APPLICATION PROGRAM INTERFACES

Introduction

This chapter describes the available options for developers who wish to enhance the operation of the current T24 system, where there are insufficient options provided by the Standard T24 utilities. Within specific applications the system allows jBase subroutines (or jBase commands in some cases) to be defined, which will then be invoked when running the applications. This provides a powerful tool to allow customisation of T24 to meet regional or customer specific requirements.

All programs written should follow the programming standards, documented in the Programming Standards section of this manual.

This chapter has been divided into the following areas of the system:

- Application Customisation
- System Management Customisation
- Reporting – Enquiry Customisation
- Delivery System
- Interface – Local Clearing
- Local Statutory Reporting
- Takeover – Installation Customisation
- Limits
- Company Customisation

Application Customisation

Introduction

This section is concerned with the options available in customising the operation and content of T24 applications.

VERSION

The *VERSION* application allows user defined subroutines to be specified in the following fields:

```
AUT.NEW.CONTENT
VALIDATION.RTN
INPUT.ROUTINE
AUTH.ROUTINE
```

AUT.NEW.CONTENT

This field would normally be used to automatically enter a value into a field specified in the associated `AUTOM.FIELD.NO`, when a record is read using the I,C,H or V functions. The automatic contents are only used if the existing content of the field matches that defined in the associated `AUT.OLD.CONTENT`.

This field may also contain a sub-routine used to perform conditional defaulting, which cannot be defined in Version, or defaulting from related files.

Format: @subroutine name

Subroutine name must be defined in *PGM.FILE* as a type S application. The field APPL.FOR.SUBR contains the application(s) allowed to use the subroutine.

Invoked: From RECORD.READ with FUNCTIONS I,C,H,V

Arguments: None

Details: Any subroutine defined in this field will be called from RECORD.READ. At this point the record has been read and is contained in R.NEW. This subroutine should modify the contents of R.NEW as required.

Note that this routine will always be invoked where defined and the contents defined in the field AUT.OLD.CONTENT are not checked. Any conditional defaulting should be contained in the subroutine coding.

Example:

The following example demonstrates a routine, which will automatically default the *CUSTOMER* SHORT.NAME into the field SHORT.NAME on the file *DE.ADDRESS*.

PGM.FILE definition:

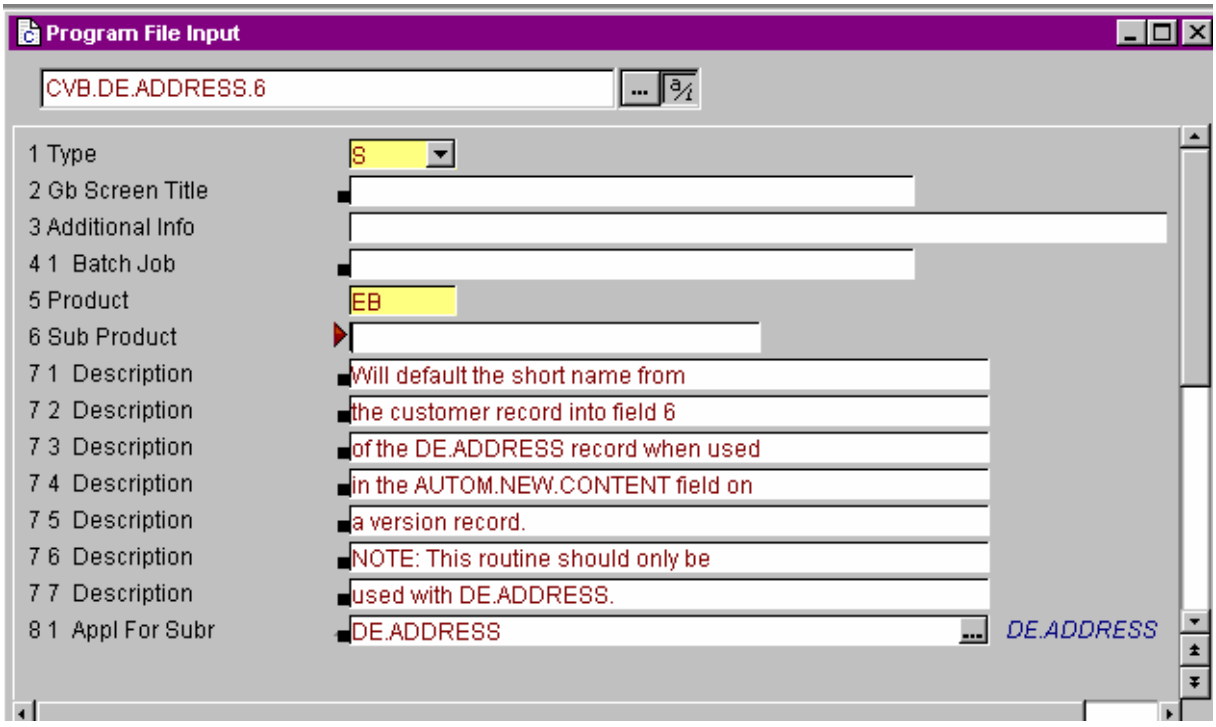


Figure 1 - PGM.FILE input

Figure 2 - Adding routine to version

Program

```
0001: * Version 3 28/02/94  GLOBUS Release No. 14.1.0 31/03/94
0002:       SUBROUTINE CVB.DE.ADDRESS.6
0003:
0004: *   This routine defaults the value in SHORT NAME column in DE.ADDRESS
0005: *   table with the customer's short name. The routine is defined in the VERSION
0006: *   record  DE.ADDRESS,
0007:
0008: $INSERT I_COMMON
0009: $INSERT I_EQUATE
0010: $INSERT I_F.CUSTOMER
0011: $INSERT I_F.DE.ADDRESS
0012:
0013:       CUSTOMER.ID = FIELD(ID.NEW, ".", 2)
0014:       CUSTOMER.ID = FIELD(CUSTOMER.ID, "-", 2)
0015:       CALL DBR("CUSTOMER":FM:EB.CUS.SHORT.NAME, CUSTOMER.ID, SHORT.NAME)
0016:       R.NEW(DE.ADD.BRANCHNAME.TITLE) = SHORT.NAME
0017:
0018:       RETURN
0019:
0020:
0021:    END
```

Figure 3 - Details of subroutine

VALIDATION.RTN

This field, together with `VALIDATION.FLD`, allows definitions of subroutines to be executed when validating the contents of fields within an application. A routine defined here would normally be used to perform specific validation or to default the contents of fields according to local requirements.

Format: subroutine name

Subroutine name must be defined with an associated `VALIDATION.FLD`. This may contain a field name, together with optional multi-value number and sub-value number.

Invoked: At Field Input validation time, immediately after the call to IN2xx as defined in the T parameters for the application, and before any validation in the section CHECK.FIELDS is executed. Also at Cross-Validation time, before the CROSS.VALIDATION section of the application is executed. The subroutine will be invoked from the program VERSION.VALIDATION.

Arguments: None

Details:

Routines defined here have the standard system variables available for checking. The following variables should be used when performing checks/defaults:

COMI

Contains the contents of specified VALIDATION.FLD. This variable should be used when defaulting/checking values for the specified VALIDATION.FLD, not R.NEW(AF), as this does not contain the value at this point.

COMI.ENRI

This should contain the enrichment for the contents of COMI.

DISPLAY

Contains the formatted version of COMI for display purposes.

ETEXT

Contains any error message generated from the subroutine. This field should be populated when an error is found.

MESSAGE

This variable can be used to determine whether the system is performing field level checks or cross validation. At cross-validation time, it will contain the value "VAL"; on-line it will be null.

Example:

The following example shows an example of defaulting the *CUSTOMER* name based on the following rules:

If local reference field CUST.STATUS (value #3 in the LOCAL.REF field) = "A"
Mandatory Input

If local reference field CUST.STATUS (value #3 in LOCAL.REF) = "B"
Default BEN.CUSTOMER = "CUSTOMER TYPE B"

If local reference field CUST.STATUS (value #3 in LOCAL.REF) = "C"
BEN.CUSTOMER not allowed

If local reference field CUST.STATUS (value #3 in LOCAL.REF) = ""
CUST.STATUS = "C"

At cross validation default, the value "ACCOUNT.NUMBER" in the field BEN.ACCT.NO of BEN.CUSTOMER is present

On the version the following is specified:

Figure 4 - Version input

```
0001:       SUBROUTINE VERSION.DEFAULT
0002: *
0003: $INSERT I_COMMON
0004: $INSERT I_EQUATE
0005: $INSERT I_F.FUNDS.TRANSFER
0006: *
0007: * This routine validates the field BEN.CUSTOMER against a local reference
0008: * item CUST.STATUS which is value 3 of the LOCAL.REF field.
0009: *
0010:       IF COMI = "" THEN                   ; * Default value C
0011:          COMI = "C"                       ; * Default value
0012:          COMI.ENRI = "CUSTOMER TYPE C"    ; * Enrichment
0013:       END
0014:       CUST.STATUS = COMI                  ; * Entered Value On VERSION
0015: *
0016:       BEGIN CASE
0017:          CASE CUST.STATUS = "A"
0018:             IF R.NEW(FT.BEN.CUSTOMER) = "" THEN     ; * Mandatory field
0019:                ETEXT = "BEN CUSTOMER MANDATORY WITH VALUE 'A'"
0020:             END
0021:          CASE CUST.STATUS = "B"
0022:             IF R.NEW(FT.BEN.CUSTOMER) = "" THEN
0023: *
0024: ** Default the value "CUSTOMER TYPE B"
0025: *
0026:                R.NEW(FT.BEN.CUSTOMER) = "CUSTOMER TYPE B"
0027:             END
0028:          CASE CUST.STATUS = "C"
0029:             IF R.NEW(FT.BEN.CUSTOMER) THEN
0030:                ETEXT = "BEN CUSTOMER NOT ALLOWED WITH CUST STATUS 'C'"
0031:             END
0032:       END CASE
0033: *
0034: ** Default value into BEN.ACCT.NO if BEN.CUSTOMER is present
0035: *
0036:       IF NOT(ETEXT) THEN                  ; * Don't default if errors
0037:          IF MESSAGE = "VAL" THEN          ; * Only at Crossval
0038:             IF R.NEW(FT.BEN.CUSTOMER) AND R.NEW(FT.BEN.ACCT.NO) = "" THEN
0039:                R.NEW(FT.BEN.ACCT.NO) = "ACCOUNT NUMBER"
0040:             END
0041:          END
0042:       END
0043: *
0044:       RETURN
0045: *
0046:    END
```

Figure 5 - Subroutine details

INPUT.ROUTINE

A subroutine may be executed at the unauthorised stage of transaction processing, as the final stage before updating unauthorised files. Multiple routines may be defined by expanding the multi-values.

Routines called at this stage may be used to update local files, or to provide additional checking or override processing. At this point of the transaction, all standard default and validation processing will have taken place.

Format: subroutine name

Subroutine name must be defined.

Invoked: From UNAUTH.RECORD.WRITE. This routine is called after CROSS.VALIDATION and BEFORE.UNAU.WRITE subroutines within the standard template.

Arguments: None

Validation should be processed in the same manner as standard cross-validation calling STORE.END.ERROR when an error is encountered. The example below shows a local

subroutine, which checks that the SHORT.NAME in the application *SECTOR* is fully alpha. If it is, the routine LOCAL.UPDATE.ROUTINE is invoked.

```
SUBROUTINE EXAMPLE.INPUT.ROUTINE
*
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.SECTOR
*
** Check that the short name is all alpha
*
      IF NOT(R.NEW(EB.SEC.SHORT.NAME)<1,1> MATCHES "1A0A") THEN
         ETEXT = "MUST BE ALPHA"
         AF = EB.SEC.SHORT.NAME ; AV = 1
         CALL STORE.END.ERROR
      END ELSE
*
** Call local routine to log change
*
         CALL LOCAL.UPDATE.ROUTINE
*
      END
```

Figure 6 - Subroutine details

Override messages should be generated using the standard STORE.OVERRIDE processing. If "NO", is replied no further processing should continue.

Updates to files must use the standard F.WRITE, F.MATWRITE and F.DELETE routines to ensure data base integrity.

NOTE: At this point file updates may have occurred, although not written to disk, since accounting will have been performed.

AUTH.ROUTINE

A subroutine may be executed at the authorised stage of transaction processing, as the final stage before updating authorised files. Multiple routines may be defined by expanding the multi-values.

Routines called at this stage may be used to update local files. No checking or override processing should be performed at this stage, as the system cannot process error conditions at this point.

Format: subroutine name

Subroutine name must be defined.

Invoked: From AUTH.RECORD.WRITE. This routine is called after BEFORE.AUTH.WRITE subroutines within the standard template.

Arguments: None

A routine used at this stage will be typically used to provide updates to local files.

OVERRIDE.CLASS.DETAILS

OVERRIDE.CLASS.DETAILS allows a subroutine to be defined to manipulate an override message so that the conditions defined in this table may be applied to it. For example, an

overdraft message could be converted into local currency for allocating an override class depending on the amount.

DATA.DEF

This field is used to define a variable element of the override message, which may be used as the basis for sub-classification. A routine may be defined to perform required extraction/conversion to the elements of the override message.

Format: @subroutine name(par_1,....par_n)

Subroutine name must be defined in PGM.FILE as a type V application. Where parameters are required, these must be defined in the PGM.FILE record field ADDITIONAL.INFO as .PAR(xx,xx...xx) where xx describes the validation rules to be applied to the parameter. Where no parameters are required the parentheses are still required.

Invoked: From STORE.OVERRIDE

Arguments: *SCAN.TEXT*, *OVERRIDE.VALUE*, *DATA.DEF*

Where:

SCAN.TEXT contains the override message as defined in *OVERRIDE.CLASS* e.g. “ACCOUNT and -UNAUTHORISED OVERDRAFT”

OVERRIDE.VALUE contains SCAN.TEXT plus the variable values

DATA.DEF contains the parameter definition in the field `DATA.DEF`

Details:

Any subroutine defined here must return details in both OVERRIDE.VALUE and DATA.DEF. The variable elements in the OVERRIDE.VALUE can be converted to a required value, for example converting foreign amounts to a local currency for checking.

The DATA.DEF value can be used to return a derived value for the specified element.

Example:

The following screenshot example shows a routine used to give a different override class depending on the amount of overdraft.

