      RETURN
      END
      GOSUB PROCEED
      RETURN

* Subroutines *
OPENFILES:
      Y.ERR = ''
      FN.ACCT = 'F.ACCOUNT' ; F.ACCT = ''
      CALL OPF(FN.ACCT, F.ACCT)
      FN.CUST = 'F.CUSTOMER' ; F.CUST = ''
      CALL OPF(FN.CUST, F.CUST)
      RETURN

PROCEED:
* Do something not relevant to our sample
      RETURN
END
```

What's wrong:

- There's no Y.ERR setting logic in OPENFILES section so error analysis is not necessary.
- Reference to I\_ENQUIRY.COMMON where global variable ENQ.ERROR is declared is missing.

Example 3:

```
SUBROUTINE SAMPLE3
* This is an input routine for TELLER application.
* If local field INPUT.DATE exists, it's populated with
* current bank date.
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.TELLER

  Y.LREF.POS = ''
  CALL GET.LOC.REF('TELLER', 'INPUT.DATE', Y.LREF.POS)
  IF Y.LREF.POS EQ '' THEN RETURN

  R.NEW(TT.TE.LOCAL.REF, Y.LREF.POS) = TODAY

  RETURN
END
```

Here we introduce core subroutine `GET.LOC.REF` which gets local field position based on its name. Though not mentioned in “Subroutine guide”, it can be safely used.

Local fields deserve a separate chapter. In brief – it’s emulation of many fields in one field called `LOCAL.REF`. Thus we can add our fields to any application (though losing one of 3 dimensions – local field can be multi-valued but not sub-valued). 2 applications are responsible for setting up local fields – `LOCAL.TABLE` and `LOCAL.REF.TABLE`.

There’s always a mess with local fields since their IDs in `LOCAL.TABLE` are numbers rather than names. Often somebody just changes local field name (or, for example, uses a name with spaces inside) and many local developments stop working. If names were IDs it would be impossible (though a smart guy can always reverse a field to prove who’s the master).

The last note goes not only for local fields (though I have never tried to reverse a local field). Most applications – like `CATEGORY` – allow the reversal of records though their IDs might be used somewhere else. The result

is the loss of referential integrity and this might happen in T24. Normally you find it out when COB crashes...

And – once you attached local fields to an application in `LOCAL . REF . TABLE`, you're not able to remove fields – only add (or amend their single/multi attribute). Of course if you know what you do you can correct it from `jsh` but it might be tricky – especially if application tables are already populated with data. So you have to support the order of local fields at all working environments (otherwise some local code and `VERSIONs` will become incompatible between areas).

**What's wrong in the example above:**

- `R.NEW` is a dimensioned array with one dimension. So assignment statement should look like:  
`R.NEW(TT.TE.LOCAL.REF)<1, Y.LREF.POS> = TODAY`
- We really don't need to insert `I.F.TELLER` here because the value of `LOCAL.REF` field position is stored in global variable `LOCAL.REF.FIELD`. So finally the assignment statement might look like:  
`R.NEW(LOCAL.REF.FIELD)<1, Y.LREF.POS> = TODAY`

## Example 4

```
SUBROUTINE SAMPLE4(PAR.ERR.MSG)
* This is RUN routine for W-type template.
* It should be run by only one user at any time.
* Shall return error message if it's being run
* by somebody else.

$INSERT I_COMMON
$INSERT I_EQUATE

* Make a lock to be sure nobody else runs this routine

  FN.LOCKING = 'F.LOCKING' ; F.LOCKING = ''
  CALL OPF(FN.LOCKING, F.LOCKING)

  Y.LOCKING.ID = 'USER.SAMPLE4'
  CALL F.READU(FN.LOCKING, Y.LOCKING.ID, R.LOCKING,
    F.LOCKING.ID, Y.ERR, 'E')

  IF Y.ERR AND Y.ERR NE 'RECORD NOT FOUND' THEN
    PAR.ERR.MSG = 'BEING RUN BY ANOTHER USER'
  END

* Do our logic here

  RETURN
END
```

### What's wrong:

- No RETURN after raising the error so the processing will continue.
- We don't need to open F.LOCKING since it's already opened by core T24 -- see F.LOCKING variable in I.RULES.
- There should be call to F.RELEASE after finishing the processing (but only if we don't write to the record being locked):  
CALL F.RELEASE('F.LOCKING', Y.LOCKING.ID, F.LOCKING)

Example 5:

```
SUBROUTINE SAMPLE5(PAR.RET.LIST, PAR.ERR.MSG)

$INSERT I_COMMON
$INSERT I_EQUATE

    GOSUB INIT
    GOSUB PROCEED

    RETURN

* Subroutines *
INIT:

    Y.LD.TYPE.POS = ''
    Y.LREF.NAME = 'PRODUCT.TYPE'
    CALL GET.LOC.REF('LD.LOANS.AND.DEPOSITS', Y.LREF.NAME,
                    Y.LD.TYPE.POS)
    IF Y.LD.TYPE.POS EQ '' THEN
        PAR.ERR.MSG = 'NO LOCAL REF PRODUCT.TYPE'
        RETURN
    END

    FN.CUST = 'F.CUSTOMER' ; F.CUST = ''
    CALL OPF(FN.CUST, F.CUST)

    RETURN

PROCEED:

* Do something - not relevant to our sample

    RETURN
END
```

What's wrong:

- There's no any comment in routine header.
- There's an error in section INIT and the return value of the subroutine was set accordingly, but there's no any error analysis in main section.
- Error message is to be composed of constant and variable part (otherwise in case of users with the language other than English we'll end up with many similar records in application *DYNAMIC.TEXT*). Here's the correct code:  
PAR.ERR.MSG = 'NO LOCAL REF &' :FM: Y.LREF.NAME

Some comments about *DYNAMIC.TEXT*: when a user with non-English language sees a message in T24, in this application appears an unauthorised record with specially formed ID. For example, for a message “DO NOT ENTER” record ID will be `DO.NOT.ENTER` and so on. All you have to do is to put message translation into appropriate field and authorise *DYNAMIC.TEXT* record. Next time that user will see that message in his native language.

But what if message has a variable part? E.g. “DO NOT ENTER HERE”, “DO NOT ENTER THERE”, “DO NOT ENTER ANYWHERE” and so on? Maybe it's not a good example but imagine that we have a field name as a variable part – as in example above. Answer is: use ampersand (&) instead of variable part or parts, e.g.:

```
PAR.ERR.MSG = 'NO LOCAL REF &' :FM: Y.LREF.NAME
or:
PAR.ERR.MSG = 'NO LOCAL REFS & AND &' :FM: Y.LREF.NAME \
:VM: Y.LREF2.NAME
```

Then in *DYNAMIC.TEXT* we'll have only the following IDs:

```
NO.LOCAL.REF.&
and
NO.LOCAL.REFS.&.AND.&
```

... and not hundreds of them like:

```
NO.LOCAL.REF.PRODUCT.NAME
NO.LOCAL.REF.PRODUCT.TYPE
NO.LOCAL.REF.INPUT.DATE
NO.LOCAL.REFS.INPUT.DATE.AND.INPUT.TYPE
```

Recently the new application for storing messages appeared in T24: *EB.ERROR*. It's intertwined with *DYNAMIC.TEXT* and I'm not absolutely sure how it all exactly works in each case (I wrote earlier here that for outputting an error several different methods are used in T24).

In general, in most cases messages in T24 can be translated using these 2 applications. (Unfortunately it doesn't apply to messages that are built into *BrowserWeb.war*.)

Example 6:

```
* This code is part of an input routine which is to check
* if we have local currency at any side of FOREX deal

IF R.NEW(FX.CURRENCY.BOUGHT) EQ "USD" \
    OR R.NEW(FX.CURRENCY.SOLD) EQ "USD" THEN
    ETEXT = "NO LOCAL CURRENCY ALLOWED"
    CALL STORE.END.ERROR
END
```

What's wrong:

- Usage of local currency as a constant is not recommended, use global variable *LCCY* instead.
- *CALL STORE.END.ERROR* should be preceded by setting of *AF* variable to indicate the field where error message appears. So it's better to make it two *IF* statements -- one for currency bought and another for currency sold.



Example 7:

```
SUBROUTINE SAMPLE7(PAR.RET.LIST, PAR.ERR.MSG)
* This subroutine does some processing and returns
* a list of values.