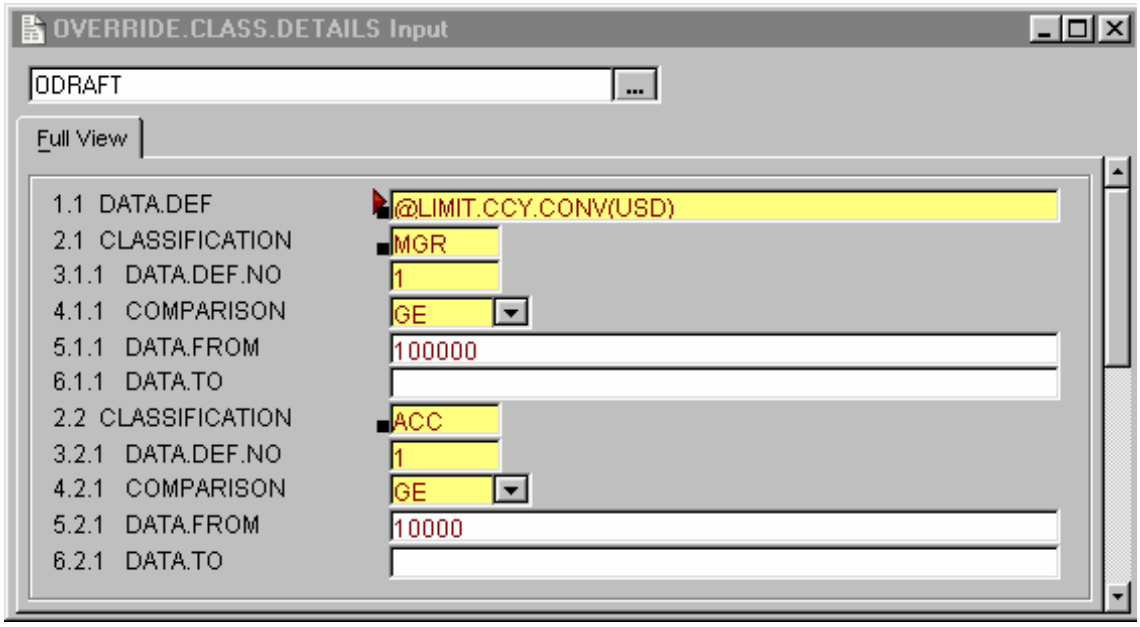


Figure 7 – OVERRIDE.CLASS.DETAILS record

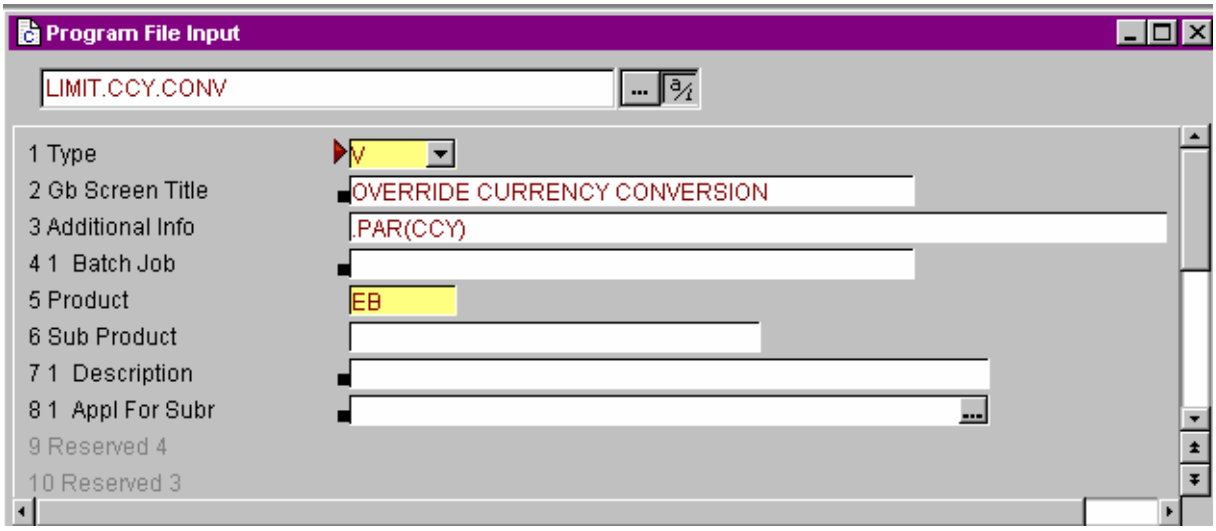


Figure 8 – PGM.FILE record for relevant subroutine

Program LIMIT.CCY.CONV

```
0002:       SUBROUTINE LIMIT.CCY.CONV(OVERRIDE,TEXT$,CURRENCY.AMOUNT)
0003: *
0004: *     LIMIT CURRENCY CONVERSION (OVERRIDES)
0005: *     ================================
0006: *
0007: *     Currency conversion for limit & overdraft override messages.
0008: *
0009: *     TEXT$           = (in) second part of override text (variables)
0010: *                       (out) converted amount
0011: *     CURRENCY.AMOUNT = currency
0012: *
0013: $INSERT I_COMMON
0014: $INSERT I_EQUATE
0015: *
0016: *====MAIN CONTROL================================================
0017: *
0018:       IF CURRENCY.AMOUNT='' THEN CURRENCY.AMOUNT=LCCY
0019:       CURRENCY=''; AMOUNT=''
0020:       LOOP
0021:          REMOVE OK FROM TEXT$ SETTING REMOVE$
0022:          BEGIN CASE
0023:             CASE OK MATCHES '3A'; CURRENCY=OK     ;* currency assumed to be ...
0024:             CASE NUM(OK)                          ;* ... prior to amount ...
0025:                IF CURRENCY#'' THEN
0026:                   IF CURRENCY#CURRENCY.AMOUNT THEN
0027:                      CALL LIMIT.CURR.CONV(CURRENCY,ABS(OK),CURRENCY.AMOUNT,AMOUNT,'')
0028:                   END ELSE
0029:                      AMOUNT=ABS(OK)
0030:                   END
0031:                   REMOVE$=0
0032:                END
0033:             CASE 1 ; CURRENCY=''                  ;* ... (immediately)
0034:          END CASE
0035:       WHILE REMOVE$ DO
0036:       REPEAT
0037:       CURRENCY.AMOUNT=AMOUNT
0038:       RETURN
0039:    END
```

Figure 9 - Subroutine details

EB.API

After the subroutine has been written, it needs to specify in EB.API application

FIELDS	DESCRIPTION
ID	Specify the name of the BASIC subroutine
Description	Subroutine description
Protection Level	Specify the security protection level
Source Type	Select 'BASIC'

Program (ID)

For on-line applications the key to this record must be the same as the program to be run.

Format: Subroutine name

Subroutine name is the name of the application, which can be invoked. The field TYPE indicates the application type to be executed. For on-line use this may H,U,L,T,W or M.

Invoked: From RUN.APPLICATION

Arguments: None

Details:

Any application of types H,U,L,T or W are used to maintain a file and must conform to the standard TEMPLATE application type. See the section Template Programming for details.

Type M routines are used to execute a specific main line program where there is no standard file maintenance required, for example a conversion program, or program to print a report. Where a history is required of a program being run, a type W program should be used. See the section Template Programming for details.

When a type M program is written, it **must** contain a SUBROUTINE statement at the start in order to return to T24 once executed. See the Programming Standards section for details of commands, which must not be used.

Example:

The parameter file IBLC.PARAMETER contains local reporting codes for Belgium and Luxembourg reporting. This is a type U routine (i.e. it does not maintain a history file).

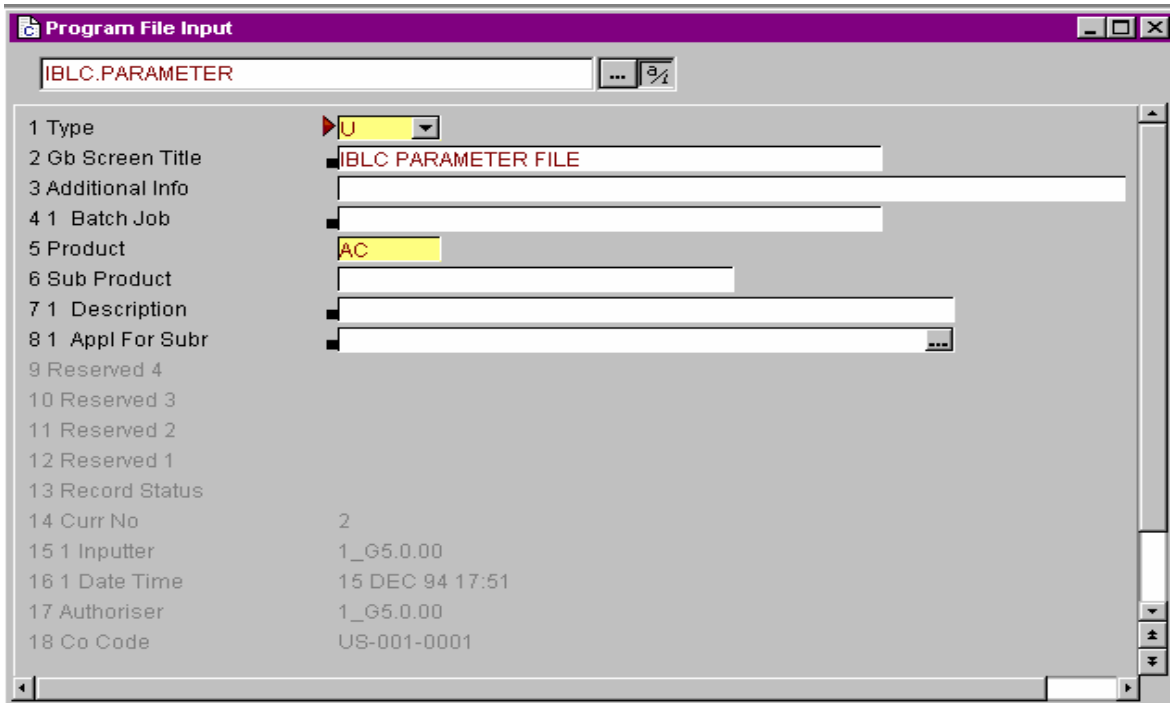


Figure 10 – PGM.FILE record for IBLC.PARAMETER

The type M program LIST.LOCK allows an operator to list the active system locks.

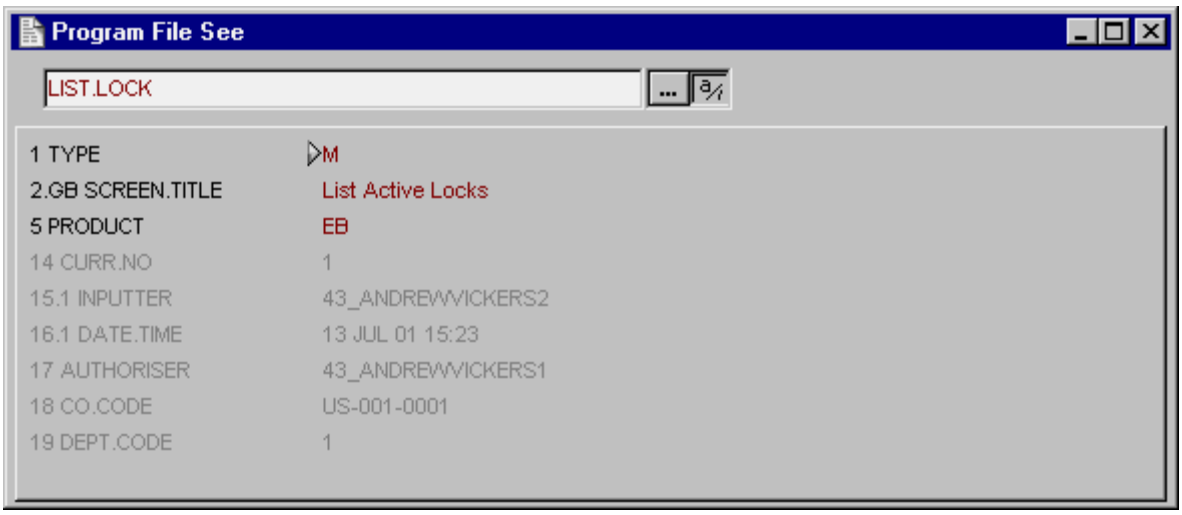


Figure 11 – PGM.FILE record for LIST.LOCK

Program

```
0001:       SUBROUTINE LIST.LOCK
0002: *
0003: * This program performs the LIST.READU command
0004: *
0005:       EXECUTE "CS" ;*  Clear Screen
0006:       EXECUTE "LIST.READU"
0007:       CRT "Press Return to Return to GLOBUS":
0008:       INPUT WAIT
0009:       RETURN
0010: *
0011:    END
```

Figure 12 – Subroutine details

BATCH.JOB

This field is used to define the names of subroutines or jBase commands which may be executed from the T24 BATCH.CONTROL process. To be executed during the end of day the `BATCH.JOB` must be defined on a *BATCH* record.

Format: @Subroutine name or jBase Command

Subroutine name is the name of the application, which can be invoked.

jBase Command the name of the item defined in the local VOC file to be executed.

This is a multi-valued field, and several subroutines and or commands may be executed sequentially.

Invoked: From B.INITIATE.PROCESS

Arguments: None.

Details:

A subroutine may be written to perform specific end of day processing. This may have a variety of functions, such as maintenance of local files/applications, production of reports,

interface file production etc. See the Programming Standards section for rules when writing end of day programs.

A jBase command or paragraph can also be executed from this option. Any item recognised in the VOC file of the correct type may be executed.

Example:

Example of UniVerse list command to produce a report on the JOURNAL file.

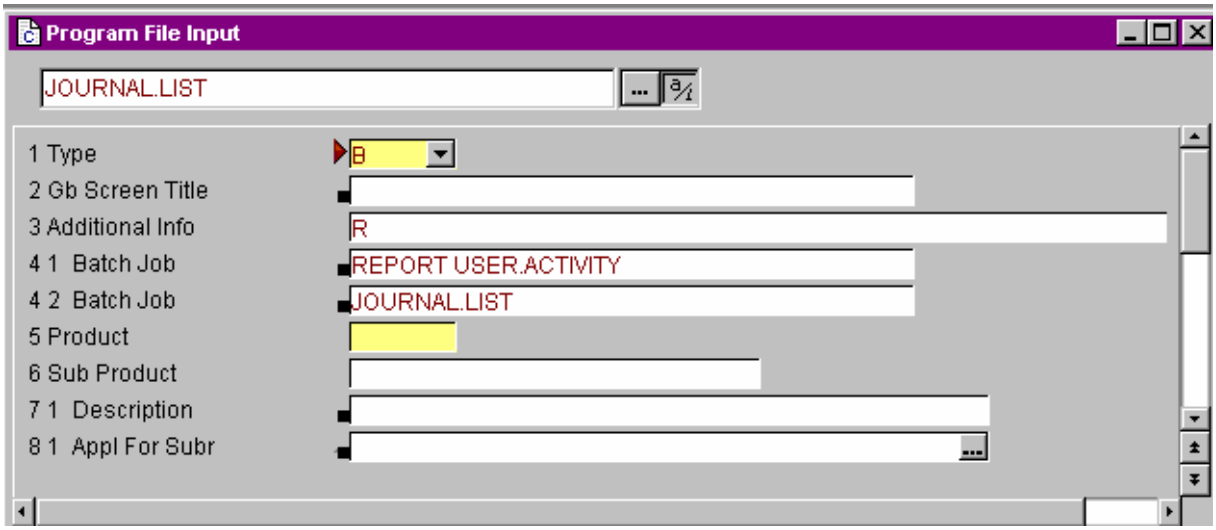


Figure 13 – PGM.FILE record for JOURNAL.LIST

```
0001: PA Paragraph to print Activity Journal for the day by department by user
0002: LIST F.JOURNAL WITH @ID NE 'HEADER' BY DEPT
0003: BY USER BY APPLICATION BY TIME BREAK.SUP "'B¯P'" DEPT
0004: HEADER "U S E R  A C T I V I T Y  J O U R N A L  - DEPARTMENT  'B'" ID.SUP BREAK.SUP
USER USER TIME APPLICATION FUNCTION TXN.REF LPTR
```

Figure 14 - Example of paragraph JOURNAL.LIST

BATCH.CONTROL

API calls like subroutine, script, crystal report or enquiry can be run at the pre batch stage. Multiple calls may be defined by expanding the multi-values.

Format: Subroutine name
It has to be defined in VOC

SPT Script Name
It has to be defined in F.SCRIPT.DESIGNER

RPT Report Name
It has to be defined in F.REPORT.CONTROL and it has to be a crystal report.

ENQ Enquiry Name
It has to be defined in F.ENQUIRY

This is a multi-valued field and several API calls may be executed sequentially.

Invoked: From BATCH.CONTROL. This routine is called just before control passes to the batch menu.

Arguments: None.

Details: Error messages are passed back in ETEXT.

System Management Customisation

Introduction

This section contains details of options available to users for customisation of system management. This includes the ability to be able to define commands (or executable programs) to perform system backups as part of the end of day process.

SPF

The *SPF* file allows definition of the command(s) used to perform the system backup and restore at end of day.

UNIX BACKUP and UNIX RESTORE

These fields allow definition of the UNIX backup and restore commands to be executed when the end of day process is run.

Format: UNIX command name

UNIX command name may contain any UNIX command(s) to be executed. May also contain a shell script.

Invoked: From SYSTEM.BACKUP and SYSTEM.RESTORE

Arguments: None.

Details:

Complex series of instructions may be specified in a UNIX shell script.

For further details see the Backup, Restore and Recovery chapter in the System Administration Guide and the Helptext on SPF.

Reporting / Enquiry Customisation

Introduction

The T24 utilities REPGEN and ENQUIRY provide several APIs for users who wish to perform operations on data outside the delivered functionality. T24 also provides options

for users to redirect output from reports or microfiches, which could include definition of their own subroutines.

ENQUIRY

The *ENQUIRY* application provides three main areas where local routines may be added to provide additional functionality. These are:

- Selection of data where criteria cannot be easily specified within the existing application (this will be covered under *STANDARD.SELECTION*).
- `CONVERSION` routines to manipulate/enhance data to required format.
- `BUILD.ROUTINE` to build initial data to base enquiry.

Conversion

Local conversion routines may be written to manipulate extracted data.

Format: @ subroutine name

Subroutine name is the name of the jBase subroutine to be executed. Note the required space between @ and the name.

Invoked: From ENQ.BUILD.PAGE for each item the conversion is associated with

Arguments: None

Details:

The enquiry system has its own common area **I_ENQUIRY.COMMON,** which must be inserted at the start of all conversion routines. This allows access to the variables controlling the enquiry processing. See the insert in GLOBUS.BP for details of all the variables passed in this common.

The following variables are the most likely ones to be required when writing a conversion routine:

ID	- Current id of the record being processed
R.RECORD	- The current record being processed
O.DATA	- The current incoming data being processed. This is also the returned data.
VC	- The current multi-value number being processed
S	- The current sub-value number
VM.COUNT	- The maximum number of multi-values within the current record
SM.COUNT	- The maximum number of sub-values within the current record

Example:

The following example shows a routine, which displays either the actual maturity date of an LD/MM deal, or displays the number of day's notice:

```
0001: * Version 3 27/09/94  GLOBUS Release No. G6.0.00 13/04/95
0002:       SUBROUTINE LD.ENQ.MATDATE
0003:
0004: * Enquiry subroutine to convert final maturity date
0005:
0006: ************************************************************
0007:
0008: $INSERT I_COMMON
0009: $INSERT I_EQUATE
0010: $INSERT I_ENQUIRY.COMMON
0011:
0012: ************************************************************
0013:
0014:       BEGIN CASE
0015:          CASE O.DATA EQ ''
0016:          CASE O.DATA EQ 0
0017:             O.DATA = 'CALL'
0018:          CASE O.DATA LT 1000
0019:             O.DATA := ' DAYS NTCE'
0020:          CASE OTHERWISE
0021:             DISPLAY = O.DATA
0022:             CALL MSK(11,'D')
0023:             O.DATA = DISPLAY
0024:       END CASE
0025:
0026: RETURN
0027:
0028: END
```

Figure 15 - Subroutine details

BUILD.ROUTINE

A routine may be called prior to the selection phase of the enquiry when running the enquiry. This routine should be used to manipulate the data prior to selection, for instance it could be used to build a work file.

Format: Subroutine name

Subroutine name is the name of the jBase subroutine to be executed. More than one routine may be specified.

Invoked: From T.ENQUIRY.SELECT and S.ENQUIRY.SELCTION

Arguments: ENQUIRY

Where ENQ is a dynamic array containing the entered selection criteria as follows:

ENQ<1>	Name of enquiry
ENQ<2,x>	Selection field names
ENQ<3,x>	Associated Operands
ENQ<4,x,y>	Data List

Details:

The data passed in ENQ should be used within the subroutine to determine the action to be taken. Data is not required to return to the enquiry system.

External Link to Enquiries

The enquiry system is not dependent on being invoked by the selection screen *ENQUIRY*. It can be invoked from applications (providing they are running under T24) using the following argument syntax:

CALL ENQUIRY.DISPLAY (QQQ)

Where QQQ is a dynamic array with the format:

QQQ<1>	Enquiry name (key to F.ENQUIRY)
QQQ<2,x>	Selection field names
QQQ<3,x>	Associated selection operands
QQQ<4,x,y>	Associated selection data
QQQ<9,z>	Multi valued list of sort requirements
QQQ<10>	Display mode can be: OUTPUT - Print in report format Null - Display to screen P - Print of screen format

Standard Selection

The *STANDARD.SELECTION* application allows definition of local subroutines, which can be used as selection items in the Enquiry system. These can be used within enquiry to perform selections of data not possible through the existing system.

They can also be used to build virtual records containing data from different files; this can be achieved using a NOFILE record type.

SYS.FIELD.NO and USR.FIELD.NO

These fields are used to hold the routine name when the associated SYS.TYPE or USR.TYPE is a type R.

Format: Subroutine name

Subroutine name is the name of the subroutine to be executed.

Invoked: From CONCAT.LIST.PROCESSOR

Arguments: RTN.LIST
Where RTN.LIST is a dynamic array containing the selected keys to be returned to the enquiry system, separated by field markers (@FM).

Details:

The main purpose of using a routine at this point is to return a list of keys for the enquiry process to use. Possible reasons for using a routine may be: the selection required might not be possible to be entered directly into the enquiry system; require additional checks; or the selection may simply be too slow and may require an alternative access method.

Since the routine will need to use the common enquiry variables, the insert **I_ENQUIRY.COMMON** should always be inserted at the top of the subroutine. The main variables likely to be required are:

D.FIELDS	- Contains a list of the Selection Field Names
D.LOGICAL.OPERANDS <X>	- Contains a list of the associated operands entered in numeric form. The following values are used: 1 EQ 2 RG

3 LT
4 GT
5 NE
6 LK
7 UL
8 LE
9 GE
10 NR

D.RANGE.AND.VALUE <X,Y> - Associated entered values

The routine must perform the required processing to build the RTN.LIST. Note that if this routine is being used to “pre-select” data ready for a further selection within the enquiry system, if no records are returned, the enquiry system will attempt to select the whole file. This may have performance implications.

NOFILE Standard Selection Records

Where an enquiry is required to show data, which cannot be extracted, from a specific file, it may be necessary to create a NOFILE *STANDARD.SELECTION* record. As its name implies the STANDARD.SELECTION record does not describe an existing file. This standard selection item can be used in *ENQUIRY* as a valid `FILE.NAME`.

Since there is no actual underlying file in the system, the selection must be performed by a routine, described in the previous section. An additional CONVERSION routine will usually be required to build R.RECORD, the record used in the enquiry.

For example an enquiry may be required which is driven from two files, A and B.

- A NOFILE STANDARD.SELECTION record will be created containing at least one field, used to define the selection routine
- The selection routine selects files A and B and returns a list of keys in the format filename*id
- A conversion routine is written, so that given filename*id it will read the correct file, and build the data into a common format in R.RECORD
- The routine is attached to field 0 in the enquiry, so that as soon as field 0 is processed, R.RECORD contains the expected layout with data extracted from the correct file.

REPGEN.CREATE

The repgen utility allows use of subroutines in two areas:

- MODIFICATION
- FL.DECISION.FR

FL.DECISION.FR

Repgen allows a subroutine to be entered to perform a selection. The value SUB must be entered in `FL.DECISION` to indicate that this field contains a sub-routine definition.

Format: Subroutine name

Subroutine name is the name of the subroutine to be executed. Only one subroutine may be defined per read file. The subroutine name must be defined on PGM.FILE file as a type S application.

Invoked: From RGS.... program generated.

Arguments: FILENAME
Where FILENAME is the full filename to be selected.

Details:

The routine should perform the required selection of the FILENAME supplied and return an **ACTIVE** select list to the RGS... program.

The *REPGEN.SORT* record is available in R.NEW, and may contain specified values in the fields CONSTANTS.

Example:

The following routine selects *CUSTOMER* records with a specified account officer or customer number. Account officer is specified in Value 1 of the field CONSTANTS, customer number is specified in Value 2.

```
0001: * Version 1 07/10/93  GLOBUS Release No. 12.2.0 18/11/93
0002:       SUBROUTINE RG.SEL.CUST.FOR.CENTRALE(FILENAME)
0003:
0004: $INSERT I_COMMON
0005: $INSERT I_EQUATE
0006: $INSERT I_F.REPGEN.SORT
0007:
0008:
0009: * First comment line = account officer
0010: * Second comment line = customer numbers
0011: * Third comment line  = report currency
0012:
0013:       GET.ACCOUNT.OFFICERS = R.NEW(RG.SRT.CONSTANTS)<1,1>
0014:       CONVERT SM TO SPACE(1) IN GET.ACCOUNT.OFFICERS
0015:       GET.CUSTOMERS = R.NEW(RG.SRT.CONSTANTS)<1,2>
0016:       CONVERT SM TO SPACE(1) IN GET.CUSTOMERS
0017:
0018:       BEGIN CASE
0019:          CASE GET.ACCOUNT.OFFICERS
0020:                          SEL.COMM  =  "SELECT  ":FILENAME:"  WITH  ACCOUNT.OFFICER  EQ
":GET.ACCOUNT.OFFICERS
0021:          CASE COUNT(GET.CUSTOMERS,' ')
0022:             SEL.COMM = "SELECT ":FILENAME:" WITH @ID EQ ":GET.CUSTOMERS
0023:          CASE GET.CUSTOMERS NE ''
0024:             SEL.COMM = "SELECT ":FILENAME:" '":GET.CUSTOMERS:"'"
0025:          CASE 1
0026:             SEL.COMM="SELECT ":FILENAME:" WITH PRINT.CENTRALE EQ 'Y'"
0027:       END CASE
0028:
0029:       CALL !HUSHIT(1)
0030:       EXECUTE SEL.COMM
0031:       CALL !HUSHIT(0)
0032:
0033:       RETURN
0034:    END
```

Figure 16 - Subroutine details

Modification

The MODIFICATION field allows a sub-routine to be called to manipulate the extracted data.

Format: @ Subroutine name#n

Subroutine name is the name of the jBase subroutine to be executed. #n denotes the argument number in the call to the subroutine.
The subroutine must be defined in *PGM.FILE* as a type R routine, together with the required number of parameters in field ADDITIONAL.INFO as .PAR(xx, ..xx).
The actual name of the sub-routine must be defined in the PGM.FILE record in the field BATCH.PROCESS prefixed by a @.

Invoked: From RGS.... program generated.

Arguments: Par_1 ,Par_n
Where Par_n may be any number of parameters (at least one) as per the definition in PGM.FILE.

Details:

All details required in the subroutine from the repgen must be passed into the subroutine as separate arguments. A single value may be returned.

Example:

The following routine returns a formatted *LIMIT.REFERENCE* with leading zeroes.

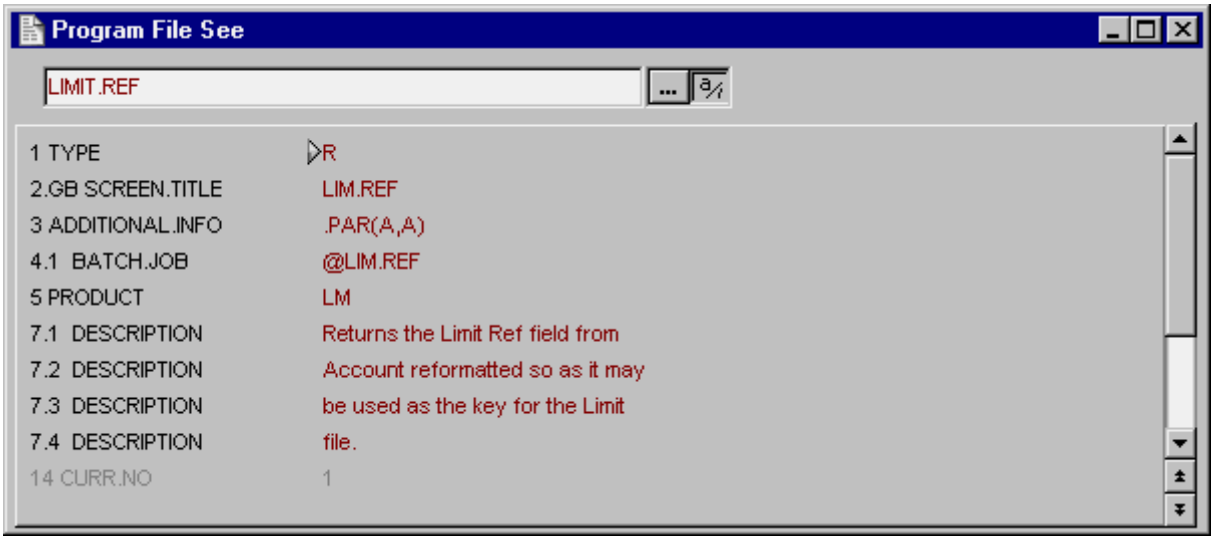


Figure 17 – LIMIT.REF routine in PGM.FILE

Program

```
0001:       SUBROUTINE LIM.REF (Y.LIMIT.REF.IN,Y.LIMIT.REF.OUT)
0002: * Subroutine to return the LIMIT.REF field from the Account
0003: * Record with the correct number of zeros added in front so
0004: * that i.e. can be used to read the LIMIT file.
0005: *
0006: * Incoming:
0007: *         Y.LIMIT.REF.IN - LIMIT.REF field from Account record in the
0008: *                          format XXX.## where XXX can be 1 to 7 char num
0009: *
0010: * Outgoing:
0011: *         Y.LIMIT.REF.OUT - 0000XXX.## Zeros added to make the Limit
0012: *                           reference as 7 chars.
0013: *
0014:       ZEROS = '0000000'
0015:       Y.TEMP.LIMIT.REF = FIELD(Y.LIMIT.REF.IN,'.',1)
0016:       LEN.LIM.REF = LEN(Y.TEMP.LIMIT.REF)
0017:       NO.OF.ZEROS = 7 - LEN.LIM.REF
0018:       Y.LIMIT.REF.OUT = ZEROS[1,NO.OF.ZEROS]:Y.LIMIT.REF.IN
0019:       IF Y.LIMIT.REF.IN = '' THEN Y.LIMIT.REF.OUT = ''
0020:    END
```

Figure 18 - Subroutine details

RE.STAT.REQUEST

An alternative print routine may be specified instead of *RE.STAT.REQUEST* to produce CRF reports.

PRINT.ROUTINE

Alternative print routine to RE.STAT.PRINT standard routine

Format: Subroutine name

Subroutine name is the name of the subroutine to be executed.

Invoked: From RE.STAT.REQUEST$RUN

Arguments: Report Params, Output Mode, Lang Code, Base Currency
Where Report Params contains the name of the report, plus *D is the detailed report has been requested, or *B if both summary and detail are requested.
Output Mode contains the output mode requested
Lang Code contains the language code requested
Base currency contains requested base currency

Details:

Production of CRF reports is a complex process – extreme care should be taken if using this option.

CREATE.FICHE.TAPE

Allows definition of a command (or routine), which can be used to create a fiche file.

TAPE.CREATE

Any jBase command or subroutine may be specified here. Usually a UNIX cat command will be used to build a file.

Format: Command

Command may be any executable command from jBase.

Invoked: From EXECUTE.COMMAND.

Arguments: None

Details:

The routine is driven from F.FICHE.HOLD.CONTROL, and will be able to pass the id in the command line. This can then be accessed within any required routine using the system variable @SENTENCE.

For example, the command specified could be:

```
LOCAL.FICHE.LOAD &FICHE.HOLD.CONTROL>@ID&
```

The routine would then have to check @SENTENCE[" ",2,1] in order to extract the id.

See the Helptext for further examples.

PRINTER.ID

Allows definition of a command (or routine) that can be used to create a file of printed output. This means that whenever T24 output is directed to this printer id, the command will be invoked.

Command

Any jBase command or subroutine may be specified here. Usually a UNIX cat command will be used to build a file.

Format: Command

Command may be any executable command from jBase.

Invoked: From EXECUTE.COMMAND.

Arguments: None

Details:

The routine is driven from F.HOLD.CONTROL, and will be able to pass the id in the command line. This can then be accessed within any required routine using the system variable @SENTENCE.

For example, the command specified could be:

```
LOCAL.PRINT.LOAD &HOLD.CONTROL>@ID&
```

The routine would then have to check @SENTENCE[" ",2,1] in order to extract the id.

See the Helptext for further examples.

Delivery System

Introduction

The T24 delivery system provides the ability for user defined routines for mapping messages, control of disposition, processing SWIFT interfaces, and for formatting inward and outward Swift messages. The Delivery system has been further opened up to enable users to define their own formatting rules for messages and to write interface routines, using the Generic Delivery Interface.

DE.FORMAT.SWIFT

The *DE.FORMAT.SWIFT* application allows a subroutine to be called for a particular Swift Field when processing incoming Swift messages

INWARD.ROUTINE

Format: Enter Y in field.

Subroutine must be called DE.ICONV.nn where nn is the SWIFT field tag, e.g. DE.ICONV.61.

Invoked: From DE.I.FORMAT.SWIFT.MESSAGE.

Arguments: FIELD, LINE
Where FIELD contains the data from the SWIFT message for decoding LINE contains the decoded message. Field markers should separate each component of the LINE using the field marker, (@FM).

Details:

A routine may be written where a SWIFT field contains several components, which need to be separated and possibly converted so that incoming processing can correctly handle the message.

Example:

The following routine decodes swift field 32, which is comprised of:

- Value Date (6 chars)
- Currency (3 alpha)
- Amount

E.g. 010195GBP1234,56

```
SUBROUTINE DE.ICONV.32(FIELD,DECODED.LINE)
 *
 * Decodes statement line from SWIFT format to fields on records
 *
 $INSERT I_COMMON
 $INSERT I_EQUATE
 *
* DECODED.LINE<1> = Value date
 *              <2> = Currency
 *              <3> = Amount
 *
   VALUE.DATE = FIELD[1,6] ;* Value date
   CCY = FIELD[7,3]
   AMOUNT = FIELD[10,20]
 *
   CONVERT ',' TO '.' IN AMOUNT ;* Amount is stored with , decimals
 *
   DECODED.LINE<1> = VALUE.DATE
   DECODED.LINE<2> = CCY
   DECODED.LINE<3> = AMOUNT
 *
   RETURN
 END
```

Figure 19 - Subroutine details

DE.WORDS

This application allows a user routine to be defined for a given language to allow translation of numbers to words.

Format: Subroutine Name

Invoked: From DE.O.FORMAT.PRINT.MESSAGE
DE.O.FORMAT.TELEXP.MESSAGE
PRODUCE.DEAL.SLIP.