$INSERT I_COMMON
$INSERT I_EQUATE

      GOSUB INIT
      GOSUB PROCEED

RETURN

* Subroutines *
INIT:
  FN.LD = "F.LD.LOANS.AND.DEPOSITS"
  F.LD = ""
  CALL OPF (FN.LD, F.LD)

  FN.LC = "F.LETTER.OF.CREDIT"
  F.LC = ""
  CALL OPF(FN.LC, F.LC)

  *FN.COMPANY = "F.COMPANY"
  *F.COMPANY = ""
  *CALL OPF(FN.COMPANY, F.COMPANY)

  *CALL F.READ(FN.COMPANY, ID.COMPANY,
  * R.COMPANY, F.COMPANY, LOC.ERR.MSG)
  *IF LOC.ERR.MSG THEN RETURN

PROCEED:

* Do something - not relevant to our sample

      RETURN
END
```

What's wrong:

- Commented code complicates the view, remove it.
- There's no RETURN statement in section INIT. It means that section PROCEED will be executed twice.

Example 8:

Local application, section DEFINE.PARAMETERS:

```
MAT F = "" ; MAT N = "" ; MAT T = ""
MAT CHECKFILE = "" ; MAT CONCATFILE = ""
ID.CHECKFILE = "" ; ID.CONCATFILE = ""
ID.F = 'ORG.TYPE.ID' ; ID.N = '250' ; ID.T = 'A'

Z = 0

Z += 1 ; F(Z) = 'XX.LL.DESCRPTION'
      N(Z) = '50' ; T(Z) = 'ANY'
Z += 1 ; F(Z) = 'REPORTING.CODE'
      N(Z) = '3.1' ; T(Z) = FM: '-'
Z += 1 ; F(Z) = 'XX.ADDITIONAL.INFORM'
      N(Z) = '35' ; T(Z) = 'A'
Z += 1 ; F(Z) = 'RESERVED.5'
      N(Z) = '35' ; T(Z) = '' ; T(Z)<3> = 'NOINPUT'
Z += 1 ; F(Z) = 'RESERVED.4'
      N(Z) = '35' ; T(Z) = '' ; T(Z)<3> = 'NOINPUT'
Z += 1 ; F(Z) = 'RESERVED.3'
      N(Z) = '35' ; T(Z) = '' ; T(Z)<3> = 'NOINPUT'
Z += 1 ; F(Z) = 'RESERVED.2'
      N(Z) = '35' ; T(Z) = '' ; T(Z)<3> = 'NOINPUT'
Z += 1 ; F(Z) = 'RESERVED.1'
      N(Z) = '35' ; T(Z) = '' ; T(Z)<3> = 'NOINPUT'
V = Z
```

What's wrong:

- ID length -- do we really need this much? There's limit for Oracle which is 200, also we'll have difficulties with displaying ID which is that long.
- Field XX.ADDITIONAL.INFORM: name length exceeds 18 characters. Historically -- when terminal mode was only used -- field name with bigger length corrupted the screen (and it still does). However, for Desktop and Browser it doesn't matter much. Still, I try to conform to that rule.
- As we can see, this is L-type application ('V=Z' tells us so). So we don't have audit trail and therefore don't need reserved fields. For applications that allow input it's a good idea to provide reserved noinput fields that later can be renamed and used to store data; in this case existing records don't need to be amended regarding the position of audit trail since it will remain the same.

Example 9:

```
* This is part of VERSION-level routine for TELLER
* application.
* Get the category of till account

FN.ACCOUNT = 'F.ACCOUNT' ; F.ACCOUNT = ''
CALL OPF(FN.ACCOUNT, F.ACCOUNT)

Y.ACCOUNT.NO = R.NEW(TT.TE.ACCOUNT.1)
CALL F.READ(FN.ACCOUNT,Y.ACCOUNT.NO,R.ACCOUNT,F.ACCOUNT,
LOC.NOT.FOUND)
IF LOC.NOT.FOUND EQ '' THEN
    Y.ACC.CAT = R.ACCOUNT<AC.CATEGORY>
END
```

What's wrong:

- For an internal account it's much easier to get category number: simply extract characters 4 to 8 in its ID -- and you don't need to read an account record.

Example 10:

```
* This piece of code tries to find a category code
* in multi-valued field of local parameter application.
* That field was read into variable Y.TT.PAR.
```

```
FOR I.TT.PAR = 1 TO DCOUNT(Y.TT.PAR, VM)
  Y.TT.ACC.CAT = Y.TT.PAR<I.TT.PAR>
  IF Y.ACC.CAT = Y.TT.ACC.CAT THEN
    CAT.FOUNDED = 1
  END
NEXT I.TT.PAR
```

What's wrong:

- The number of values in a field will be calculated using DCOUNT at every iteration -- it can be done only once and put into a variable.
- We are looking for a value, not for a field -- hence should use:  
Y.TT.PAR<1, I.TT.PAR>
- It's a good idea to use BREAK and thus to leave the loop when the given value is found.
- Or -- use FIND instead of this loop.

Example 11:

```
* LOC.ID.ACC is account ID which is somehow relevant
* to a record in another application that we're in.
* This code gives an error when account record
* is not authorised.
* LOC.R.ACC is a record read from ACCOUNT$NAU file.
```

```
IF LOC.R.ACC<AC.RECORD.STATUS> EQ "INAU" THEN
  E = 'ACCOUNT.&:.NAU.RECORD.EXISTS':FM:LOC.ID.ACC
  RETURN
END
```

What's wrong:

- Status of unauthorised record might be not only INAU, but CNAU, RNAU, IHLD etc.
- Actually, if there's a record in \$NAU file, there's no need to check its status -- we can raise the error message at once.

Example 12:

```
* This code is part of VERSION-level routine.  
*  
  IF (R.NEW(TT.TE.AMOUNT.LOCAL.1) EQ '') OR \  
      (R.NEW(TT.TE.AMOUNT.LOCAL.1) EQ 0) THEN
```

What's wrong:

- Value of field is extracted from R.NEW twice.
- We can simply write:  
 IF NOT(R.NEW(TT.TE.AMOUNT.LOCAL.1))

Example 13:

```
* This code calculates local amount from amount  
* in foreign currency  
* and puts the result to appropriate local field.
```

```
LOC.FLAT.AMT = LOC.AMT.FCCY * LOC.CUR.EXH  
R.NEW(LOCAL.REF.FIELD)<1,LCY.AMT.POSN> = LOC.FLAT.AMT
```

What's wrong:

- Use EB.ROUND.AMOUNT T24 API subroutine to make amount not to look like, e.g., 9.999762.

Example 14:

```
  IF ((LOC.R.REC<MM.CATEGORY> = 21025 \  
      OR LOC.R.REC<MM.CATEGORY> = 21026 \  
      OR LOC.R.REC<MM.CATEGORY> = 21027 \  
      OR LOC.R.REC<MM.CATEGORY> = 21028 \  
      OR LOC.R.REC<MM.CATEGORY> = 21029) AND \  
      (LOC.REPORT.TYPE = 'B')) \  
      OR (LOC.REPORT.TYPE = 'P') THEN  
* Logic here
```

What's wrong:

- Again, the field CATEGORY is to be extracted once and put to a variable for future use.
- Constants feel better being parameterised -- put it to an external application, either core or local one.

Example 15:

```
IF LEN(LOC.TT.ID) < 2 THEN LOC.TT.ID = '000' : LOC.TT.ID
IF LEN(LOC.TT.ID) < 3 THEN LOC.TT.ID = '00' : LOC.TT.ID
IF LEN(LOC.TT.ID) < 4 THEN LOC.TT.ID = '0' : LOC.TT.ID
```

What's wrong:

- See FMT() Basic function.

(Here we see the code “that works”. I often hear “this code works, what’s wrong?” form this or that developer. My point is that even if it works, it shows the lack of knowledge – or lack of desire to learn. What if LOC.TT.ID was up to 100 characters in length?)

Example 16:

```
Y.SEL = 'SELECT FBNK.ACCOUNT WITH ALT.ACCT.ID EQ ':ID.ACC
```

What's wrong:

- Use OPF to get full file name instead of hard-coding “FBNK” prefix.
- No need for this SELECT at all -- we have concat file ALTERNATE.ACCOUNT.

About concat files: this is analog of indexing (supported at T24 application level). Personally, I prefer them to jBASE indexing. Links to concat files are programmed in application code (see somewhat misty explanations in I.RULES) or being set up using core application *EB.ALTERNATE.KEY*.

## 31 I-descriptors

Now we are a bit back into enquiry-related programming. Actually there is another place where we can program to make things easier in enquiry processing. It's called I-descriptors.


It doesn't mean that I-descriptors require programming in all cases. There are simple things that do not. Remember when we used EVAL clause in a SELECT?