Arguments: IN.AMT, OUT.AMT, LINE.LENGTH, NO.OF.LINES,ERR.MSG

Where IN.AMT contains the amount to be converted to words. Note that this may also be in the format amount*CCY where CCY is the SWIFT currency code. When this option is passed, the currency code could be translated and appended to the amount returned.

OUT.AMT contains the alpha character representation of the amount to be returned.

LINE.LENGTH may be passed with the maximum length of the amount. When this is exceeded the amount should be split into multi values.

NO.OF.LINES may be passed. This can be used to format the amount to a specific number of lines. Blank lines should be padded with a character (such as “*”) as these may be used in cheque printing.

ERR.MSG is used to return any error message related to errors encountered in processing.

Details:

Where a new language is used which does not fit into the existing DE.WORDS format, or requires special processing, a routine should be written to perform the conversion of numbers to words. This may reference a DE.WORDS record for the language if required.

DE.DISP.CONTROL

A user-defined routine may be called to provide enhanced selection for disposition control.

FIELD.NAME

Allows a subroutine to be defined to return either true or false depending on a selection match.

Format: @Subroutine name.

Must be an entry in the VOC of type 'V'.

Invoked: From DE.DISP and DE.O.DISPOSITION.CONTROL

Arguments: The routine is passed the current *DE.O.HEADER* record in argument one, the `OPERAND` in argument two and the `CONDITION` in argument three. The return argument is argument four and should evaluate to true (1) or false (0 or null).

Details:

The routine itself should perform a selection and determine a match and set the return argument accordingly.

DE.MAPPING

The *DE.MAPPING* application allows a user subroutine to modify the information passed to APPLICATION.HANDOFF by the calling application and hence to map additional data, which is not normally available for the message type.

Routine

Allows a subroutine to be defined, which may modify the contents of the delivery hand-off.

Format: @Subroutine name.

Must be an entry in the VOC of type 'V'.

Invoked: From APPLICATION.HANDOFF.

Arguments: A DIMensioned array of the nine hand-off records is passed as the first argument and a null in the second argument, which is used as a return error message.

Details:

The routine is passed all nine of the hand-off records in a DIMensioned array as the first argument and if there is a value in the second value on return from the routine the mapping does not proceed and the error message is handed back to the calling application.

If all the records are blanked by the call to the user routine the mapping process does not proceed and an error returned to the calling application.

DE.CARRIER

The delivery carrier file, *DE.CARRIER*, contains details of all the carriers available in Delivery. To enable a carrier, it must be specified on the Delivery Parameter file, *DE.PARM*.

The id of this file is the name of the carrier, as used in *DE.PRODUCT*. Each record contains the address type to be used for the carrier (i.e. when accessing DE.ADDRESS), the formatting rules (DE.FORMAT.CARRIER) and the carrier module (e.g. DE.O.CC.SWIFT). If the carrier module is GENERIC, i.e. the messages are handled by the generic program DE.CC.GENERIC, then the interface must be specified. The interface must reference a record on *DE.INTERFACE*, which contains details of the protocol for all generic interfaces (non-generic interface details are stored on the parameter file, DE.PARM).

When the record is authorised, formatting and carrier files are created if they do not already exist. These files are F.DE.O.MSG.format-module and F.DE.O.PRI.format-module for the formatting files and F.DE.O.MSG.interface and F.DE.I.MSG.interface for the interface files.

Address

Specifies the type of record to be read from the delivery address file, *DE.ADDRESS*, to get the address for this carrier the following can be keys can be used:

E.g. ADDRESS could be specified as SWIFT. Therefore, the delivery address file will be accessed with a key of:

company-code "." "C-" customer-no "." "SWIFT" "." address-no

E.g. USD0010001.C-10001.SWIFT.1

Carrier Module

Specifies the formatting module to be used. The rules describing the formatting of the messages should therefore exist on the file, DE.FORMAT.format-module, e.g. *DE.FORMAT.SWIFT*. Various formatting modules are included in T24 (e.g. SWIFT, PRINT, different telex formats). However, new formatting modules can be written. The formatting rules would be specified on a new table, DE.FORMAT.carrier-module. Therefore, a template-type program DE.FORMAT.carrier-module must be written to define the formatting rules. The messages will then be formatted by the formatting program, DE.O.FORMAT.format-module.MESSAGE, which would also need to be written.

Interface

Specifies the name of the interface to be used. The `CARRIER.MODULE` must be specified as "GENERIC". Messages will be processed by the generic delivery interface, DE.CC.GENERIC, but will be sent/received by the interface routines specified on DE.INTERFACE. The name of the interface specified in this field must reference a record on *DE.INTERFACE*.

DE.INTERFACE

This file contains details of the protocols for all interfaces which use the Generic Delivery Interface. The protocols for interfaces written prior to the introduction of the Generic Delivery Interface are either stored on *DE.PARM* or are hard-coded in the program. Sequence numbers for existing interfaces are stored on F.LOCKING.

The id of the file is the interface as defined in the interface field on *DE.CARRIER*.

There is little validation of the fields on DE.INTERFACE. This is to allow for maximum flexibility when new interfaces are written. Each field can be used to control how the interface is defined and used, more information on this can be found in the Helptext.

OUT.IF.ROUTINE

Defines the name of the interface routine, which is called from the generic delivery interface program, to send the messages to the required carrier.

If this field is left blank, the messages are still written to the interface file, but it is assumed that a separate program is invoked at a later time to send the messages (for example, to create batched messages once a day).

Format:	Subroutine name
	A VOC entry must exist
Invoked:	From DE.CC.GENERIC.
Arguments:	MISN - the message sequence number MSG - the formatted message to be sent

Details:

The routine is called from DE.CC.GENERIC, the generic delivery program. DE.CC.GENERIC controls all the updates of the delivery files - the outward interface program merely has to send the message to the interface required. The routine is “executed”. Therefore, the routine does not have to be an Info/Basic routine. However, a VOC entry must exist for it.

IN.IF.ROUTINE

Defines the name of the interface routine, which is called from the generic delivery interface program, to receive messages from the required carrier.

Format:	Subroutine name.
	A VOC entry must exist
Invoked:	From DE.CC.GENERIC.
Arguments:	GLOBUS.REF - the 5-digit sequence number CODE - a code determining the type of message (ACK (positive acknowledgement), NAK (negative acknowledgement) or blank for an incoming message) MSG - the formatted message to be sent R.HEAD - a dynamic array of the delivery header record

Details:

The routine is called from DE.CC.GENERIC, the generic delivery program. DE.CC.GENERIC controls all the updates of the delivery files - the inward interface program merely receives messages and acknowledgements (positive or negative) from the interface required. The routine is "executed". Therefore, the routine does not have to be an Info/Basic routine. However, a VOC entry must exist for it.

The delivery header record is passed back from the routine. Although this record will be created by DE.CC.GENERIC, the interface routine can populate any fields recognised in the message.

DE.MESSAGE

A routine can be defined to process inward messages to generate Funds Transfers using the OFS module.

IN.OFS.RTN

A default routine, FT.OFS.DEFAULT.MAPPING is available for message types 100, 200, 202, and 205.

Format: Subroutine name.

Subroutine must be defined in *PGM.FILE* as a type S program.

Invoked: From FT.OFS.INWARD.MAPPING.

Defined in `IN.DIR.RTN` on OFS.SOURCE, this routine is called from the OFS phantom process *OFS.REQUEST.MANGER*.

Arguments:
DEI.MSG.FT.IN key
R.INWARD - mapped from delivery system
R.SWIFT - swift message text
MESSAGE.TYPE - e.g. 100, 200, 202
R.DE.MESSAGE - DE.MESSAGE record
OFS.KEY - returned OFS message key
OFS.MESSAGE - returned OFS message

The source code for FT.OFS.DEFAULT.MAPPING is released and a detailed description of the default mapping logic follows.

FT.OFS.DEFAULT.MAPPING

R.INWARD

This record is mapped from incoming SWIFT messages by the delivery system, and is used to create OFS messages, which will in turn create Funds transfers.

Field No	Field Name	Map from	Swift field
	CUSTOMER		50
	MSG.TYPE		
	TXN.REF.NO		20
	RELATED.REF.NO		21
	VALUE.DATE		32
	CURRENCY		
	AMOUNT		
	ORDER.CUST		52
	ORD.BK.CD		52
	ORD.BK.ACC		52
	ORD.BK.CUS		52
	S.COR.BK.CD		53
	S.COR.BK.ACC		53
	S.COR.BK.CUS		53
	R.COR.BK.CD		54
	R.COR.BK.ACC		54
	R.COR.BK.CUS		54
	INTMED.BK.CD		56
	INTMED.BK.ACC		56
	INTMED.BK.CUS		56
	ACC.WITH.BK.CD		57
	ACC.WITH.BK.ACC		57
	ACC.WITH.BK.CUS		57
	BEN.BK.CD		58
	BEN.BANK.ACC		58
	BEN.BANK.CUS		If message type = 200 set to CUSTOMER
	BEN.ACCT.NO		59
	BEN.CUSTOMER		59
	PAYMENT.DETAILS		
	BEN.OUR.CHARGES		
	BK.TO.BK.INFO		
	CUS.COMPANY		
	COMPANY		
	DEPT.CODE		
	APP.FORMAT		
	LANGUAGE		

Figure 20 - FT.OFS.DEFAULT.MAPPING details

File: FUNDS.TRANSFER

Following fields are mapped from R.INWARD, and then used to generate an OFS message.

FT Field Name	Map to	FT Inward field
TRANSACTION.TYPE		
DEBIT.ACCT.NO		

IN.DEBIT.ACCT.NO		If S.COR.BK.CUS then
		S.COR.BK.CUS else
		If R.COR.BK.CUS then
		R.COR.BK.CUS

CURRENCY.MKT.DR		
DEBIT.CURRENCY		CURRENCY
DEBIT.AMOUNT		AMOUNT
DEBIT.VALUE.DATE		VALUE.DATE
IN.DEBIT.VALUE.DATE		VALUE.DATE
DEBIT.THEIR.REF		TXN.REF.NO
CREDIT.THEIR.REF		RELATED.REF.NO
CREDIT.ACCT.NO		
CURRENCY.MKT.CR		
CREDIT.CURRENCY		
CREDIT.AMOUNT		
CREDIT.VALUE.DATE		
TREASURY.RATE		
NEG.DEALER.REF.NO		
PROCESSING.DATE		
ORDERING.CUST		FT.IN.ORDERING.CUST
IN.ORDERING.CUST		ORDER.CUST
ORDERING.BANK		FT.IN.ORDERING.BANK

IN.ORDERING.BANK	MV	If ORD.BK.ACC then
		ORD.BK.ACC
		Else
		ORD.BK.CUS
		If ORD.BK.CUS then
		<1,2> = ORD.BK.CUS
		If field still null then set to
		CUSTOMER

ACCT.WITH.BANK		
IN.ACCT.WITH.BANK		
BEN.ACCT.NO		
IN.BEN.ACCT.NO		BEN.ACCT.NO
BEN.CUSTOMER		
IN.BEN.CUSTOMER		BEN.CUSTOMER
BEN.BANK		

Figure 21 - Inward mapping field details in Funds Transfer

FT Field Name	Map to	FT Inward field
IN.BEN.BANK		If BEN.BANK.ACC then BEN.BANK.ACC If BEN.BANK.CUS then BEN.BANK.CUS
CHEQUE.NUMBER		
PAYMENT.DETAILS		PAYMENT.DETAILS
IN.PAYMENT.DETAILS		PAYMENT.DETAILS
BC.BANK.SORT.CODE		
RECEIVER.BANK		
REC.CORR.BANK		
INTERMED.BANK		
XX.IN.INTERMED.BANK		If INTMED.BK.ACC then add mv INTMED.BK.ACC If INTMED.BK.CUS then Add mv INTMED.BK.CUS
MAILING		
PAY.METHOD		
BEN.OUR.CHARGES		
IN.BEN.OUR.CHARGES		BEN.OUR.CHARGES
CHARGES.ACCT.NO		
CHARGE.COM.DISPLAY		
COMMISSION.CODE		If BEN.OUR.CHARGES then If BEN.OUR.CHARGES = "BEN" then set to "C" If = "OUR" set to "D" If BEN.OUR.CHARGES = "" Set to "C"
COMMISSION.TYPE		
COMMISSION.AMOUNT		
CHARGE.CODE		If BEN.OUR.CHARGES then If BEN.OUR.CHARGES = "BEN" then set to "C" If = "OUR" set to "D" If BEN.OUR.CHARGES = "" Set to "C"
CHARGE.TYPE		
CHARGE.AMT		
CUSTOMER.SPREAD		
BASE.CURRENCY		

Figure 22 - Inward mapping field details in Funds Transfer (cont.)

FT Field Name	Map to	FT Inward field
PROFIT.CENTRE.CUST		
PROFIT.CENTRE.DEPT		
RETURN.TO.DEPT		
PRIORITY.TXN		
BK.TO.BK.INFO		BK.TO.BK.INFO
IN.BK.TO.BK.INFO		BK.TO.BK.INFO
EXPOSURE.DATE		
FED.FUNDS		
POSITION.TYPE		
NO.OF.BATCH.ITEMS		
FREE.TEXT.MSGTO		
MESSAGE		
LOCAL.REF		
TAX.TYPE		
TAX.AMT		
AMOUNT.DEBITED		
AMOUNT.CREDITED		
TOTAL.CHARGE.AMOUNT		
TOTAL.TAX.AMOUNT		
CUSTOMER.RATE		
IN.REC.CORR.BK		
INWARD.PAY.TYPE		
IN.SEND.CORR.BK		If S.COR.BK.CUS then S.COR.BK.CUS else If R.COR.BK.CUS then R.COR.BK.CUS
TELEX.FROM.CUST		If CUSTOMER is a GLOBUS customer then set to short name Else set to CUSTOMER
DELIVERY.INREF		@ID OF INWARD
DELIVERY.OUTREF		
CREDIT.COMP.CODE		
DEBIT.COMP.CODE		
STATUS		
DELIVERY.MKR		
BATCH.NO		
ACCT.WITH.BK.ACNO		

Figure 23 - Inward mapping field details in Funds Transfer (cont.)

FT Field Name	Map to	FT Inward field
LOC.AMT.DEBITED		
LOC.AMT.CREDITED		
LOC.TOT.TAX.AMT		
LOCAL.CHARGE.AMT		
LOC.POS.CHGS.AMT		
MKTG.EXCH.PROFIT		
RATE.INPUT.MKR		
CUST.GROUP.LEVEL		
DEBIT.CUSTOMER		
CREDIT.CUSTOMER		
SEND.PAYMENT		
DR.ADVICE.REQD		
CR.ADVICE.REQD		
DEAL.MARKET		
CHARGED.CUSTOMER		
IN.REASON.OVE		
DEALER.DESK		
RECALC.FWD.RATE		
RETURN.CHEQUE		
DRAWN.ACCOUNT		

Figure 24 - Inward mapping field details in Funds Transfer (cont.)

SPECIAL.FIELDS

`W.ACCOUNT.FOUND` set to account number, (may not be a T24 account number).
`W.ACCT.FOUND` set to TRUE if an account is found.

`W.INTER.WITH` set to TRUE if an intermediary is found.
`W.ACCOUNT.WITH` set to TRUE if an account with a beneficiary is found.

DEBIT.ACCOUNT LOGIC in order of processing

Call TEST.RECEIVER.CORR if no account found, then call TEST.SENDER.CORR If no account found call TEST.SENDER.BANK.

If account found, then read *ACCOUNT* file to check if it is one of ours.

RECEIVER.CORR Swift field 54 MT100, 202, 205

Uses `FT.INWARD R.CORR` fields.

W.CD	Set to	R.COR.BK.CD
W.ACC	Set to	R.COR.BK.ACC
W.CUS	Set to	R.COR.BK.CUS

Figure 25 - SWIFT field 54 mapping

If W.CUS = "" and either W.CD or W.ACC is set then set W.CUS to CUSTOMER

Then calls the routine FIND.CUSTOMERS.ACCOUNT to attempt to determine the FT.DEBIT.ACCOUNT.

If an account has been found then:

W.ACCOUNT.FOUND is set to the account number.

W.ACCT.FOUND is set to True to indicate and account has been found.

FT.DEBIT.ACCOUNT is set to W.ACCOUNT.FOUND.

SENDER.CORR Swift field 53 MT100, 200, 202, 205

Only called if W.ACCT.FOUND is false.

Uses FT.INWARD S.CORR fields.

W.CD	Set to	S.COR.BK.CD
W.ACC	Set to	S.COR.BK.ACC
W.CUS	Set to	S.COR.BK.CUS

Figure 26 - SWIFT field 53 mapping

If W.CUS = "" and either W.CD or W.ACC is set then set W.CUS to CUSTOMER

Then calls the routine FIND.CUSTOMERS.ACCOUNT to attempt to determine the FT.DEBIT.ACCOUNT.