```
jsh mb10 ~-->LIST FBNK.ACCOUNT$HIS EVAL  
"FIELD(@ID,',' ,2)"
```


We can create an I-descriptor that returns the second part of ID. To do it in non-T24 world one would use dictionary. In T24 we have to use *STANDARD.SELECTION* application. Enter *SS* record, go directly to the field 15.1 by typing its number, then type < to expand the MV block and enter the following:

R10 Model	Bank	STANDARD SELECTION FIELDS, INPUT	
FILE.NAME.....		ACCOUNT	
-----			
15.	1	USR.FIELD.NAME.	I.SECOND.PART
16.	1	USR.TYPE.....	I
17.	1.	1 USR.FIELD.NO	@ID[';' ,2,1]
18.	1.	1 USR.VAL.PROG	
19.	1	USR.CONVERSION.	
20.	1	USR.DISPLAY.FMT	2L
21.	1	USR.ALT.INDEX..	
22.	1.	1 USR.IDX.FILE	
23.	1	USR.INDEX.NULLS	
24.	1	USR.SINGLE.MULT	S
25.	1	USR.LANG.FIELD.	N

Now commit the record, see multiple messages “I-descriptor is incompatible with” something (TAFJ, I suspect), ignore them all, return into `jsh` prompt. Let’s see the corresponding dictionary entry:

```
jsh mb10 ~-->CT DICT FBNK.ACCOUNT I.SECOND.PART   
  
      I.SECOND.PART  
001 I  
002 @ID[';','2,1]  
003  
004 I.SECOND.PART  
005 2L  
006 S  
007  
008  
009  
010  
011  
012  
013  
014 @ID[';','2,1]
```

We can now use this field in `SELECT`, `LIST` etc:

```
jsh mb10 ~-->LIST FBNK.ACCOUNT$HIS I.SECOND.PART   
  
@ID..... I.SECOND.PART  
      38288;10 10  
      17736;1 1  
      38008;17 17  
      38008;21 21  
      38288;5 5  
      38008;5 5  
USD112100001;1 1  
      38296;9 9
```



You might ask: we already have `EVAL` to achieve that. But I-descriptor can be used as a selection criteria for an enquiry, as an enquiry fixed selection. Now let's try to create an I-descriptor with local routine attached to it. As an example, we can create I-descriptor that returns 1 if the record in question was created by current user and 0 if it was somebody else:

```
0001 SUBROUTINE IDESC.TEST(P.RET, P.INPUTTER)
0002 *-----
0003 * This routine returns 1 if the record was input
0004 * by current user, otherwise it returns 0.
0005 * Type of routine: I-descriptor.
0006 *-----
0007 $INSERT I.COMMON
0008 $INSERT I.EQUATE
0009
0010 P.RET = 0
0011
0012 IF FIELD(P.INPUTTER, '_', 2) EQ OPERATOR THEN
0013     P.RET = 1
0014 END
0015
0016 RETURN
0017 END
0018
```

Note that the 1<sup>st</sup> parameter is an output one. All others are input ones. Normally we pass some field or fields to such routine to proceed. See how it's attached to `STANDARD.SELECTION`:

```

R10 Model Bank          STANDARD SELECTION FIELDS, INPUT
FILE.NAME.....        FUNDS.TRANSFER
-----
15.  1  USR.FIELD.NAME.  I.MY.RECORD
16.  1  USR.TYPE.....   I
17.  1.  1  USR.FIELD.NO  SUBR('IDESC.TEST',INPUTTER)
18.  1.  1  USR.VAL.PROG
19.  1  USR.CONVERSION.
20.  1  USR.DISPLAY.FMT  1L
21.  1  USR.ALT.INDEX..
22.  1.  1  USR.IDX.FILE
23.  1  USR.INDEX.NULLS
24.  1  USR.SINGLE.MULT  S
25.  1  USR.LANG.FIELD.  N

```

Try yourself to create an enquiry which uses this I-descriptor.

One more note: you might find fields with type“J” in SS records. They are so-called “J-descriptors” though there’s no such term in jBASE itself. They are simply I-descriptors which call a standard routine to fetch a field from another application, e.g.:


```

R10 Model Bank          STANDARD SELECTION FIELDS, INPUT
FILE.NAME.....        ACCOUNT
-----
1.   9  SYS.FIELD.NAME.  SECTOR
2.   9  SYS.TYPE.....   J
3.   9.  1  SYS.FIELD.NO  CUSTOMER.NO>CUSTOMER>SECTOR

```

Let’s see dictionary entry for it:

```

jsh mb10 ~-->CT DICT FBNK.ACCOUNT SECTOR 
      SECTOR
001 I
002 CUSTOMER.NO; SUBR("ENQ.TRANS","CUSTOMER", @1, "SECTOR")
003
004 SECTOR
005 4R
006 S
007
008
009
010
011
012
013
014 CUSTOMER.NO; SUBR("ENQ.TRANS","CUSTOMER", @1, "SECTOR")

```

The syntax in *SS* record is quite evident so if you need something like that it's easy.

## 32 COB programming

**COB** stands for “close of business” when the current work day is over and a sequence of “jobs” is being run to produce such things as daily reports, account accruals etc. The COB programming idea in T24 is that you need to write so-called “multi-threaded” routines to be run during COB. Actually it has nothing to do with threading – it's just possibility for several sessions to simultaneously proceed the task given. As a result, performance gain is achieved – even if you run 2-3 “threads” (also called “agents”) on a dual-core PC processor.

Again, terminology might be a bit confusing. “COB agents” have nothing to do with “jBASE” agent which is used for Browser T24 client. COB agents are launched using `tSM` and `tSA` programs from `jsh` prompt. There is

“phantom” mode and “debug” mode; I’d recommend to use the former only when everything is absolutely stable (which is near to impossible in real life).

See T24 manuals for more info; here I’d like only to produce a very simple example of COB job (or at least I believe that it will remain simple).

What we should not forget when we write routines for COB job?

- That NS (“Non-stop” module) might be installed and new transactions may be input during COB.
- Even if it’s not – that users are able to login to see records.
- That `F.WRITE` doesn’t cache writes (and issues immediate `WRITES`) so no need in calling `JOURNAL.UPDATE`.
- That parallel agent sessions shouldn’t be locking the same record, otherwise the performance gain goes down the drain.
- That in multi-company environment COB batches (and therefore jobs) are run separately for each company so you need to decide whether to run your job for each company or the architecture of your solution assumes that you need to run it only once.
- That all area might be restored after serious crash so if you have some work files – put them to `bnk.data` rather than keep in `bnk.run` like some people often do. And – in case of outward interfaces – take care about output files so the information wouldn’t be duplicated.

One note for `JOURNAL.UPDATE`: it’s a core subroutine that is not described in manuals but is sometimes necessary in local code. When you work inside T24 transaction boundary (e.g. in a `VERSION` routine) you don’t need it. Moreover, if you use it there you might ruin everything. Like: you use `F.WRITE` to some local application or work file to reflect something from current transaction, then use `JOURNAL.UPDATE`, then user rejects the main transaction but your supplementary data is already committed (as well as some pending writes already made by the core).

You can use `JOURNAL.UPDATE` in a “mainline” routine and in a routine that is invoked upon “V”-function in “W”-type application (in latter case when such routine is being invoked the main T24 transaction is already finished). If you don’t use `JOURNAL.UPDATE` in that 2 cases – all changed that were done using `F.WRITE` or `F.DELETE` will not be saved. Sometimes you need to issue `JOURNAL.UPDATE` after certain number of `F.WRITES` – otherwise T24 write cache will be exhausted and you’ll get fatal error like “\*\* FATAL ERROR IN (I\_IO.ROUTINES) \*\* FILE I/O CACHE EXCEEDS nnnn RECORDS”. By the way, if you get this message in T24 core and “nnnn” is equal to 9999, write to Temenos Helpdesk immediately (9999 is the maximum cache size that can be specified in *SPF*).

OK, back to COB example. Let’s program a simple COB report. It will be based on *ACCOUNT* application. Why can’t we use an enquiry to output all we need? Well, in our Model Bank *ACCOUNT* is not very big but in real life it might be; and enquiry can be prepared by only one of agents and we’re talking about performance here.

What we need is one insert file (`I.TEST.REP.COMMON`):

```

0001 *-----
0002 *
0003 * [Author], [date]
0004 *
0005 * Defines named common area which holds all initialized
0006 * variables for use by the job.
0007 *-----
0008 *
0009 * Modifications :
0010 *
0011 * DD/MM/YYYY, [author] - [details of change]
0012 *-----
0013
0014     COM /TEST.REP.COM/ FN$TEST.REP.OUTPUT,
0015                          F$TEST.REP.OUTPUT,
0016                          FN$ACCOUNT,
0017                          F$ACCOUNT

```

...and 4 routines. I'll describe them in the order of their processing.

Name prefix (TEST.REP) should be the same for 3 following routines. Suffixes are: .SELECT, .LOAD and for so-called "record" routine there's no suffix.

```
0001 SUBROUTINE TEST.REP.SELECT
0002 *-----
0003 * [Author], [date]
0004 *
0005 * Mandatory process to build the "key only" file that
the main processes will
0006 * handle. That file contains all the IDs which are to
be processed.
0007 *-----
0008 *
0009 * Modifications :
0010 *
0011 * DD/MM/YYYY, [author] - [details of change]
0012 *-----
0013 *
0014 $INSERT I_COMMON
0015 $INSERT I_EQUATE
0016 $INSERT I_TEST.REP.COMMON
0017
```

Usual header. Note I\_TEST.REP.COMMON – it's our common area that has to be included to all but one source file.