If an account has been found then:

W.ACCOUNT.FOUND is set to the account number.

W.ACCT.FOUND is set to true to indicate an account has been found.

FT.DEBIT.ACCOUNT is set to W.ACCOUNT.FOUND.

TEST.SENDER.BANK Mandatory swift field

Only called if W.ACCT.FOUND is false.

W.CUS is set to CUSTOMER.

N.B. W.CD and W.ACC are not initialised, so should be set to S.COR.BK.CD and S.COR.BK.ACC.

If CURRENCY is not equal to the local currency then the routine GET.CUSTOMERS.NOSTRO is called.

If CURRENCY is equal to the local currency the routine GET.CUSTOMERS.VOSTRO is called.

FT.DEBIT.ACCOUNT will be set to W.ACCOUNT.FOUND, which may be null.

FT.CREDIT.ACCOUNT logic

W.ACCT.FOUND is set to FALSE and W.ACCOUNT.FOUND is set to null.

W.INTER.WITH is set to FALSE.

The routine TEST.INTERMEDIARY is called.

If an error message is returned then no further tests take place.

W.ACCOUNT.WITH is set to FALSE.

The routine TEST.ACCOUNT.WITH.BK is called.

If an error message is returned then no further tests take place.

If the message type is a 200 then FT.BEN.BANK is set to CUSTOMER.

IF the message type is a 202 or 205 then the TEST.BENE.BANK routine is called.

IF an error message is returned then no further tests take place.

If the message type is 100 then the TEST.BENEFICIARY routine is called.

If an error message is returned then no further tests take place.

If FT.CREDIT.ACCOUNT has been set then the following checks take place.

Read the ACCOUNT.FILE with FT.CREDIT.ACCT.NO.

If no record is found then set FT.BEN.ACCT.NO to FT.CREDIT.ACCT.NO and set FT.CREDIT.ACCT.NO to null and return with an error message.

If a record is found then set FT.BEN.ACCT.NO and FT.IN.BEN.ACCT.NO to null.

Set FT.CREDIT.CURRENCY to the currency of the account.

If FT.DEBIT.CURRENCY is not equal to FT.CREDIT.CURRENCY then return with an error message.

If all the above tests have been passed then the following processing takes place.

The FT.TRANSACTION.TYPE will be set to "IT" unless the following conditions occur when it will be set to "OT".

Message type is 100 and there is an account with bank.

Message type is 200, as there will always be an account with bank and the sender bank is always the beneficiary bank.

Message type is 202, if there is no account with or intermediary then DW otherwise OT.

If the message type is 100.

If W.ACCOUNT.WITH is FALSE then FT.TRANSACTION.TYPE is set to "IT" otherwise it is set to "OT"

If the message type is 200.

FT.TRANSACTION.TYPE is set to "OT"

If the message type is 202 or 205.

If W.ACCOUNT.WITH is TRUE and FT.BEN.BANK is null then FT.TRANSACTION.TYPE is set to "DW".

Otherwise:

FT.TRANSACTION.TYPE is set to "OT".

If W.ACCOUNT.WITH is FALSE then FT.ACCT.WITH.BANK is set to FT.IN.ACCT.WITH.BANK.

If W.INTER.WITH is FALSE then FT.INTERMED.BANK is set to FT.IN.INTERMED.BANK.

TEST.INTERMEDIARY Swift field 56 MT200,202,205

If INTMED.BK.CD is set then return with an error.

If INTMED.BK.ACC is set then set FT.CREDIT.ACCOUNT to this value and set W.INTER.WITH and W.ACCT.FOUND to TRUE and return.

N.B. no validation of the account number takes place at this stage.

If INTMED.BK.ACC is not set then check the INTMED.BK.CUS as follows.

It must not be > 10 characters long and must be numeric otherwise return with an error.

Set W.CUS to INTMED.BK.CUS.

Call the routine GET.CUSTOMERS.VOSTRO

Set FT.CREDIT.ACCOUNT to W.ACCOUNT.FOUND.

If an error has been returned by GET.CUSTOMERS.VOSTRO then return.

IF FT.CREDIT.ACCOUNT is not null then set W.INTER.WITH and W.ACCT.FOUND to TRUE otherwise return with an error.

TEST.ACCOUNT.WITH.BK Swift field 57 MT100, 200,202,205

If an intermediary has been found (W.INTER.WITH = TRUE) then return.

If ACC.WITH.BK.CD is set then return with an error.

If ACC.WITH.BK.ACC is set then set FT.CREDIT.ACCOUNT to this value and set W.INTER.WITH and W.ACCT.FOUND to TRUE and return.

N.B. no validation of the account number takes place at this stage.

If ACC.WITH.BK.ACC is not set then check the ACC.WITH.BK.CUS as follows.

It must not be > 10 characters long and must be numeric otherwise return with an error.

Set W.CUS to ACC.WITH.BK.CUS.

Call the routine GET.CUSTOMERS.VOSTRO

Set FT.CREDIT.ACCOUNT to W.ACCOUNT.FOUND.

If an error has been returned by GET.CUSTOMERS.VOSTRO then return.

IF FT.CREDIT.ACCOUNT is not null then set W.INTER.WITH and W.ACCT.FOUND to TRUE otherwise return with an error.

TEST.BENE.BANK Swift field 58 MT202, 205

FT.BEN.BANK is set to BEN.BANK.CUS.

IF W.ACCT.FOUND has been set to TRUE then return.

N.B this condition is not present in FT.IN.PROCESSING.

If BEN.BANK.CD is set then return with an error.

BEN.BANK.CUS is checked as follows.

It must not be > 10 characters long and must be numeric otherwise return with an error.

Set W.CUS to BEN.BANK.CUS.

Call the routine GET.CUSTOMERS.VOSTRO

Set FT.CREDIT.ACCOUNT to W.ACCOUNT.FOUND.

If an error has been returned by GET.CUSTOMERS.VOSTRO then return.

IF FT.CREDIT.ACCOUNT is not null then set W.INTER.WITH and W.ACCT.FOUND to TRUE otherwise return with an error.

TEST.BENEFICIARY Swift field 59 MT100

If BEN.ACCT.NO is set and W.ACCOUNT.WITH is false then read the ACCOUNT file.

If a record is found then set W.ACCT.WITH to TRUE and set FT.CREDIT.ACCT.NO to BEN.ACCT.NO otherwise set FT.BEN.ACCT.NO to BEN.ACCT.NO.

If BEN.CUSTOMER is not null then set FT.BEN.CUSTOMER to this value and return.

N.B. the current version of FT.IN.PROCESSING has a further section of code, which uses BEN.CUSTOMER to check for a VOSTRO account. This code will not be executed.

FIND.CUSTOMERS.ACCOUNT

The following logic applies to this routine.

If W.CD is set then the following logic applies.

If the customer is null and the account is set then read the account file to determine if the account is one of ours, if it is then set FT.DEBIT.ACCOUNT to the account number.

N.B. the above processing does not take place in FT.IN.PROCESSING

Otherwise if the R.COR.BK.CD is set to "C" then call the routine GET.CUSTOMERS.NOSTRO.

Otherwise call the routine GET.CUSTOMERS.VOSTRO

If W.CD is null and W.CUS is set then the following logic applies

If CURRENCY is not equal to the local currency then the routine GET.CUSTOMERS.NOSTRO is called.

If CURRENCY is equal to the local currency the routine GET.CUSTOMERS.VOSTRO is called.

If an account has been found then set the flag W.ACCT.FOUND to TRUE.

GET.CUSTOMERS.NOSTRO

Read the *AGENCY* file for W.CUS.

If a record is NOT found then return.

Locate W.ACC in the NOSTRO.ACCT.NO field of the AGENCY record. If it is found then set W.ACCOUNT.FOUND to the relevant value, otherwise set it to null.

GET.CUSTOMERS.VOSTRO

This program calls the routine GET.AGENT with the following parameters set.

Parameters in	Set to	Parameters Out	Description
IN.CUST	W.CUS		Customer
IN.CCY	CURRENCY		Inward currency
IN.APP	FT		Funds Transfer
		RB.SWIFT	Agents Swift address
		RB.COUNTRY	Receiver bank country
		RB.NOSTRO	Input customers nostro accounts
		RB.VOSTRO	Input customers Vostro accounts
		RB.TEST.SIG	Test Signature
		CB.COUNTRY	Corespondent bank country
		CB.REGION	Corespondent Region
	OUT.CUSTOMER	CB.CUST	Corespondent Customer
	OUT.ACCOUNT.NUMBER	CB.ACCT	Corespondent Account number
		IB.COUNTRY	Intermediate bank country
		IB.REGION	Intermediate region
		IB.CUST	Intermediate country
		IB.ACCT.	Intermediate account number
		RETURN.CODE	Negative = error

Figure 27 - GET.CUSTOMERS.VOSTRO parameters

The following conditions must be met otherwise an error is returned.

The *AGENCY* record must be present for IN.CUST.

AUTOROUTING must not be set to "NO" on the AGENCY record.

RESIDENCE must be set on the *CUSTOMER* record for IN.CUS.

IN.CCY must be present in the AUTORTE field of the AGENCY record

"FT" or "ALL" must be present in the AUTORTE.APPL field of the AGENCY record.

If the AUTORTE.BANK for the relevant AUTORTE.APPL is set to "VOSTRO" or "CUSTOMER" then the following processing applies, otherwise the AGENCY record for the AUTORTE.BANK

for the relevant AUTORTE.APPL is read, and the above validation checks apply to the new record, if successful the following logic applies.

CB.COUNTRY is set to the residence of the CUSTOMER.

The following fields are set to the values for either "FT" or "ALL" AUTORTE.APPL field, with "FT" taking precedence.

CB.REGION is set to AUTORTE.REGN.

CB.CUST is set to AUTORTE.BANK.

CB.ACCT is set to AUTORTE.ACCT.

If CB.CUST is set to "CUSTOMER" then it is set to "VOSTRO".

On return from GET.AGENT if OUT.CUSTOMER is set to "VOSTRO" then set W.ACCOUNT.FOUND to OUT.ACCOUNT.NUMBER.

If OUT.CUSTOMER is not set to "VOSTRO" then read the CUSTOMER.CCY.ACCT file with a key of W.CUS:CURRENCY:1.

If a record is found then set W.ACCOUNT.FOUND to the first account number in the list.

FD.ACTIVITY

The Fiduciary application allows subroutines to be called to modify the contents of data passed to delivery from the application.

HANDOFF.ROUTINE

Allows a subroutine to be defined, which may modify the contents of the delivery hand-off in record number 7.

Format: Subroutine name.

Subroutine must be defined in *PGM.FILE* as a type S program.

Invoked: From FD.GENERATE.DELIVERY.

Arguments: *SPECIAL.REC*
Where SPECIAL.REC contains the additional data to be passed to delivery from the Fiduciary application.

Details:

The contents of SPECIAL.REC must be created within this routine. The Fiduciary common are I_FID.COMMON is available at this point. The following variables are likely to be required:

FD$R.ORDER( )	The current FD.FID.ORDER record
FD$R.PLACEMENT( )	The current FD.FIDUCIARY record
FD$R.BALANCES( )	The current balances record

MG.ACTIVITY

The mortgage application allows subroutines to be called to modify the contents of data passed to delivery from the application.

HANDOFF.ROUTINE

Allows a subroutine to be defined, which may modify the contents of the delivery hand-off.

Format: Subroutine name.

Subroutine must be defined in *PGM.FILE* as a type S program.

Invoked: From MG.DE.HANDOFF.

Arguments: REC1, REC2, REC3, REC4, REC5, REC6, REC7, REC8, REC9
Where REC_n contains the data to be passed to delivery from the mortgage application.

Details:

The contents of REC_n may be added to or modified according to local requirements.

Interfaces – Local Clearing

Introduction

T24 provides options for allowing the required additional functionality to be added to the Funds Transfer module in order to allow local clearing transactions to be entered according to the local rules. This functionality is provided by the parameter file *FT.BC.PARAMETER* for the local clearing transaction types, BC, BI and BD. The parameter allows subroutines to be added to perform specific local validation, and update of cross-reference files and production of additional/new delivery messages.

A further option allows a sub-routine to be invoked from the delivery processing, which can allow diversion of messages with different carriers into the local clearing system according to the coded rules.

FT.BC.PARAMETER

This application allows customisation of existing field level validation for the BC Funds Transaction type. Additionally subroutines may be defined to perform specific local validation within the FT module in the following fields:

```
FT.VALIDATION.RTN
FT.DELIVERY.RTN
STO.VALIDATION.RTN
BULK.STO.VALID.RTN
```

Additionally the ability to define subroutines called from the *CUSTOMER* and *ACCOUNT* applications is provided in the fields:

ACCOUNT.UPD.RTN
CUSTOMER.UPD.RTN

A subroutine to allow diversion of messages into the local clearing system within the delivery system may be defined in:

- **DIVERSION.RTN**

FT.VALIDATION.RTN (FUNDS TRANSFER)

This field allows definition of a subroutine, which will be used to perform cross-validation specific to the local clearing system. This routine applies to the BC transaction type within Funds Transfer, and all related transaction types, i.e. BCxx where xx is any alpha character.

Format: Subroutine name
Subroutine name contains the name of the Info Basic subroutine to be executed. The Subroutine name defined must exist on PGM.FILE as a type S program.

Invoked: From FT.CROSSVAL, after standard cross-validation (performed by FT.BC.CROSSVAL).

Arguments: Curr No
Where Curr No contains the current number of overrides held on the Funds Transfer record.

Details:

The purpose of a subroutine written at this point is to perform cross-validation of the input in the *FUNDS.TRANSFER* record according to the local requirements.

A local clearing common area is available in the insert I_F.FT.LOCAL.COMMON, and must be inserted at the start of the subroutine, together with Funds Transfer common area I_F.FTCOM.

The following variables are most likely to be used in the sub-routine:

FTLC$BC.PARAMS	Contains the FT BC PARAMETER record for the system
FTLC$LOCAL.CLEARING	Contains the FT LOCAL CLEARING record
R.CREDIT.ACCT( )	Contains the credit account record
R.DEBIT.ACCT( )	Contains the debit account record
R.CHARGE.ACCT( )	Contains the charge account record

All validation must be performed using the contents of R.NEW, the current record. Management of errors must cater for the fact that the routine will be executed on-line under user control, on-line automatically (when processing clearing tapes/files), and at end of day (when processing Standing Orders). The common variable AUTO.PROCESS will be set to “Y” when processing during the end of day, or automatically on-line. Error message processing should set ETEXT and call STORE.END.ERROR when NOT processing automatically, otherwise the routine should return when an error is found.

Overrides may be generated when processing manually on-line, in the standard manner, by setting TEXT and calling STORE.OVERRIDE.

Where local reference items are used to contain local clearing elements, a list of elements can be found in the fields REQ.LOCREF.NAME, REQ.LOCREF.APP and REQ.LOCREF.POS in the FT.LOCAL.CLEARING record, which give the location within the LOCAL.REF field in the specified application.

Example:

The following example illustrates use of local reference identifiers in *FT.LOCAL.CLEARING*. The items SCC.TXN.CODE is mandatory for BC transaction types:

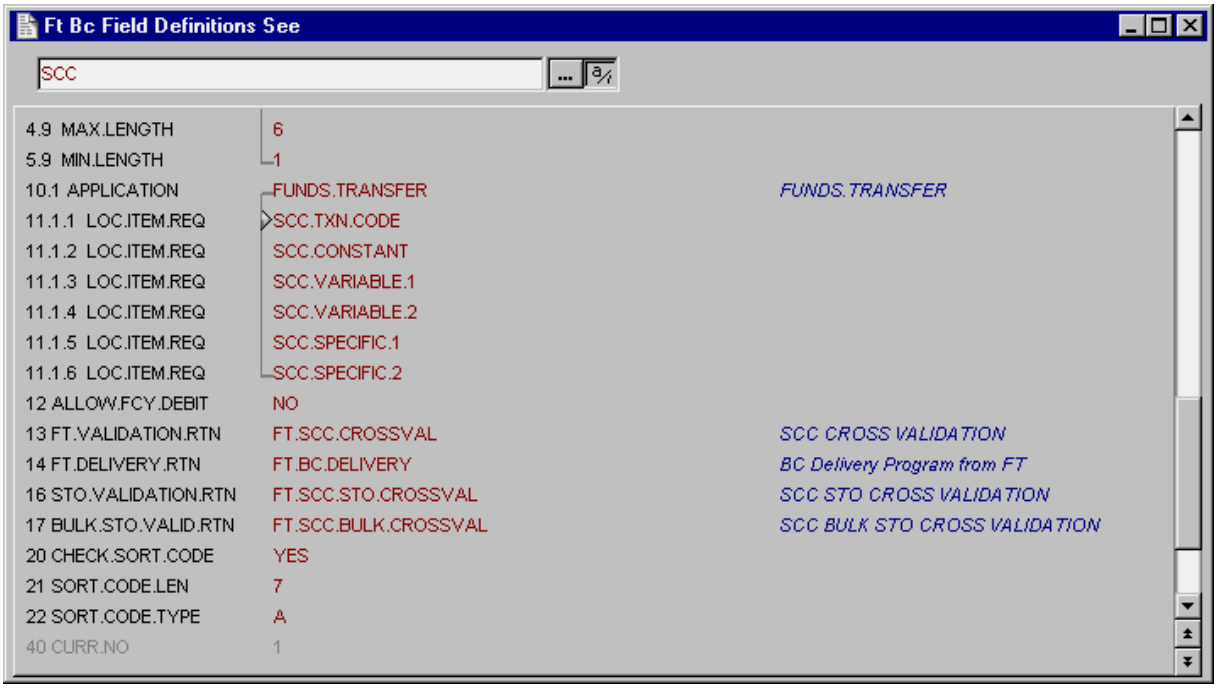


Figure 28 - FT BC Fields Definitions

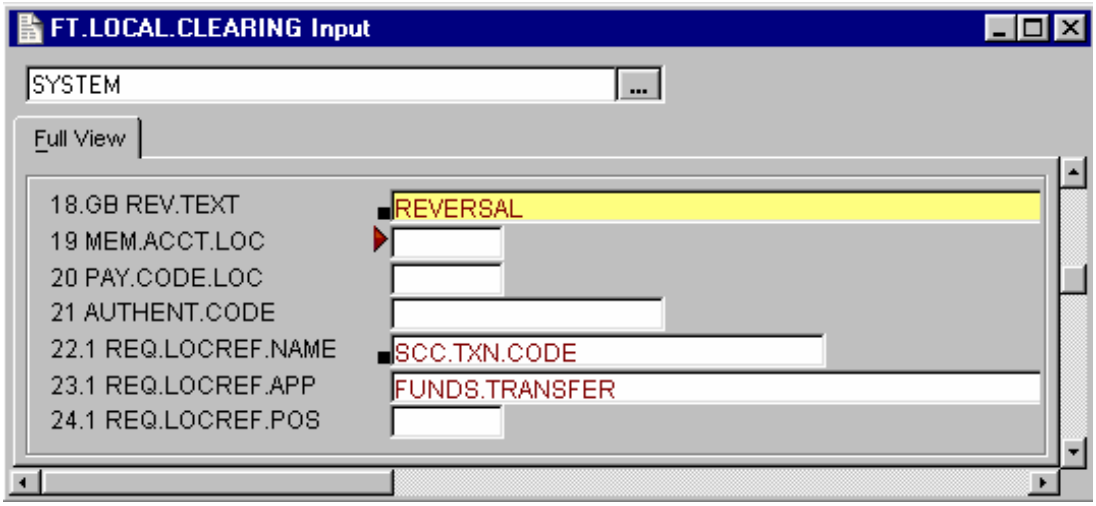


Figure 29 – FT.LOCAL.CLEARING Input Screen

This code will check to see if the local reference SCC.TXN.CODE contains a value in the *FUNDS.TRANSFER* record:

```
*
* SCC.TXN.CODE is taken from LOCAL.REFERENCE. It is a mandatory field and must
* be present. It's position is held in REQ.LOCREF.POS on FT.LOCAL.CLEARING for the
* corresponding REQ.LOCREF.NAME field.
*
      SCC.TXN.CODE.POS = ''
      LOCATE  'SCC.TXN.CODE'  IN  FTLC$LOCAL.CLEARING(FT.LC.REQ.LOCREF.NAME)<1,1>  SETTING
NAME.POS ELSE NAME.POS = ''
      IF NAME.POS THEN
         SCC.TXN.CODE.POS = FTLC$LOCAL.CLEARING(FT.LC.REQ.LOCREF.POS)<1,NAME.POS>
      END

      SCC.TXN.CODE = R.NEW(FT.LOCAL.REF)<1,SCC.TXN.CODE.POS>
      IF SCC.TXN.CODE = '' THEN
         AF = FT.LOCAL.REF;AV = SCC.TXN.CODE.POS
         ETEXT = 'INPUT MANDATORY'
         CALL STORE.END.ERROR
      END
```

Figure 30 - Check if local reference SCC.TXN.CODE contains a value

FT.DELIVERY.ROUTINE (FUNDS TRANSFER)

A routine may be called at authorisation of a Local Clearing Funds Transfer. This may be used to generate additional delivery messages or to update cross-reference files required in the local clearing process.

Format: Subroutine Name

Subroutine Name contains the name of the subroutine to be invoked.

Invoked: From FT.DELIVERY before generation of Standard delivery output

Arguments: None

Details:

The subroutine must contain the insert files I_F.FT.LOCAL.COMMON and I_F.FTCOM, which hold the local clearing common variables and the Funds Transfer common variables respectively.

Any of the variables described in the FT.VALIDATION.RTN may be used. Any error found should set ETEXT. This will be detected on return to FT.DELIVERY and cause an error to be generated by Funds Transfer.

Any additional delivery messages required must be generated by calling the subroutine APPLICATION.HANDOFF (see Standard subroutine guide for further details). Any delivery message used for local clearing must use the reserved range 1200 - 1300 of message types.

Examples:

This routine will update a cross-reference file, FT.BC.XREF when a local clearing funds transfer is authorised. The record will be deleted if a reversal is authorised. An additional check is made so that if a reversal is attempted and there is no record present on FT.BC.XREF, the reversal will be aborted.

```
SUBROUTINE FT.BC.DELIVERY

*************************************************************************
*
* This program will update a cross-reference file FT.BC.XREF which will be
* used to build the ASCII file to be sent to the local clearing centre.
* At the authorisation of an FT, a record will be entered and at reversal
* authorisation the record will be deleted. The ID to the cross-reference file
* will be the ID of the FT and the only data required is the current number of
* the FT in field 1.
*
*************************************************************************

$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.FUNDS.TRANSFER
$INSERT I_F.FT.BC.XREF
$INSERT I_F.FTCOM
$INSERT I_F.FT.LOCAL.COMMON

      GOSUB INITIALISE
      IF R.NEW(FT.RECORD.STATUS) = 'REVE' THEN
         GOSUB CHECK.RECORD.ON.FILE
         IF ETEXT THEN RETURN
         GOSUB DELETE.RECORD.FROM.FILE
      END ELSE
         GOSUB ADD.RECORD.TO.FILE
      END
      RETURN

*************************************************************************
INITIALISE:

      F.FT.BC.XREF = ''
      CALL OPF('F.FT.BC.XREF', F.FT.BC.XREF)

      BC.XREF.REC = ''
      ETEXT = ''
      READ.FAILED = ''

      RETURN

*************************************************************************

ADD.RECORD.TO.FILE:

      BC.XREF.REC<FT.BCX.CURRENT.NUMBER> = R.NEW(FT.CURR.NO)
      IF BC.XREF.REC<FT.BCX.CURRENT.NUMBER> = "" THEN
         BC.XREF.REC<FT.BCX.CURRENT.NUMBER> = '1'
      END
      BC.XREF.REC<FT.BCX.CR.VALUE.DATE> = R.NEW(FT.CREDIT.VALUE.DATE)
      IF BC.XREF.REC<FT.BCX.CR.VALUE.DATE> = "" THEN
         BC.XREF.REC<FT.BCX.CR.VALUE.DATE> = TODAY
      END
      CALL F.WRITE('F.FT.BC.XREF', ID.NEW, BC.XREF.REC) RETURN
```

Figure 31 - This routine will update a cross-reference file, FT.BC.XREF

```
*************************************************************************

CHECK.RECORD.ON.FILE:
*
* First check if the record is present in which case delete the record.
* Otherwise the payment has already been sent to the clearing system.
*
      CALL F.READ('F.FT.BC.XREF', ID.NEW, BC.XREF.REC, F.FT.BC.XREF, READ.FAILED)
      IF READ.FAILED THEN
         ETEXT = 'CANNOT REVERSE - PAYMENT ALREADY SENT TO CLEARING'
      END

      RETURN

*************************************************************************

DELETE.RECORD.FROM.FILE:

      CALL F.DELETE('F.FT.BC.XREF', ID.NEW)

      RETURN

***
   END
```

Figure 32- Check if reversal is attempted on FT.BC.XREF

STO.VALIDATION.RTN (STANDING.ORDER)

A subroutine may be defined to perform cross-validation of *STANDING.ORDER* records, which are paid through the local clearing method (*FUNDS.TRANSFER* BC TRANSACTION.TYPE).

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program.

Invoked: From STANDING.ORDER at cross-validation time, when PAY.METHOD [1,2] is BC.

Arguments: R.FT.LOCAL.CLEARING, R.FT.BC.PARAMETER
Where R.FT.LOCAL.CLEARING contains the FT.LOCAL.CLEARING record R.FT.BC.PARAMETER contains the FT.BC.PARAMETER record.

Details:

The contents of R.NEW should be validated according to local requirements. The validation should ensure that information entered in the STANDING.ORDER record is sufficient, and correct, in order to produce a valid BC FUNDS.TRANSFER record.

Error messages must be reported by setting ETEXT and calling STORE.END.ERROR in the usual manner.

Local reference items required in the Funds Transfer to be generated are entered in the fields FT.LOC.REF.NO (the position within local reference in Funds Transfer) and FT.LOC.REF.DATA. The correct position of local reference items within the LOCAL.REF field in FUNDS.TRANSFER can be checked using the variables REQ.LOCREF.NAME, REQ.LOCREF.APP and REQ.LOCREF.POS in the FT.LOCAL.CLEARING record.

Example:

The following subroutine performs specific validation for the Slovak clearing system:

```
 SUBROUTINE FT.SCC.STO.CROSSVAL(LOCAL.CLEARING.REC, BC.PARAM.REC)

*************************************************************************
*
* This routine will cross-validate fields used in the BC transactions for
* standing orders in the SCC (Slovak Clearing Centre) system.
*
*************************************************************************
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.COMPANY
$INSERT I_F.ACCOUNT
$INSERT I_F.ACCOUNT.PARAMETER
$INSERT I_F.STANDING.ORDER
$INSERT I_F.FT.LOCAL.CLEARING
$INSERT I_F.FT.BC.PARAMETER
*
* SCC.TXN.CODE is taken from LOCAL.REFERENCE. It is a mandatory field and therefore
* be present. It's position is held in REQ.LOCREF.POS on FT.LOCAL.CLEARING for the
* corresponding REQ.LOCREF.NAME field. This position number must be present in field
* FT.LOC.REF.NO on the standing order record.
*
      SCC.TXN.CODE.POS = ''
      LOCATE 'SCC.TXN.CODE' IN LOCAL.CLEARING.REC<FT.LC.REQ.LOCREF.NAME,1> SETTING NAME.POS
ELSE NAME.POS = ''
      IF NAME.POS THEN
         SCC.TXN.CODE.POS = LOCAL.CLEARING.REC<FT.LC.REQ.LOCREF.POS,NAME.POS>
      END

      LOCATE SCC.TXN.CODE.POS IN R.NEW(STO.FT.LOC.REF.NO)<1,1> SETTING TXN.POS ELSE TXN.POS
= ''
      IF TXN.POS = '' THEN
         AF = STO.FT.LOC.REF.NO
         ETEXT = 'INCOMPLETE LOCAL REF DETAILS'
         CALL STORE.END.ERROR
         RETURN
      END

*
* Make sure BENEFICIARY.ACCTNO satisfies MOD 11 validation
*
      IF R.NEW(STO.BEN.ACCT.NO) MATCHES '1NON' THEN
         AF = STO.BEN.ACCT.NO
         SAVE.COMI = COMI
         COMI = R.NEW(AF)
         GOSUB CHECK.ACCT.NO.VALIDATION
         IF ETEXT THEN
            CALL STORE.END.ERROR
            RETURN
         END
      END
      COMI = SAVE.COMI
*
* If no input is made to the beneficiary field then force input
* if the payment is to the Czech Republic (defined in field PTT.SORT.CODE
* on the FT.LOCAL.CLEARING file).
*
      IF R.NEW(STO.BANK.SORT.CODE) MATCHES LOCAL.CLEARING.REC<FT.LC.PTT.SORT.CODE> THEN
         IF R.NEW(STO.BENEFICIARY) = '' THEN
            AF = STO.BENEFICIARY;AV = 1
            ETEXT = 'BENEFICIARY MUST BE PRESENT'
            CALL STORE.END.ERROR
            RETURN
         END
      END

      RETURN
```

Figure 33 - The above subroutine performs specific validation for the Slovak clearing system

```
*************************************************************************

CHECK.ACCT.NO.VALIDATION:

$INSERT I_CHECK.ACCT.NO

      RETURN

***
   END
```

Figure 34 - Check Account Number Validation

BULK.STO.VALID.RTN (BULK.STO)

A sub-routine may be defined to perform cross-validation of Bulk Standing Order records, which are paid through the local clearing method (*FUNDS.TRANSFER* BC TRANSACTION.TYPE).

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program.

Invoked: From BULK.STO at cross-validation time, when PAY.METHOD[1,2] is BC.

Arguments: R.FT.LOCAL.CLEARING, R.FT.BC.PARAMETER
Where R.FT.LOCAL.CLEARING contains the FT.LOCAL.CLEARING record
R.FT.BC.PARAMETER contains the FT.BC.PARAMETER record.

Details:

The contents of R.NEW should be validated according to local requirements. The validation should ensure that information entered in the *BULK.STO* record is sufficient, and correct, in order to produce a valid BC FUNDS.TRANSFER record.

Error messages must be reported by setting ETEXT and calling STORE.END.ERROR in the usual manner.

Local reference items required in the Funds Transfer to be generated are entered in the fields FT.LOC.REF.NO (the position within local reference in FUNDS.TRANSFER) and FT.LOC.REF.DATA. The correct position of local reference items within the LOCAL.REF field in FUNDS.TRANSFER can be checked using the variables REQ.LOCREF.NAME, REQ.LOCREF.APP and REQ.LOCREF.POS in the FT.LOCAL.CLEARING record.

Example:

The following example performs the specific validation for the Bulk Standing Orders in the Slovak clearing system.

```
SUBROUTINE FT.SCC.BULK.CROSSVAL(LOCAL.CLEARING.REC, BC.PARAM.REC)

*************************************************************************
*
* This routine will cross-validate fields used in the BC transactions for
* bulk standing orders in the SCC (Slovak Clearing Centre) system.
*
*************************************************************************

$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.COMPANY
$INSERT I_F.ACCOUNT
$INSERT I_F.ACCOUNT.PARAMETER
$INSERT I_F.BULK.STO
$INSERT I_F.FT.LOCAL.CLEARING
$INSERT I_F.FT.BC.PARAMETER

      NO.OF.PAY.METHODS = DCOUNT(R.NEW(BST.PAY.METHOD), VM)
      FOR AV = 1 TO NO.OF.PAY.METHODS
         IF R.NEW(BST.PAY.METHOD)<1,AV>[1,2] = 'BC' THEN
*
* SCC.TXN.CODE is taken from LOCAL.REFERENCE. It is a mandatory field and therefore must
* be present. It's position is held in REQ.LOCREF.POS on FT.LOCAL.CLEARING for the
* corresponding REQ.LOCREF.NAME field. This position number must be present in field
* LOC.REF.NO on the bulk standing order record.
*
            SCC.TXN.CODE.POS = ''
            LOCATE 'SCC.TXN.CODE' IN LOCAL.CLEARING.REC<FT.LC.REQ.LOCREF.NAME,1> SETTING
NAME.POS ELSE NAME.POS = ''
IF NAME.POS THEN
               SCC.TXN.CODE.POS = LOCAL.CLEARING.REC<FT.LC.REQ.LOCREF.POS,NAME.POS>
            END

            LOCATE SCC.TXN.CODE.POS IN R.NEW(BST.LOC.REF.NO)<1,AV,1> SETTING TXN.POS ELSE
TXN.POS = ''
            IF TXN.POS = '' THEN
               AF = BST.LOC.REF.NO
               ETEXT = 'INCOMPLETE LOCAL REF DETAILS'
               CALL STORE.END.ERROR
               RETURN
            END
 *
 * Make sure BENEFIC.ACCTNO satisfies MOD 11 validation
 *
             IF R.NEW(BST.BENEFIC.ACCTNO)<1,AV> MATCHES '1NON' THEN
                AF = BST.BENEFIC.ACCTNO
                SAVE.COMI = COMI
                COMI = R.NEW(AF)<1,AV>
                GOSUB CHECK.ACCT.NO.VALIDATION
                IF ETEXT THEN
                   CALL STORE.END.ERROR
                   RETURN
                END
                COMI = SAVE.COMI
             END
 *
 * If no input is made to the beneficiary field then force input
 * if the payment is to the Czech Republic (defined in field PTT.SORT.CODE
 * on the FT.LOCAL.CLEARING file).
 *
             IF R.NEW(BST.BANK.SORT.CODE)<1,AV> MATCHES
LOCAL.CLEARING.REC<FT.LC.PTT.SORT.CODE> THEN
                IF R.NEW(BST.BENEFICIARY)<1,AV> = '' THEN
                   AF = BST.BENEFICIARY;AS = 1
                   ETEXT = 'BENEFICIARY MUST BE PRESENT'
```

Figure 35 - Bulk Standing Order Validation

```
                   CALL STORE.END.ERROR
                   RETURN
                END
             END
 *
          END
       NEXT AV       RETURN

 *************************************************************************

 CHECK.ACCT.NO.VALIDATION:

 $INSERT I_CHECK.ACCT.NO

       RETURN

 ***
    END
```

Figure 36 - Bulk Standing Order Validation

ACCOUNT.UPD.RTN (ACCOUNT)

Where a clearing system requires static information from the *ACCOUNT* file, a subroutine may be called to update an extract file.