```
0018 * Clear temporary file
0019
0020 CALL EB.CLEAR.FILE(FN$TEST.REP.OUTPUT,
F$TEST.REP.OUTPUT)
0021 PRINT FN$TEST.REP.OUTPUT : ' has been cleared'
0022
```

Self-explanatory stuff...

```
0023 * Main SELECT
0024
0025 V.SELECT.CMD = 'SELECT ': FN$ACCOUNT
0026 V.SELECT.CMD := ' WITH CATEGORY EQ 1001 1005 2001
3101 3102 3103 4001 5001'
0027 V.SELECT.CMD := ' AND WORKING.BALANCE GT 0'
0028 V.SELECT.CMD := ' BY CATEGORY'
0029 V.TEST.REP.LIST = ''
0030 CALL EB.READLIST(V.SELECT.CMD, V.TEST.REP.LIST, '',
'', '')
0031
```

Here we proceed with SELECT to prepare a list for processing.

```
0032 * Build "key only" file
0033
0034 CALL BATCH.BUILD.LIST('', V.TEST.REP.LIST)
0035
0036 RETURN
0037
0038 END
0039
```

Finally we pass this list to the core.

“Load” routine:

```
0001 SUBROUTINE TEST.REP.LOAD
0002 *-----
0003 *
0004 * [Author], [date]
0005 *
0006 * This mandatory process is required to set up
0007 * the common area used by the process.
0008 *-----
0009 *
0010 * Modifications :
0011 *
0012 * DD/MM/YYYY, [author] - [details of change]
0013 *-----
0014 *
0015 $INSERT I.COMMON
0016 $INSERT I.EQUATE
0017 $INSERT I.TEST.REP.COMMON
0018
```

```
0019 * Main
0020
0021   GOSUB OPEN.FILES
0022
0023   RETURN
0024
```

Nothing to comment so far...



```

0025 * Subroutines
0026
0027 *-----
0028 OPEN.FILES:
0029
0030 * This hashed file contains all the report lines which
are to be sent out
0031
0032 FN$TEST.REP.OUTPUT = 'F.TEST.REP.OUTPUT'
0033 F$TEST.REP.OUTPUT = ''
0034 CALL OPF(FN$TEST.REP.OUTPUT, F$TEST.REP.OUTPUT)
0035
0036 FN$ACCOUNT = 'F.ACCOUNT'
0037 F$ACCOUNT = ''
0038 CALL OPF(FN$ACCOUNT, F$ACCOUNT)
0039
0040 *-----
0041
0042 RETURN
0043
0044 END
0045

```

This routine is triggered before a “record” routine proceeds with an entry from the list file (e.g. F.JOB.LIST.1) and sets up the necessary environment.

Now, “record” routine:

```
0001 SUBROUTINE TEST.REP(P.NEXT.ID)
0002 *-----
0003 *
0004 * [Author], [date]
0005 *
0006 * This is the main program, will be invoked for every
0007 * ID in the list file, generated by the system at
0008 * .SELECT stage.
0009 *-----
0010 *
0011 * Related programs for multi-threading are :
0012 * I_TEST.REP.COMMON - common area (INSERT-file)
0013 * TEST.REP.LOAD - sets up the common area used by
0014 * all threads
0015 * TEST.REP.SELECT - builds the "key only" file
0016 * TEST.REP.OUT - next job, outputs the data from
0017 * hashed file to a flat file
```

```
0018 *-----
0019 *
0020 * Modifications :
0021 *
0022 * DD/MM/YYYY, [author] - [details of change]
0023 *-----
0024 *
0025 $INSERT I.COMMON
0026 $INSERT I.EQUATE
0027 $INSERT I.TEST.REP.COMMON
0028 $INSERT I.F.ACCOUNT
0029
```

```

0030 * Main
0031
0032 GOSUB PROCESSING
0033 GOSUB WRITE.TO.HASHED.FILE
0034
0035 RETURN
0036
0037 * Subroutines
0038

```

Quite usual main section...

```

0039 *-----
0040 PROCESSING:
0041
0042 V.ERR = ''
0043 CALL F.READ(FN$ACCOUNT, P.NEXT.ID, R.ACCOUNT,
F$ACCOUNT, V.ERR)
0044
0045 IF V.ERR THEN
0046     TEXT = 'ACCOUNT RECORD (&) COULD NOT BE READ' :FM:
P.NEXT.ID
0047     CALL FATAL.ERROR(SYSTEM(40))
0048 END
0049
0050 V.LINE.OUT = P.NEXT.ID : ' ' : \
0051     R.ACCOUNT<AC.ACCOUNT.TITLE.1> : ' ' : \
0052     R.ACCOUNT<AC.CURRENCY> : ' ' : \
0053     FMT(R.ACCOUNT<AC.WORKING.BALANCE>, "19R")
0054
0055 RETURN
0056

```

Here we prepare the line which will be written to our report...

```

0057 *-----
0058 WRITE.TO.HASHED.FILE:
0059
0060 * Writing, ID is the same as ACCOUNT ID
0061
0062   CALL F.WRITE(FN$TEST.REP.OUTPUT, P.NEXT.ID,
V.LINE.OUT)
0063
0064   RETURN
0065
0066 END
0067

```

...and here we write it to hashed file.

And – finally – the output routine that will grab all info from a hashed file and then output it to a flat file:

```

0001 SUBROUTINE TEST.REP.OUT
0002 *-----
0003 * [Author], [date]
0004 *
0005 * Process to output the data stored in the
0006 * F[xxx].TEST.REP.OUTPUT file to a text file
0007 * after all multi-threading has finished.
0008 *-----
0009 *
0010 * Modifications :
0011 *
0012 * DD/MM/YYYY, [author] - [details of change]
0013 *-----
0014 *
0015 $INSERT I.COMMON
0016 $INSERT I.EQUATE
0017

```

```
0018 GOSUB INITIALIZATION
0019
0020 * This hashed file contains all the report lines which
are to be sent out.
0021 * Select all records sorted by deal reference
0022
0023 V.SELECT.COMD = 'SSELECT ': FN.TEST.REP.OUTPUT
0024 V.LIST = ''
0025 CALL EB.READLIST(V.SELECT.COMD, V.LIST, '', V.QTY, '')
0026
0027 IF NOT(V.LIST) THEN RETURN
0028
```

```
0029 * Create output file in the temporary directory
0030
0031 OPENSEQ V.TEMP.DIR, V.OUT.FILE.NAME TO F.TEMP.FILE
THEN
0032     WEOFSEQ F.TEMP.FILE
0033 END ELSE
0034     CREATE F.TEMP.FILE ELSE
0035         TEXT = 'OUTPUT FILE CREATION ERROR'
0036         CALL FATAL.ERROR(SYSTEM(40))
0037     END
0038 END
0039
```

```

0040 FOR V.I = 1 TO V.QTY
0041
0042     V.ID = V.LIST<V.I>
0043     V.LINE = ''
0044     V.ERROR = ''
0045     CALL F.READ(FN.TEST.REP.OUTPUT, V.ID, V.LINE,
F.TEST.REP.OUTPUT, V.ERROR)
0046
0047     IF V.ERROR THEN
0048         TEXT = 'INPUT FILE - READ ERROR'
0049         CALL FATAL.ERROR(SYSTEM(40))
0050     END
0051
0052     WRITESEQ V.LINE TO F.TEMP.FILE ELSE
0053         TEXT = 'OUTPUT FILE - WRITE ERROR'
0054         CALL FATAL.ERROR(SYSTEM(40))
0055     END
0056
0057 NEXT V.I
0058

```

```

0059 CLOSESEQ F.TEMP.FILE
0060
0061 * Move file to interface directory
0062
0063 IF SYSTEM(1017)[1,3] EQ 'WIN' THEN
0064     V.CMD = "DOS /c 'MOVE /Y " :
''':V.TEMP.DIR:'\':V.OUT.FILE.NAME :'" "': V.OUT.DIR
:'''':'"'
0065 END ELSE
0066     V.CMD = "SH -c 'mv -f " :''':V.TEMP.DIR :
'/':V.OUT.FILE.NAME :'" "': V.OUT.DIR :'''':'"'
0067 END
0068
0069 EXECUTE V.CMD
0070
0071 RETURN
0072

```

```

0073 * Subroutines
0074
0075 *-----
0076 INITIALIZATION:
0077
0078   V.TEMP.DIR = '&TEMP&'
0079
0080 * Open files
0081
0082   FN.TEST.REP.OUTPUT = 'F.TEST.REP.OUTPUT'
0083   F.TEST.REP.OUTPUT = ''
0084   CALL OPF(FN.TEST.REP.OUTPUT, F.TEST.REP.OUTPUT)
0085
0086 * Compose output file name
0087
0088   V.TIME.NOW = OCONV(TIME(), 'MTS')
0089   CONVERT ':' TO '.' IN V.TIME.NOW
0090   V.DATE.NOW = FMT(TODAY, 'R####.##.##')
0091   V.OUT.FILE.NAME = 'TEST_REP_': V.DATE.NOW : '_' :
V.TIME.NOW
0092
0093   V.OUT.DIR = '../bnk.interface'
0094
0095   RETURN
0096
0097 END
0098

```

Hope that comments in the code are quite clear.

Now we need to set up some records:

```

R10 Model Bank      PROGRAM FILE, INPUT
PROGRAM            TEST.REP
-----
1 TYPE..... B
2.  1 GB SCREEN.TITLE MULTI-THREADED REPORT
3 ADDITIONAL.INFO...
4.  1 BATCH.JOB.....
5 PRODUCT..... EB

```

```

R10 Model Bank      PROGRAM FILE, INPUT
PROGRAM            TEST.REP.OUT
-----
1 TYPE..... B
2.  1 GB SCREEN.TITLE TEST.REP REPORT OUTPUT
3 ADDITIONAL.INFO... R
4.  1 BATCH.JOB..... TEST.REP.OUT
5 PRODUCT..... EB

```

You see that the field BATCH.JOB is not populated in TEST.REP record. This (or the value @BATCH.JOB.CONTROL) means that it's standard multi-threaded job. TEST.REP.OUT is not multi-threaded since we're going to write to a flat file there which normally requires a single session.




R10 Model Bank	BATCH ENTRY, INPUT	
BATCH PROCESS	BNK/TEST.REP	
-----		
1	BATCH.STAGE.....	
2	DEFAULT.PRINTER...	
3	PROCESS.STATUS....	
4	BATCH.ENVIRONMENT.	F
5	DEPARTMENT.CODE...	
6.	1 JOB.NAME.....	TEST.REP            MULTI-THREADED REPORT
7.	1. 1 VERIFICATION	
8.	1 FREQUENCY.....	D
	...	
6.	2 JOB.NAME.....	TEST.REP.OUT      TEST.REP REPORT OUTPUT
7.	2. 1 VERIFICATION	TEST.REP            MULTI-THREADED REPORT
8.	2 FREQUENCY.....	D

We leave the field `BATCH.STAGE` empty intentionally to be able firstly to test our functionality as an online service without the need to run COB (which might be not an easy task on Model Bank – usually there are numerous problems with core jobs that wouldn't be easy for you to overcome).

As soon as testing finishes, field `BATCH.STAGE` is to be populated with something like `R001` to make it run at “Reporting” COB stage. Then the *BATCH* record is to be repeated for every company that has its own *ACCOUNT* file (so IDs will be `SG1/TEST.REP`, `EU1/TEST.REP` and `MF1/TEST.REP`). Companies `MF2` and `MF3` share *ACCOUNT* file with company `MF1`. We can see it here:

```

jsh mb10 ~-->LIST F.COMPANY MNEMONIC FINANCIAL.MNE 
@ID..... MNEMONIC FINANCIAL.MNE

SG0010001 SG1 SG1
GB0010003 MF2 MF1
EU0010001 EU1 EU1
GB0010002 MF1 MF1
GB0010004 MF3 MF1
GB0010001 BNK BNK

```

To be able to run our task as an online service we need to create the following records:

```

R10 Model Bank TSA.WORKLOAD.PROFILE, INPUT

WORKLOAD.PROFILE TEST.REP
-----
1 DESCRIPTION..... FOR TEST REPORT
2. 1 TIME.....
3. 1 AGENTS.REQUIRED 2

```

Record ID can be any (or you can use any other record that has the number of agents the same as you need... for example, TWO... no, it has 4 agents... never trust a name...). And:

```

R10 Model Bank TSA.SERVICE, INPUT


SERVICE BNK/TEST.REP
-----
1. 1 DESCRIPTION.... TEST REPORT ONLINE
2. 1 SERVER.NAME....
3. 1 WORK.PROFILE... TEST.REP FOR TEST REPORT
4. 1 SERVER.STATUS..
5 USER..... INPUTTER INPUTTER
6 SERVICE.CONTROL...

```

Again, ID can be any. To start our testing (I assume that routines are already compiled) put the value `START` into field `SERVICE.CONTROL` both in our new record and in the record `TSM`. Then open another Telnet session so we have 2 of them. In one of them at `jsh` prompt type `START.TSM -DEBUG` (yes, that's `DEBUG` mode) and see what's there on the screen:

```
...
Manually launch tSA 14 SMS.OUT
Manually launch tSA 15 SMS.OUT
Manually launch tSA 16 BNK/TEST.REP
Manually launch tSA 17 BNK/TEST.REP
...
```

In the other session at `jsh` prompt type:

```
jsh mb10 ~-->tSA 16 
...
_BNK/TEST.REP_TEST.REP_16_23 SEP 2010_12:15:50_Standard
multi-thread job
_BNK/TEST.REP_TEST.REP_16_23 SEP 2010_12:15:50.Calling load
routine
_BNK/TEST.REP_TEST.REP_16_23 SEP 2010_12:15:50.Fatal error
in OPF error NO FILE.CONTROL RECORD - F.TEST.REP.OUTPUT ,
MNEMONIC =
...
jsh mb10 ~-->
```

Yes, we've forgotten to create a work file. I used this situation to show how `FATAL.ERROR` works. It gives an error message, in addition there's an application `EB.EOD.ERROR` where we can see it:


R10 Model Bank	EB.EOD.ERROR SEE
EB.EOD.ERROR.ID...	GB0010001.20100105
-----	
1. 1 TIME.DATE.....	12:15:50 23 SEP 2010
2. 1. 1 DESCRIPTION.	NO FILE.CONTROL RECORD - F.TEST.REP.OUTPUT , MNEMONIC =
3. 1 APPLICATION.ID.	BNK/TEST.REP-TEST.REP
4. 1 ROUTINE.....	OPF
6. 1. 1 DETAIL.KEY..	156070180444150.00
8. 1 FIX.REQUIRED...	YES
23. 1 INPUTTER.....	OPF
24. 1 DATE.TIME.....	23 SEP 2010 12:15
25 AUTHORISER.....	FATAL.ERROR
26 CO.CODE.....	GB-001-0001 R10 Model Bank

If fatal error happens during COB our reaction depends on the situation. If the error was raised by some record processing that doesn't touch other records (e.g. problems with particular contract) – we can in most cases continue and take care about that particular record later. There could be cases when we'll have to restore pre-COB backup and run COB again (which is of course the last thing that is considered). In case of any doubt contact Temenos Helpdesk.

Here we're going to create the necessary file and try again. We don't even need to stop tSM.

A work file that we need can easily be created. Despite the fact that Temenos standards require a template to be written for every data file, we really don't need it. We still need *FILE.CONTROL* records so the file can be opened with OPF. And – *FILE.CONTROL* record will be useful for deployment of all development to another area via *DL.DEFINE* (since when we restore *FILE.CONTROL* record, data file will be created automatically). And we need this file to be FIN because *ACCOUNT* is.

We had already created *FILE.CONTROL* record for local application *FT.CREATION.METHOD*. So:

```
jsh mb10 ~-->COPY FROM F.FILE.CONTROL
FT.CREATION.METHOD,TEST.REP.OUTPUT 
1 records copied
```

Then it makes sense to correct fields 1 and 2 with file description and EB accordingly (“EB” stands for “core” – there’s no product code for local developments). Then the procedure already known to you – CREATE.FILES. Then – input START to TSA.SERVICE record BNK/TEST.REP. After some time tSM screen assigns the number for our agent:

```
...
Manually launch tSA 16 BNK/TEST.REP
Manually launch tSA 17 BNK/TEST.REP
```

Let’s try number 17 now:



```
...
FBNK.TEST.REP.OUTPUT has been cleared
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_SELECT
FBNK.ACCOUNT WITH CATEGORY EQ 1001 1005 2001 3101 3102 3103
4001 5001 AND WORKING.BALANCE GT 0 BY CATEGORY Selected=215
time=1secs
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_List starting
from 0 keys processed so far 0
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_Building from
list passed
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_Control list..
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_Using list file
F.JOB.LIST.1
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_Control list
processing 112 215
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_SELECT
F.JOB.LIST.1 SAMPLE 100000 Selected=101 time=0secs
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_SELECT
F.JOB.LIST.1 SAMPLE 100000 Selected=91 time=0secs
...
```

```

...
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_SELECT
F.JOB.LIST.1 SAMPLE 100000 Selected=17 time=0secs
_BNK/TEST.REP_TEST.REP_17_23 SEP 2010_13:05:55_SELECT
F.JOB.LIST.1 SAMPLE 100000 Selected=0 time=0secs
</job>
<job name = TEST.REP.OUT>
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Single
Thread routine TEST.REP.OUT
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Starting
job
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Allocating
List File for BNK/TEST.REP-TEST.REP.OUT-2
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Updating
the Locking with BNK/TEST.REP-TEST.REP.OUT-2 and
F.JOB.LIST.1
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Using list
file F.JOB.LIST.1
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_Control
list processing 1 1
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:05:55_SSELECT
FBNK.TEST.REP.OUTPUT Selected=215 time=0secs
_BNK/TEST.REP_TEST.REP.OUT_17_23 SEP 2010_13:06:09_SELECT
F.JOB.LIST.1 SAMPLE 100000 Selected=0 time=0secs
</job>
</process>
</service>
Agent stopped


```

You also have all this output in a file in &COMO& subdirectory of `bnk.run`.  
And – see report file:

```
jsh mb10 ~-->SELECT ../bnk.interface LIKE TEST...   
  