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program.

Invoked: From ACCOUNT at authorisation before AUTH.RECORD.WRITE

Arguments: None

Details:

The contents of R.NEW contain the current ACCOUNT record. R.OLD will contain the previous authorised contents of the account record.

The FT.LOCAL.CLEARING record can be used top define the position of required local reference elements within the Account local reference field.

Note: This routine should not perform any validation.

CUSTOMER.UPD.RTN (CUSTOMER)

Where a clearing system requires static information from the *CUSTOMER* file, a subroutine may be called to update an extract file.

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on PGM.FILE as a type S program.

Invoked: From CUSTOMER at authorisation before AUTH.RECORD.WRITE

Arguments: None.

Details:

The contents of R.NEW contain the current CUSTOMER record. R.OLD will contain the previous authorised contents of the CUSTOMER record.

The FT.LOCAL.CLEARING record can be used to define the position of required local reference elements within the CUSTOMER local reference field.

Note: This routine should not perform any validation.

DIVERSION.RTN (Delivery)

This is the old method used for diverting messages from a standard carrier to the local clearing carrier. It was used for the Swiss Clearing system (SIC). However, to use this, changes are required to the Delivery system. Therefore, Generic Delivery was designed and it is this which should now be used if you wish to direct messages to a local clearing carrier (see the section Adding a new interface in the Delivery User Guide.

FT.TAPE.PARAMS

The *FT.TAPE.PARAMS* application manages import and export of data for the local clearing system(s) installed. Data is typically downloaded or uploaded onto tapes, or directly to a specified file. Subroutines and commands may be defined for each type of interface, which are used for:

- LOAD.CMD
- LOAD.ROUTINE
- UPDATE.ROUTINE
- CREATE.CMD
- CREATE.ROUTINE
- GENERATE.ROUTINE
- ENQ.PURGE.ROUTINE

LOAD.CMD

A command or routine may be executed, which will be used to download a tape or file into a specific directory on the system.

Format: Command name

Command name contains the name of any valid jBase command, which can be executed. This may also include a subroutine name, which can be executed. Multiple commands may be specified.

Invoked: From FT.TAPES$RUN when run with LOAD function

Arguments: None.

Details:

Typically a UNIX command will be specified prefixed by “SH -c “ to allow the command to run from jBase. This could also be specified within an InfoBasic subroutine.

The command/subroutine must download the tape/file into the directory FT.IN.TAPE, with the name Tape.Name. WORK, where Tape Name is the key to the FT.TAPE.PARAMS record.

Example:

Example of a load command:

```
KEY.............. PTT
------------------------------------------------------------------------
 1. 1 GB DESCRIPTION. PTT TAPE LOAD
 2 DIRECTION..... INWARD
 3. 1. 1 LOAD.CMD.... SH -c "cpio -ivcBm < /dev/rmt0"
 3. 2. 1 LOAD.CMD.... SH -c "mv LOAD.FILE FT.IN.TAPE/PTT
 3. 2. 2 LOAD.CMD.... .WORK"
```

Figure 37 - Example of a load command

This will download a tape using cpio and then move the downloaded file (LOAD.FILE in this case), to the file PTT.WORK in FT.IN.TAPE.

The above command could be coded in a subroutine if required.

LOAD.ROUTINE

A subroutine must be written to process the downloaded tape/file, to extract the main header information for the file. This contains information required so that an operator can verify the correct file/tape has been downloaded prior to updating T24 with the contents.

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program.

Invoked: From FT.TAPES$RUN when the LOAD function is used, after the LOAD.CMD has been executed

Arguments: TAPE.NAME, TAPE.NO, NO.RECS, CR.TOT, DR.TOT, CHECKSUM, EBS.CHECKSUM, TAPE.DATE, TAPE.EXPIRY, CALC.CR.TOT, CALC.DR.TOT

Where:

TAPE.NAME - Contains the name of the tape as defined in FT.TAPE.PARAMS
TAPE.NO - Contains the serial number of the tape (returned)
NO.RECS - Number of records contained in the tape (returned)
CR.TOT - Total of credit transactions (returned)
DR.TOT - Total of Debit transactions (returned)
CHECKSUM - The checksum contained in the tape (returned)
EBS.CHECKSUM - The calculated checksum (returned)
TAPE.DATE - Date of tape production (returned)
TAPE.EXPIRY - Date of tape expiry (returned)
CALC.CR.TOT - Calculated Credit Total (returned)
CALC.DR.TOT - Calculated Debit Total (returned).

Details:

The routine must process the tape which has been downloaded into a file in the directory FT.IN.TAPE with the name Tape.Name .WORK, where Tape.Name is the key to the FT.TAPE.PARAMS record passed as the first argument to the subroutine. Some systems will download a separate Header record and trailer record. These must be read from an agreed location, preferably the FT.IN.TAPE directory.

The subroutine must extract as many of the passed parameters as possible so that maximum details may be recorded.

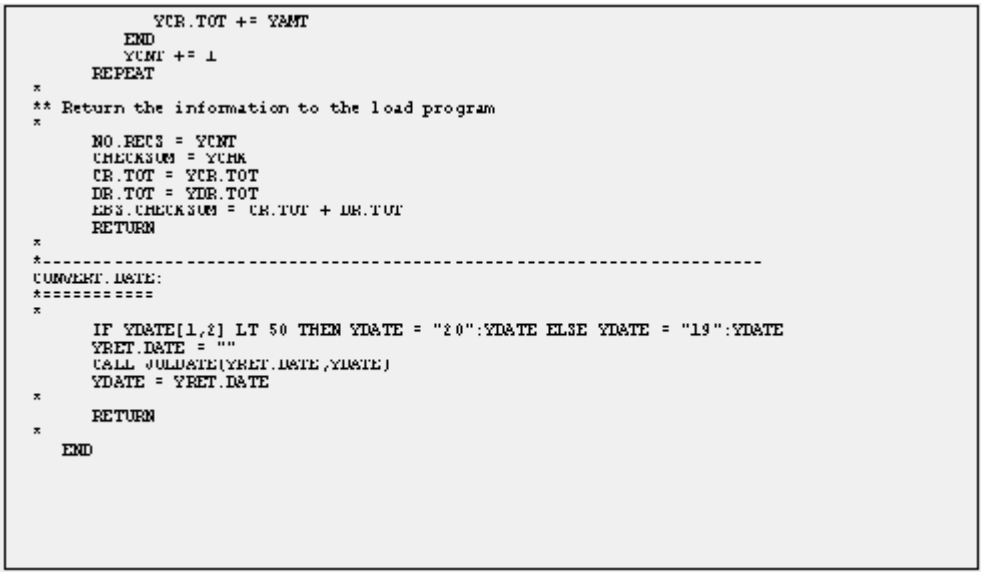
Any error detected must be returned in ETEXT.

Example:

The following example extracts the required details from a Swiss PTT tape, which has been downloaded into FT.IN.TAPE directory, record PTT.WORK.

```
* Version 3 19/01/94  GLOBUS Release No. 12.2.2 03/02/94
      SUBROUTINE FT.PTT.TAPE.LOAD(TAPE.NAME, TAPE.NO, NO.RECS, CR.TOT, DR.TOT, CHECKSUM,
EBS.CHECKSUM, TAPE.DATE, TAPE.EXPIRY, CALC.CR.TOT, CALC.DR.TOT)
*
$INSERT I_COMMON
$INSERT I_EQUATE
*
** This subroutine will extract the following header details from the
** downloaded PTT BED tape :-
**    TAPE.NO - Tape number header [1,10] field 1
**    NO.RECS - Total number of records contained on the tape
**    CR.TOT  - Total of credit entries
**    DR.TOT  - Total of Debit entries
**    CHECKSUM- From Total Line 999
**    EBS.CHECKSUM - Total Debit and Credit entries
**    ETEXT  - Set if errors
**
** The header record contains 2 80 block data strings.
** The header will be loaded into the FT.IN.TAPE file as TAPE.NAME. HDR
** The record is contained in TAPE.NAME.WORK.
*
      BLOCK.SIZE = 100                  ;* Block Size that tape should be downloaded
      YCR.TOT = ""                      ;* Running Credit total
      YDR.TOT = ""                      ;* Running Debit total
      YCHK = ""                         ;* Running Checksum total
      YCNT = ""                         ;* No of record count
      PTT.FILE = ""       F.FT.IN.TAPE = ""
*
      OPEN "","FT.IN.TAPE" TO F.FT.IN.TAPE ELSE
         ETEXT = "CANNOT OPEN FT.IN.TAPE FILE"
         RETURN
      END
*
** Extract tape number , date and expiry date from header
*
      HDR.ID = TAPE.NAME:".HDR"
      READ YR.HEAD FROM F.FT.IN.TAPE, HDR.ID ELSE
         ETEXT = "MISSING HEADER & IN FT.IN.TAPE":FM:HDR.ID
         RETURN
      END
*
      TAPE.NO = YR.HEAD[1,10]
      YDATE = YR.HEAD[123,5]
      GOSUB CONVERT.DATE
      TAPE.DATE = YDATE
      YDATE = YR.HEAD[129,5]
      GOSUB CONVERT.DATE
      TAPE.EXPIRY = YDATE
*
** Convert the dates from Julian to EBS format
*
*
** We now have to process the whole tape totalling the amounts etc.
*
      REC.ID = TAPE.NAME:".WORK"
      OPENSEQ "FT.IN.TAPE", REC.ID TO PTT.FILE ELSE
         ETEXT = "CANNOT OPEN & IN FT.IN.TAPE":FM:REC.ID
         RETURN
      END
*
      LOOP
         READBLK YREC FROM PTT.FILE, BLOCK.SIZE ELSE YREC = ""
      UNTIL YREC = ""
         YTXN = YREC[1,3]               ;* PTT Transaction code
         IF YREC[1,3] NE "999" THEN     ;* Not a Total record
            YAMT = YREC[40,10]
            YAMT = OCONV(YAMT,"MD2")
            YCHK += YAMT
         END ELSE
            YAMT = YREC[40,12]
            YAMT = OCONV(YAMT,"MD2")
```

Figure 38 - FT.IN.TAPE Directory showing details of Swiss PTT tape



```
            YCR.TOT += YAMT
         END
         YCNT += 1
      REPEAT
*
** Return the information to the load program
*
      NO.RECS = YCNT
      CHECKSUM = YCHK
      CR.TOT = YCR.TOT
      DR.TOT = YDR.TOT
      EBS.CHECKSUM = CR.TOT + DR.TOT
      RETURN
*
*-----------------------------------------------------------------------
CONVERT.DATE:
*************
*
      IF YDATE[1,2] LT 50 THEN YDATE = "20":YDATE ELSE YDATE = "19":YDATE
      YRET.DATE = ""
      CALL JULDATE(YRET.DATE,YDATE)
      YDATE = YRET.DATE
*
      RETURN
*
   END
```

Figure 39 - FT.IN.TAPE Directory showing details of Swiss PTT tape

UPDATE.ROUTINE

A subroutine must be defined to create *FUNDS.TRANSFER* records from the downloaded tape. This will be executed from the FT.TAPES application.

Format: Subroutine Name

Subroutine name contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program.

Invoked: From FT.TAPES$RUN when run with the UPDATE function

Arguments: TAPE.SEQ.NR, TAPE.NAME
Where TAPE.SEQ.NR is the sequence number allocated to the tape by T24.
TAPE.NAME is the name of the tape, the key to FT.TAPE.PARAMS.

Details:

The update routine must process the downloaded tape or file. Usually these processes will create FUNDS.TRANSFER transactions for the movements.

The tape record must be read from FT.IN.TAPE directory, using the TAPE.NAME. TAPE.SEQ.NR as a key. Once processing is complete, the *FT.TAPES* application will delete the downloaded file. For further details on the required processing see the Local Clearing User Guide.

CREATE.CMD

A command and/or subroutine can be entered, which when executed creates a tape or file. Typically this would be used to create a clearing tape or file.

Format: Command

Command contains the name of any jBase command or subroutine name to be executed.

Invoked: From FT.TAPES$RUN when run with the “CREATE” function

Arguments: None

Details:

Any executable command in jBase may be specified.

Example:

An example of a create command using cpio for production of a tape from the file BACS.OUT.FILE.

```
5 UPDATE.ROUTINE....
6. 1. 1 CREATE.CMD.. sh -c "find FT.IN.TAPE/BACS.OUT.FILE -print
6. 1. 2 CREATE.CMD..  -depth | cpio -ovcdB > /dev/rmt0"
7 CREATE.ROUTINE....
```

Figure 40 - Example of Create command from BACS.OUT.FILE

CREATE.ROUTINE

A routine may be specified which is used to create or manipulate an extract file that can be downloaded using the CREATE.COMMAND.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program

Invoked: From FT.TAPES$RUN when run with the “CREATE” function

Arguments: TAPE.SEQ.NR, TAPE.NAME.
Where TAPE.SEQ.NR is the sequence number allocated by the *FT.TAPES* application
TAPE.NAME is the key to the *FT.TAPE.PARAMS* record and identifies the type of file being processed

Details:

The create routine will be used to create an output file for the clearing system/interface in use. The file must be written to TAPE.NAME. TAPE.SEQ.NR within the directory FT.IN.TAPE.

GENERATE.ROUTINE

A routine may be specified which is used to create an extract file that can be downloaded using the CREATE.COMMAND.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must be defined on PGM.FILE as a type S program

Invoked: From FT.TAPES$RUN when run with the “GENERATE” function

Arguments: TAPE.SEQ.NR, TAPE.NAME, NO.RECS, CR.TOT, DR.TOT, CHECKSUM.

Where TAPE.SEQ.NR is the sequence number allocated by the FT.TAPES application
TAPE.NAME is the key to the FT.TAPE.PARAMS record and identifies the type of file being processed
NO.RECS returns the number of records contained in the extract file
CR.TOT returns the total of credit transactions
DR.TOT returns the total of debit transactions
CHECKSUM contains a calculated checksum.

Details:

The generate routine will create a file in the required format for the clearing system/interface in use. The file must be written to TAPE.NAME. TAPE.SEQ.NR within the directory FT.IN.TAPE.

The CR.TOT, DR.TOT, NO.RECS and CHECKSUM should be calculated and returned where available.

ENQ.PURGE.ROUTINE

Local clearing interfaces may update a file, which is used to for reporting the contents of the tape. A routine to purge the file on a regular basis may be specified here.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must be defined on PGM.FILE as a type S program

Invoked: From FT.LOCAL.DATA.PURGE in the BATCH *FT.START.OF.DAY*

Arguments: PARAM.ID, R.FT.TAPE.PARAMS.
Where PARAM.ID is the key to the FT.TAPE.PARAMS record
R.FT.TAPE.PARAMS is the FT.TAPE.PARAMS record with the key PARAM.ID.

Details:

The enquiry purge routine must determine whether the data is to be purged according to the `ENQUIRY.FILE.DAYS` field in R.FT.TAPE.PARAMS.