1 Records selected  
  
> CT ../bnk.interface   
  
TEST_REP_2010.01.05_13.05.55  
001 10499 WILLIAM GATES USD 9382000  
002 10596 TED TURNER USD 7582.19  
003 10693 MICHAEL DELL USD 88452734.12  
004 10707 MICHAEL DELL CHF 11710  
005 10812 JAY WALKER EUR 28440.62  
006 10944 Warner Buffet USD USD 79981.66  
007 10968 Warner Buffet EUR EUR 15730.22  
008 11018 COCA-COLA EUR 2618274.11  
009 11029 COCA-COLA GBP 6011527.99  
010 11096 Delta Caroline USD USD 295177.78  
...
```

Or see it in your favourite viewer/editor.

### 33 Linux

 here's a lot of places where you can insert your hooks in T24 – delivery (very large topic), accounting hook, charges/commissions hook... It will take a lot of time to describe it all. Maybe in Volume 2 I'll do that... But for time being I'm in a situation of having Windows 7 home at my laptop and there are certain difficulties in running jBASE (shared memory initialisation even if I run jsh from Admin account). So I've decided to go to a chapter that was intended to be covered much later. It's: T24 under Linux.

Which Linux distribution to take? I know that Redhat is the only one officially supported by Temenos and there's CentOS 5 which is the closest one to Redhat. But (I afraid I might be criticized here) my Linux of choice is Ubuntu.

Firstly we need to get 64-bit distribution iso file. It's quite easy – you can download it at [ununtu.org](http://ununtu.org). I understand that installation of OS that might be new to you along with your current one is quite a stress, so I'd suggest to install Ubuntu to a virtual machine.

Be sure you have Model Bank and TAFC for Linux (the latter equipped with valid license key) before you start.

In my example I'll use VmWare player 3.1.0 and downloaded image file `ubuntu-10.10-desktop-amd64.iso`. Start the Player, choose option "Create a New Virtual Machine", choose option "Installer disc image file (iso)", find your file, click "Next", input your name etc. Important is the user name – you'll have to put all your files into the directory `/home/user_name`. I'll make it `t24` to make things easier.

Everything else you can leave by default. When VM is created, put jBASE distribution to directory `/home/t24` and extract it in terminal:

```
t24@ubuntu:~$ gzip -cd TAFC_R10_GA_Linux.tar.gz | tar -xvf -
```



If you're not able to copy/paste between your host and guest OS – use USB flash drive to transfer files.


Extraction of TAFC package will create subdirectory `R10` which I'd better rename to `tafc`. Then – extract Model Bank using the same command as above. You'll have then `bnk` subdirectory.

In terminal window change the directory to `tafc/config`, then type: `sudo gedit system.properties`. Put your jBASE licence code at the very end of this file. Save it.


Then go to `/home/t24/bnk/bnk.run` directory, edit `.profile`. Put there `TAFC_HOME` as `/home/temenos/tafc`. Now you're almost ready to run T24 in classic mode.




To do that (at least to try) type in terminal (assuming that you're in `bnk.run` directory):

```
t24@ubuntu:~/bnk/bnk.run$ ./profile   
/home/t24/tafc/bin/jpqn: error while loading shared  
libraries: libcrypto.so.6 etc...
```


How to correct that:



```
t24@ubuntu:~/bnk/bnk.run$ sudo ln -s  
/usr/lib/libcrypto.so.0.9.8 /usr/lib/libcrypto.so.6 
```

Anticipating another error, it's a good idea to run the following command as well:


```
t24@ubuntu:~/bnk/bnk.run$ sudo ln -s  
/usr/lib/libssl.so.0.9.8 /usr/lib/libssl.so.6 
```


Finally:

```
t24@ubuntu:~/bnk/bnk.run$ ./profile   
START GLOBUS Y/N   
%  
[ 417 ] File /home/t24/tafc/tmp/jutil.ctrl]D created, type =  
J4  
[ 417 ] File /home/t24/tafc/tmp/jutil.ctrl created, type =  
J4  
jsh t24 ~-->
```

We are now able to login to T24. Keep in mind that function keys mapping doesn't work here so use  -U  etc.

If we need to compile Basic sources (of course we do), we need also to


install open ksh (press -D first to return to bash):

```
t24@ubuntu:~/bnk/bnk.run$ sudo apt-get install ksh 
```

(Thanks to people from jBASE Google group for that hint. Before getting it I've simply linked sh to ksh which indeed wasn't very wise.)

In case of problems firstly run the command ‘‘sudo apt-get update’’.

If – when cataloging your routine – you get the error message ‘‘cannot find -lncurses’’ – install the following packages:


```
t24@ubuntu:~/bnk/bnk.run$ sudo apt-get install libncurses5  
libncurses5-dev 
```

After that you get quite working T24 system. If you're going to run COBs there. don't forget to increase number of opened files for your user in `/etc/security/limits.conf`:

```
t24      hard    nofile   4096  
t24      soft    nofile   4096
```

If you wish, you can then install java and jBoss but let's try to log in to this system from the host OS, i.e. Windows.

Firstly get IP address of the virtual machine (command `ifconfig`). Then – use some SSH client (e.g. PuTTY) to connect to this address. Before doing that, you need to install:

```
t24@ubuntu:~/bnk/bnk.run$ sudo apt-get install  
openssh-server 
```


If packages names change at the time when you're trying to do that – simply type `sshd` at terminal prompt and it will tell you what to do.

Connect via PuTTY, say “yes” to certificate-related message, then – login as t24 with your Ubuntu password. We are now at the home directory, to launch T24 we need to change our current directory to `bnk/bnk.run` and then type `./profile`.

We’ll leave this virtual machine as it is and use it as a server. The only thing we need for it to be a server – the jBASE agent to be run there. Simply copy `profile` to `jbase_agent.sh` and amend the last line the following way:

```
$TAFC_HOME/bin/jbase_agent
```


Then – run it in a terminal:

```
t24@ubuntu:~/bnk/bnk.run$ ./jbase_agent.sh   
(4915|140158758917920) NOTICE starting up jAgent,  
Process Per Connection mode, listening on port 20002,  
SocketAcceptor.h +107
```

Now you can set up your Windows jBoss to connect to this agent. To do this just change host and port in `t24-ds.xml` (see above in “Browser client...” chapter).

But the thing which might be much more interesting for you is awaiting us – Java/jRemote.

## 34 Java and jRemote

o be honest with you, I’m not a big fan of Java. The purpose of this chapter is to show the possibilities that you have; to use or not to use them is up to you.

We’ll use Linux virtual machine with running `jbase_agent` as a server and Windows as a client.

We can use our Java SDK 1.6 for it.

Create a new directory in Windows space and put there the following files:

CallSysCall.java, contents (please note that lines are quite long and might wrap; check semicolons for line ends):

```
// JCA outbound example (Java)
//
// Author: Alexander Demin (ademin@temenos.com)
// small changes KZM
// Copyright: Temenos 2009

import com.jbase.jremote.DefaultJConnectionFactory;
import com.jbase.jremote.JConnectionFactory;
import com.jbase.jremote.JDynArray;
import com.jbase.jremote.JRemoteException;
import com.jbase.jremote.JExecuteResults;
import java.util.Properties;
import java.io.ByteArrayOutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.io.UnsupportedEncodingException;

public class CallSysCall {

    private JConnectionFactory factory = null;
    public CallSysCall(JConnectionFactory f) {
        factory = f;
    }
    ...
}
```

```

...
public void perform(String cmd) {
try {

    Properties prop = new Properties();
    prop.setProperty("allow_input", "true");
    JConnection c = factory.getConnection(null, null, prop);

    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    Writer writer = new OutputStreamWriter(bos, "UTF-8");
    c.setTerminalOutputWriter(writer);

    JExecuteResults results = c.execute(cmd);

    if (results != null) {

        JDynArray resarray = new JDynArray();
        resarray = results.getCapturingVar();

        int nattr = resarray.getNumberOfAttributes();
        for(int a = 1; a <= nattr; a++)
        {
            System.out.println(resarray.get(a));
        }
    }

    c.close();

} catch (UnsupportedEncodingException e) {
    System.out.println("Error creating OutputStreamWriter.");
} catch (JRemoteException e) {
    e.printStackTrace();
}
}
}
...

```

```

...
public static void main(String[] args) {
    DefaultJConnectionFactory factory = new
DefaultJConnectionFactory();
    // optional host
    String host = "127.0.0.1";
    if (args != null && args.length > 0) {
        host = args[0];
    }
    factory.setHost(host);
    // optional port
    int port = 9494;
    if (args != null && args.length > 1) {
        port = Integer.valueOf(args[1]);
    }
    factory.setPort(port);

    String extcmd = "jdiag";
    if (args != null && args.length > 2) {
        extcmd = args[2];
    }

    CallSysCall example = new CallSysCall(factory);
    example.perform(extcmd);
}
}

```

Now we need to compile it. Put the following commands to cmd file and run it:

```

set PATH=c:\temenos\jdk1.6.0_17\bin;%PATH%
set CLASSPATH=.\jremote.jar;%CLASSPATH%
javac CallSysCall.java

```

Of course firstly copy to current directory file `jremote.jar` from TAFC subdirectory `java\lib` or refer to it using the full path.

Execute java class using cmd file with following contents:

```
set PATH=c:\temenos\jdk1.6.0_17\bin;%PATH%
set CLASSPATH=.\jremote.jar;%CLASSPATH%
java CallSysCall 192.168.111.132 20002 "WHERE"
```

Result:

```
[H[2J Port Device Account PID Command
*1000 3 t24 4964 jbase_agent
      WHERE
```

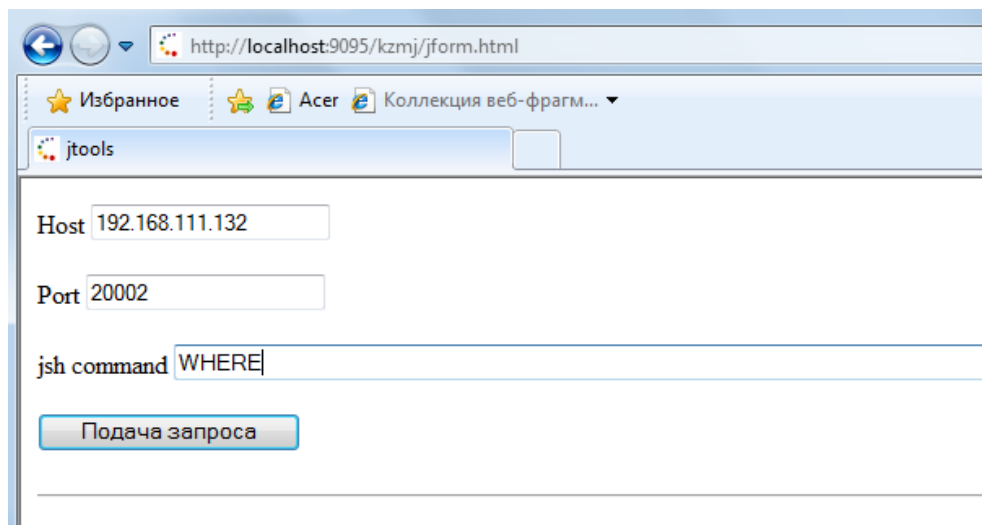
You can amend the command to, say, ‘‘LIST F.SPF’’. The result will be the same as you expected it to be:

```
@ID..... SYSTEM
@ID..... SYSTEM
SYSTEM.SPEC..... SYSTEM
RUN.DATE..... 20100120
SITE.NAME..... R10 Model Bank
OP.MODE..... 0
OP.CONSOLE.....
MAIN.ACCOUNT..... ../bnk.data
BACKUP.CYCLE.1.....
BACKUP.CYCLE.2.....
CURRENT.RELEASE..... R10.000
HIST.LIFE..... 1
ALL.PG.INC.....
CACHE.EXPIRY.....
ENQ.PAGE.LIMIT..... 200
RECV.DATE.TIME.....
SYS.BACKUP.MODE..... TAPE
HOLD.BATCH.OUTPUT.... Y
MICROFICHE.OUTPUT.... N
...
```

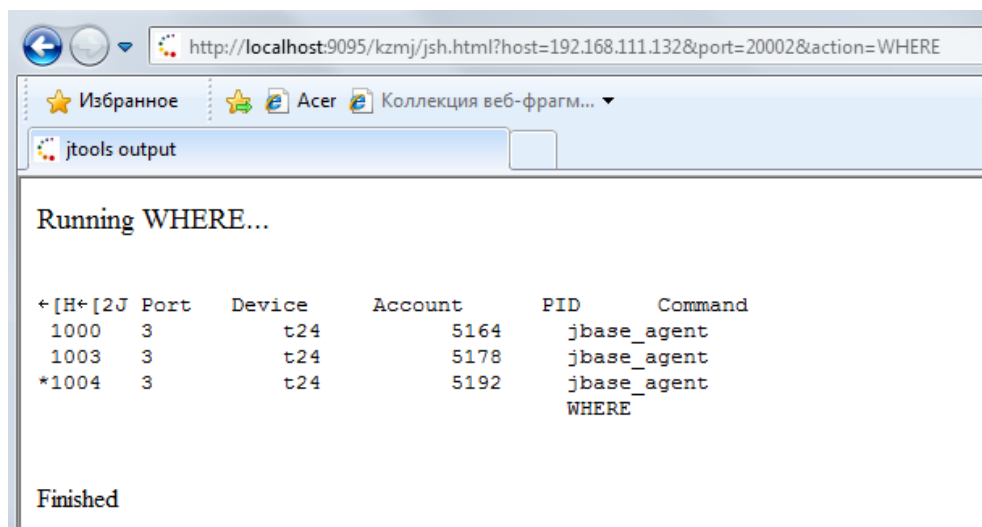
You can also call a Basic function using jRemote, open and read jBASE file. (And write too but be very careful!)

To make things look nicer I even went as far as to download Java servlet example and to make the output of jRemote appear in web browser window. I've used jBoss to deploy this servlet. See screenshots:

Request:

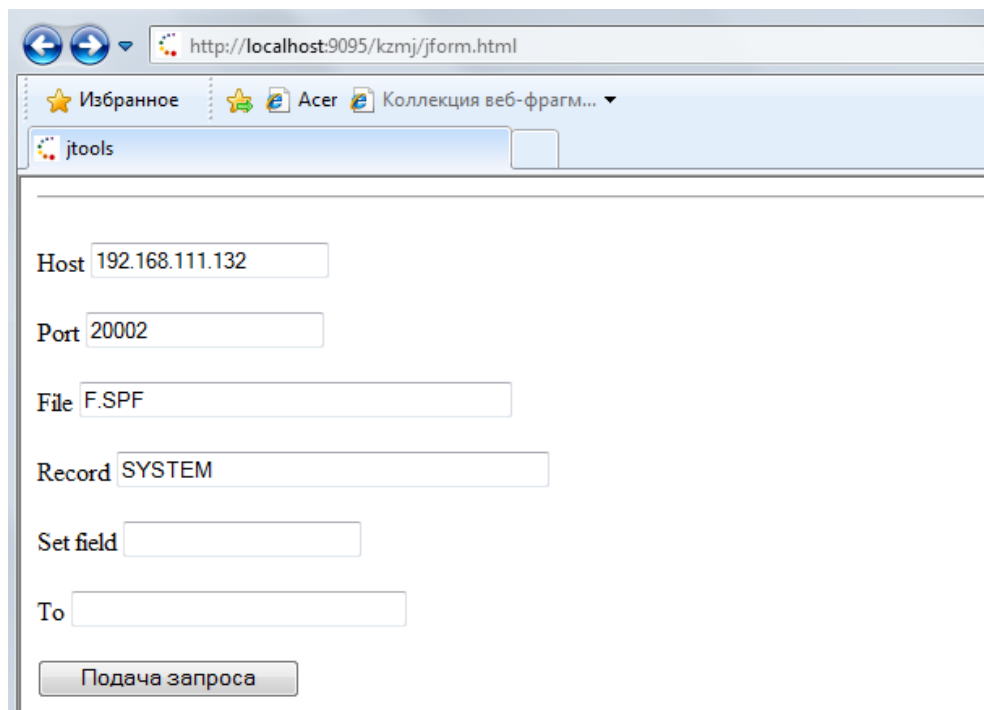


Result:





Request:



http://localhost:9095/kzmj/jform.html

Избранное Acer Коллекция веб-фрагм...

jtools

Host

Port

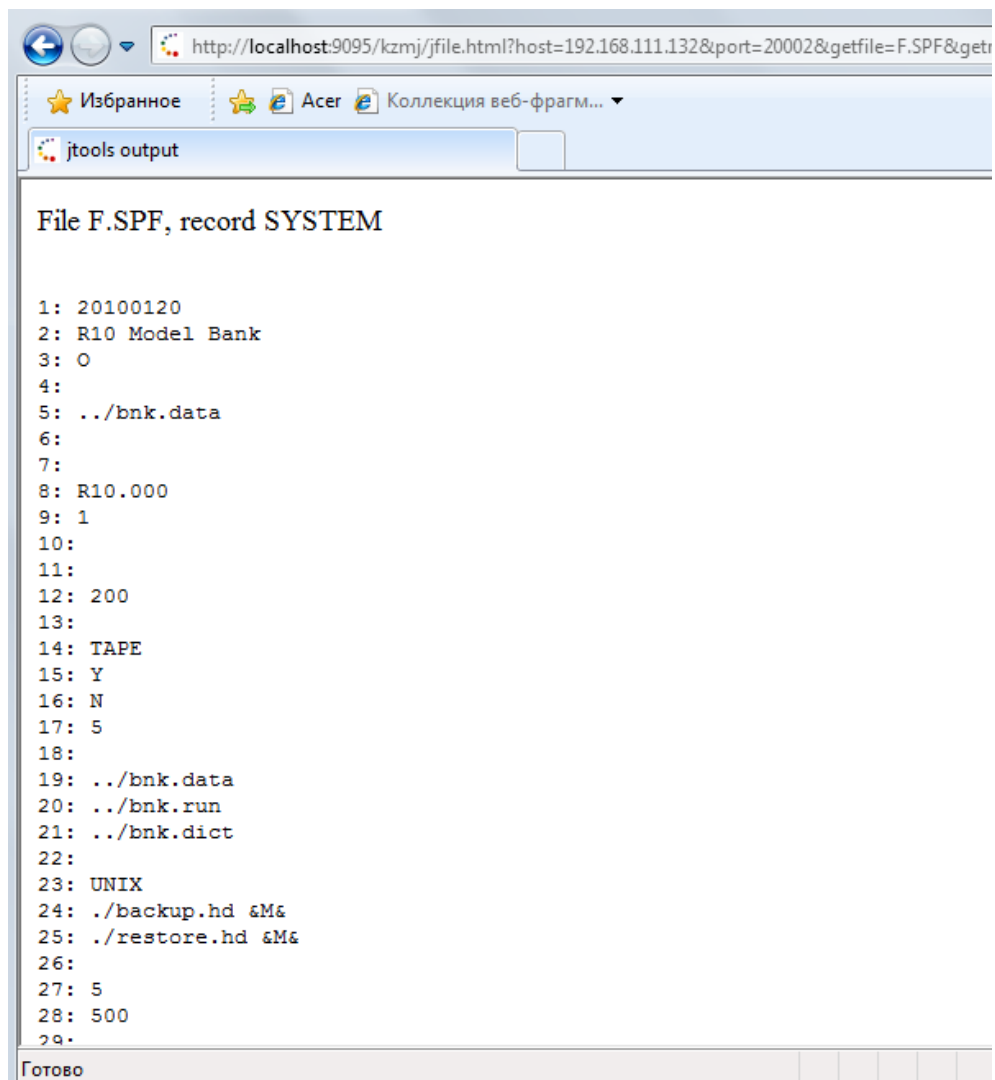
File

Record

Set field

To

Result:



Request:

http://localhost:9095/kzmj/jform.html

Избранное Acer Коллекция веб-фрагм...

jtools

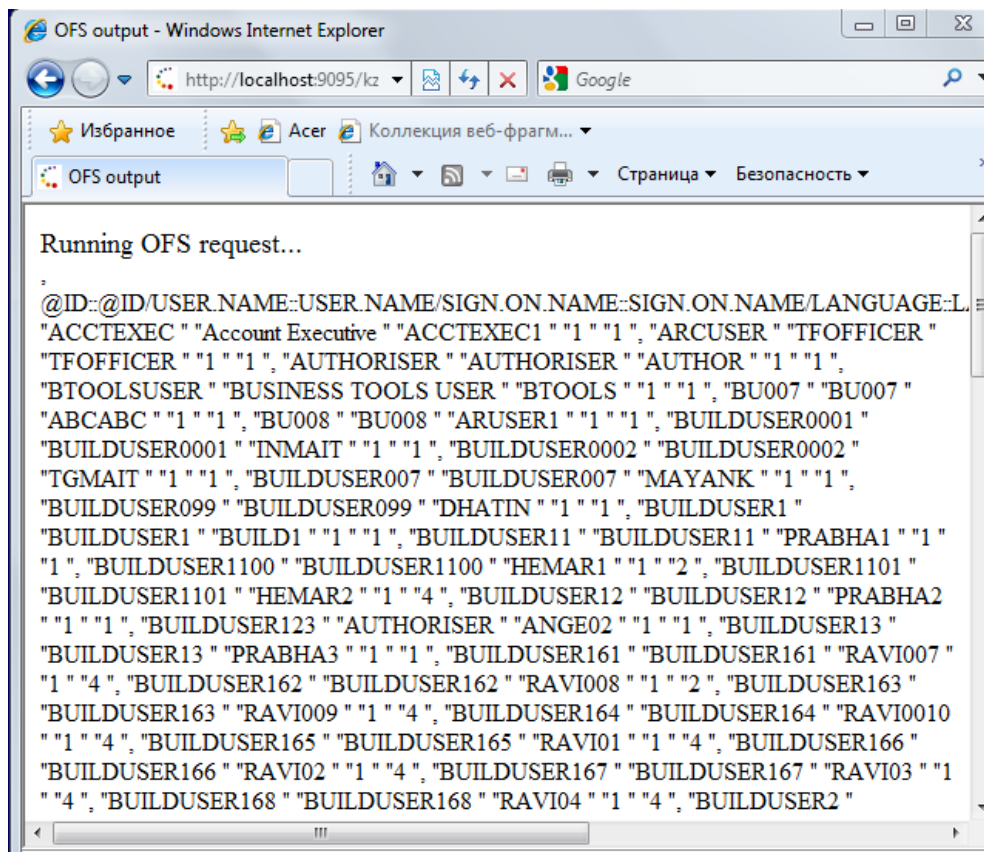
Host

Port

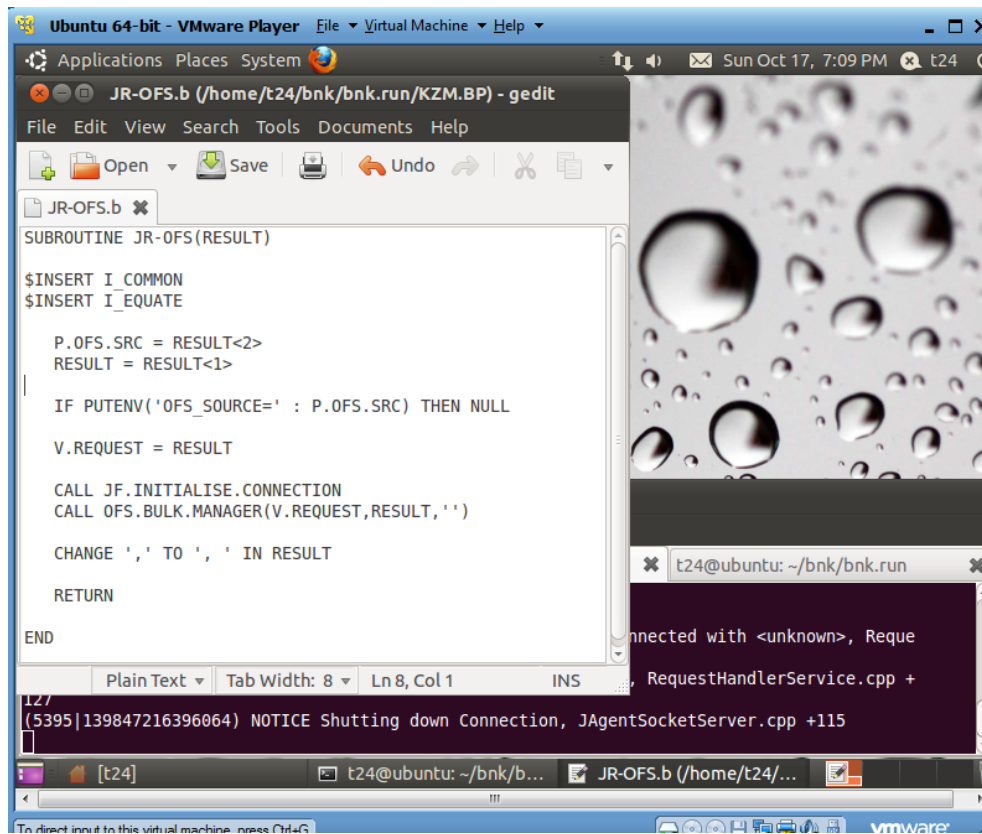
OFS.SOURCE

OFS request

Result:



To be able to run OFS request I've written a short Basic subroutine. Couldn't resist to put at least one Linux screenshot here:



On that point my playing with Java had so far ended.

## 35 Conclusions

*Having read all that you might ask – does T24 really work? It looks like it requires a lot of tampering and care? The answer to both these questions is Yes. Yes, it requires a lot of labour to get the system do what you need but once you've done it works. At least until the next upgrade.*

*Hope this book helped you to understand not only what's inside the system, but also how to approach it without fear.*

*Enjoy!*

THE END of book 1.

To be continued.

©2010 Vladimir Kazimirchik.



Written in  $\text{\LaTeX}$  using TexLIVE 2010.