Example:

```
SUBROUTINE FT.BACS.TAPE.DATA.PURGE(PARAM.ID,R.FT.TAPE.PARAMS)
 *
 * Routine to delete FT.BACS.TAPE.DATA records which are older than
 * the no of days specified on the FT.TAPE.PARAM record
 * for the tapes parameterised.
 *
 $INSERT I_COMMON
 $INSERT I_EQUATE
 $INSERT I_F.FT.TAPE.PARAMS
 $INSERT I_F.FT.BACS.TAPE.DATA
 *
 * Open Files
 *
       F.FT.BACS.TAPE.DATA = ''
       FILE.FT.BACS.TAPE.DATA = 'F.FT.BACS.TAPE.DATA'
       CALL OPF(FILE.FT.BACS.TAPE.DATA,F.FT.BACS.TAPE.DATA)
 *
       CUTOFF.DATE = ICONV(TODAY,'D') - R.FT.TAPE.PARAMS<FT.TP.ENQUIRY.FILE.DAYS>
       CUTOFF.DATE = OCONV(CUTOFF.DATE,'D/E')
       CUTOFF.DATE = CUTOFF.DATE[7,4]:CUTOFF.DATE[4,2]:CUTOFF.DATE[1,2]
 *
       STATEMENT = "SELECT ":FILE.FT.BACS.TAPE.DATA:" WITH ID LIKE '":PARAM.ID:"...' AND
WITH GLOBUS.SYSTEM.DATE LT ":CUTOFF.DATE

       CRT STATEMENT
       EXECUTE STATEMENT
       IF @SELECTED THEN
          DATA "Y"
          EXECUTE "DELETE ":FILE.FT.BACS.TAPE.DATA
       END
 *
       RETURN
 *
END
```

Figure 41 - Enquiry purge routine R.FT.TAPE.PARAMS

AC.ENTRY.PARAM

The *AC.ENTRY.PARAM* application controls the Generic Accounting Interface. It contains booking details of entries supplied in an external ASCII file, and the layout of the file. The standard routine AC.INWARD.FILE should be used as the UPDATE routine in *FT.TAPES* to run the generic interface.

Additional validation is possible by calling user routines.

DATA.VAL.RTN

A subroutine may be specified to validate / convert data extracted from the Ascii file. It may be either a standard IN2 routine, or a user defined routine.

Format: Subroutine name

Subroutine name contains the name of the user-defined subroutine to be invoked. The routine must exist in the VOC file.

Invoked: From AC.INWARD.FILE

Arguments: None

Details:

The value extracted from the tape is supplied in the variable COMI and may be modified if required. An error in the format of the value extracted should be returned in the variable ETEXT.

VALIDATION.RTN

A subroutine may be specified to validate the contents of the final extracted entry record, in order to either reject the entry, or suspense the account number

Format: Subroutine name

Subroutine name contains the name of the user-defined subroutine to be invoked. The routine must exist in the VOC file.

Invoked: From AC.INWARD.FILE

Arguments: IN.ENTRY - (In) The formatted accounting entry
REP.ENTRY - (In/Out) The reporting entry from the AC.INWARD.ENTRY record which has been constructed
RTN.ERR - (Out) Should return any error message found causing the entry NOT to be processed
RTN.OVE - (Out) Should return any message which causes the account to replaced with a suspense account.

Details:

This routine should be used where the entry requires specific validation based on the contents of the record. Using the `ENQ.FILE.LOC` field in the application, extracted data can be mapped to any field in AC.INWARD.ENTRY, which can then be checked in the routine written here. For example a SORT.CODE extracted could be mapped to LOCAL.REF value 2, which can then be validated in a user routine specified at this point.

Local Statutory Reporting

Introduction

T24 allows subroutines to be called at input time for the applications FUNDS.TRANSFER. DATA.CAPTURE, TELLER and DRAWINGS, which can validate and default local reference values used in local reporting. Typically an activity code will be allocated according to the type of contract entered.

BANK.RETURN.PARAMS

This application allows definition of subroutines to be called from the above mentioned applications at input time to allow defaulting and validation of local reference items. The record applicable to the company is defined in `LOCAL.PROCESS` in the *COMPANY* record.

INIT.ROUTINE

A routine may be called to initialise local variable required for the processing for the associated APPLICATION.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must

be defined on *PGM.FILE* as a type S program

Invoked: From *FUNDS.TRANSFER* in the initialisation section

From *TELLER* in the INITIALISE section

From *DATA.CAPTURE* in the initialisation

From *DRAWINGS* in the INITIALISE section

Arguments: None

Details:

This routine should be used to initialise variables specific to the application when the application is first used.

The application in question can be determined from the common variable APPLICATION.

CHECK.INPUT.ROUTINE

A routine can be called at input time to validate and default local reference items for the associated APPLICATION.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must be defined on PGM.FILE as a type S program

Invoked: From FUNDS.TRANSFER prior to the call to FT.COMPLETE.XVALIDATION

From TELLER in the section CROSSVALIDATION

From DATA.CAPTURE at cross-validation

From DRAWINGS in the section CROSSVALIDATION

Arguments: None

Details:

The routine may use any of the system common variables available. The current contract record is held in R.NEW. The *BANK.RETURN.PARAMS* record is held in common in the variable R.BANK.RETURN.PARAMS

The fields `LOCAL.REF.DESC` and `LOCAL.REF.POSN` may be defined for each application, and allow the correct value within the `LOCAL.REF` field to be identified for validation purposes.

Takeover / Installation Customisation

Introduction

In order to assist the take-over of existing systems when installing T24, the system allows linkage of subroutines. Subroutines will usually be required to perform processing which allows existing file structures to be mapped into the appropriate T24 data files.

ALT.ACCT.PARAMETER

The application *ALT.ACCT.PARAMETER* allows definition of the details of the account number structure in the existing system. The old account number is stored in a cross-reference table linking it to the new T24 account number, called ALTERNATE.ACCOUNT.

CHECKDIGIT.TYPE

The existing account number check-digit options for a T24 account are supported, but where these do not match, a subroutine may be used to perform the validation and formatting of the alternate account number.

Format: @Subroutine Name

Subroutine contains the name of the subroutine to be executed. It must be defined on *PGM.FILE* as a type S program

Invoked: From *ACCOUNT* at field validation time for ALT.ACCT.ID

Arguments: None.

Details:

This routine should validate and format the account number; the basic type validation (i.e. alpha, numeric) will have already been performed before the routine is called. Any account mask specified with the routine will not be validated, and should be performed in the subroutine.

The alternate number entered is contained in COMI, and should be returned in this variable. Any error message should be returned in ETEXT.

Example:

```
      SUBROUTINE ANY.ACCT
$INSERT I_COMMON
$INSERT I_EQUATE
*
* This routine will check that the account number matches
* the format AA99999A
* The account number may be entered as "2aln-5nla" and
* will be correctly formatted
*
      IN.ACCT = COMI                    ; * Incoming account no
      BEGIN CASE
         CASE NOT(IN.ACCT MATCHES "2A1N0N1A")
            ETEXT = "FORMAT MUST BE 'AA99999A'"
         CASE IN.ACCT MATCHES "2A5N1A"  ; * Okay
         CASE 1                         ; * Format number
            IN.ACCT = IN.ACCT[1,2]:FMT(IN.ACCT[3,LEN(IN.ACCT)-3],"5'0'R"):IN.ACCT[1]
      END CASE
      IF NOT(ETEXT) THEN COMI = IN.ACCT
*
      RETURN
   END
```

Figure 42 - Example subroutine ANY.ACCT

EBS.AUTO.FUNCTION

The system allows a subroutine to be inserted when building the input buffer when contents need to be calculated, and when maintaining totals for the records processed.

INPUT.BUFFER and KEYSTROKES

A routine can be called to build the input buffer required for the automatic key input.

Format: @Subroutine Name

Subroutine contains the name of the subroutine to be executed.

Invoked: From *EBS.AUTO.FUNCTION*

Arguments: LAST.ID, INPUT$BUFFER
Where LAST.ID contains the id of record being processed
INPUT$BUFFER contains the current input buffer.

Details:

Additional input should be added to INPUT$BUFFER, separated by spaces.

Example:

The following subroutine will cycle the interest review frequency on mortgage records to the next date.

```
  SUBROUTINE E.MG.INT.REV.FREQ(LAST.ID,VAR)
*
*
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.MG.MORTGAGE
*
*
*
      F.MG.MORTGAGE = ''
      CALL OPF('F.MG.MORTGAGE',F.MG.MORTGAGE)
      R.MG = ''; ETEXT = ''
      CALL F.READ('F.MG.MORTGAGE',LAST.ID,R.MG,F.MG.MORTGAGE,ETEXT)
      FIELD1 = R.MG<MG.INT.REVIEW.FREQ>[1,8]
*
      COMI = R.MG<MG.INT.REVIEW.FREQ>
      CALL CFQ
      FIELD2 = COMI
*
      VAR = FIELD1:C.F:FIELD2
      RETURN
*
   END
```

Figure 43 - Example subroutine E.MG.INT.REV.FREQ

TOTAL.FIELD

A subroutine can be used to maintain totals used for confirmation that the selection is correct.

Format: @Subroutine Name

Subroutine contains the name of the subroutine to be executed. Must be defined on *PGM.FILE* as TYPE S.

Invoked: From *EBS.AUTO.FUNCTION*

Arguments: REC.ID, RESULT
Where REC.ID contains the id of record being processed
RESULT contains the value to be added to the running total.

Details:

The calculated/derived amount should be returned in RESULT. For example a routine may be written to convert foreign amounts to local and return a local total.

TAKEOVER.MAPPING

Subroutines may be defined in TAKEOVER.MAPPING to format data into the expected T24 format from the source files, and to manipulate data according to specific rules. For example a cross-reference table may need to be maintained in order to build the correct link between T24 records.

DATA.SUBROUTINE

A subroutine may be called when data has been extracted from the source file. This is called prior to the update of the T24 file.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. Must be defined on *PGM.FILE* as TYPE S.

Invoked: From TAKEOVER.MAPPING$RUN after allocation of id and performing mapping into the T24 record as specified. This routine will be called when the *TAKEOVER.MAPPING* is run in report and update modes

Arguments: None

Details:

The current record id is contained in ID.NEW and will be used to write the record. The id may be manipulated at this point. R.NEW will contain the contents of the file mapped according to the definitions in TAKEOVER.MAPPING. If data is to be manipulated, it should be mapped into a field in the record, and the subroutine should perform the necessary changes.

UPDATE.SUBROUTINE

A subroutine may be invoked after extraction of data, just before the write to the unauthorised file is executed.

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. Must be defined on PGM.FILE as TYPE S.

Invoked: From TAKEOVER.MAPPING$RUN after mapping of data as specified and possible manipulation by the DATA.SUBROUTINE. Called only in

UPDATE mode.

Arguments: None.

Details:

The current record id is contained in ID.NEW and will be used to write the record. The id may be manipulated at this point. R.NEW will contain the contents of the file mapped according to the definitions in TAKEOVER.MAPPING. A routine called at this point may be used to update cross-reference files used for take-over purposes.

Example:

The following example has been used to update *DE.ADDRESS* for SWIFT addresses. The incoming key is a Customer number, which must be converted to a valid DE.ADDRESS key.

```
    SUBROUTINE DATA.MAP.DEAD1
*
*     DE.ADDRESS IS SWIFT.1
*
$INSERT I_F.SECURITY.MASTER
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
*
    IF LEN(FUNCTION) > 1 THEN RETURN
*
    F.READ.REC = ""
    ER = ""
    TEST.ADD.IND = ID.NEW[LEN(ID.NEW)-1,LEN(ID.NEW)] + 0
    IF TEST.ADD.IND > 1 THEN ;* Test records have suffix of 1 OR 2
       ETEXT = 0:VM:ID.NEW:VM:"IGNORED FOR THIS RUN"
       ID.NEW = ''
       RETURN
    END
    FILE.NAME = 'F.CUSTOMER'
    F.CUST.APPLICATION = ''
    CALL OPF (FILE.NAME, F.CUST.APPLICATION)
    ID.NEW = ID.NEW[1,LEN(ID.NEW)-2]
    CALL F.READ(FILE.NAME,ID.NEW,F.READ.REC,F.CUST.APPLICATION,ER)
    IF ER THEN  ;* Missing Customer
       ETEXT = 0:VM:ID.NEW:VM:"CUSTOMER LOAD HAS FAILED" ID.NEW = ''
    END ELSE
       SAVE.CUST.CODE = ID.NEW
       ID.NEW = ID.COMPANY:'.C-':SAVE.CUST.CODE:'.SWIFT.1'
    END
*
    RETURN
*
  END
```

Figure 44 - Example update subroutine DATA.MAP

Limits

Introduction

In order to allow complex calculations of risk factors to be applied to limit products or sub-products, the *LIMIT.REFERENCE* application allows definition of a subroutine, which may be invoked. The subroutine will return the amount of a given transaction to be recorded in the limits system.

LIMIT.REFERENCE

Format: Subroutine Name

Subroutine contains the name of the subroutine to be executed. Must exist as a VOC entry.

Invoked: From LIMIT.CHECK at input (validation).

Arguments: TRANSACTION.ARGUMENTS
Where TRANSACTION.ARGUMENTS contains a dynamic array of details. The layout of this array is contained in the insert I_LIMIT.SROUTINE.
TRANSACTION.ARGUMENTS should be returned with the derived limit amount.

Details:

The following details are contained in TRANSACTION.ARGUMENTS.

<1> LIAB.ORIG	The liability number (element 1 of the Limit key). This is a T24 Customer number. Always present.
<2> CUST.NO	The customer within the liability group (element 4 of the limit key). Always present.
<3> REF.NO	The limit reference number formatted with leading zeroes. Element 2 of the limit key. Always present.
<4> SER.NO	The limit serial number formatted with leading zeroes. Element 2 of the limit key. Always present.
<5> TXN.REF	The T24 transaction id being processed. Always present.
<6> COMPANY.MNE	The mnemonic of the company the transaction is entered in. Note that the transaction company may be different to the company in which limits are held. Always present.
<7> COMPANY.ID	The company code of the transaction. See above. Always present.
<8> TXN.PERIOD	The processing date used to determine time to maturity. This is TODAY when the system is online or processing application Close of Business, and PERIOD.END when running the percentage revaluation process during Close of Business. Always present.
<9> TXN.DATE	The maturity date for the transaction to be recorded in the limits system. This may be a date, or a number of days notice. Always present.
<10> TXN.CCY	The transaction currency to be recorded in the limits system. Always present.
<11> TXN.AMT	The full amount of the transaction as handed to the limits system. Note that for FX transactions,

	this contains two values: value 1 is the BUY amount, value 2 is the SELL amount. Always present. Note also that the sign of the amounts passed should not be changed.
<12> OTH.CCY.OR.COMMITM	Contains the following: For FX contracts, the SELL currency For LD commitment contracts, “Y” to indicate a commitment The value “NR” if the On-line limit amount is not to be reduced.
<13> DEAL.DESK	The dealer desk of the deal. Not used.
<14> ACC.CO	The company of the account when an account limit is processed. Note that percentage processing is ignored for accounts.
<15> ACC.NO	The account number for an account limit. See ACC.CO.
<16> ACC.BAL	The account balance to be considered. See ACC.CO
<17> ACC.CCY	The account currency of the associated ACC.BAL.
<18> CURR.NO	The current number of overrides in the T24 transaction.
<19> FIND.REC	Not relevant to percentage processing.
<20> CALL.TIME	Indicator for error/override processing. Can be: Null - On-line overrides require response ‘U’ - Batch update, always update regardless of error ‘E’ - Batch, return if error ‘V’ - On-line verify no transaction details, i.e. called from LIMIT ‘B’ - Batch verify no transaction details, i.e. called from LIMIT.
<21> CALL.ID	Indicates whether LIMIT.CHECK has been called in “VAL”, validation mode, or “DEL”, deletion mode. Note that the subroutine will only be invoked in “VAL” mode.

Access to Transaction Record

Processing to calculate the correct limit amount may require analysis of the transaction record. This can be extracted as follows:

ONLINE

The contract is contained in the common element R.NEW. The last authorised version is in R.OLD, the last unauthorised version is R.NEW.LAST.

END.OF.DAY

The contract should be read from the underlying application file. The application should be determined from the TXN.REF and the transaction file opened. When opening the file, the company mnemonic should be specified in the call to OPF to ensure the correct file is used.

Returning Information

The TRANSACTION.ARGUMENTS should contain the following.

<1> Amount 1 The amount derived to be used in updating limits. This must be the in the currency passed in TXN.CCY, and must be returned with the same sign as the passed TXN.AMT

<2> Amount 2 For FX deals only this should contain the equivalent percentage amount of the other side of the deal, in the currency passed in OTH.CCY.OR.COMMITM.

ETEXT should be returned if an error is encountered.

An example routine follows:

```
0001:       SUBROUTINE LIMIT.ALLOC.PERC.LOAN(LIMIT.ARGUMENTS)
0002: *
0003: $INSERT I_COMMON
0004: $INSERT I_EQUATE
0005: $INSERT I_LIMIT.SROUTINE
0006: $INSERT I_F.LD.LOANS.AND.DEPOSITS
0007: $INSERT I_F.CUSTOMER
0008: $INSERT I_F.LIMIT.REFERENCE
0009: *
0010: ** This is an example subroutine to demonstrate the ability to return
0011: ** an amount for an LD deal for use in the limits structure.
0012: ** Arguments are passed in LIMIT.ARGUMENTS, a dynamic array defined in
0013: ** I_LIMIT.SROUTINE
0014: ** The only argument returned is the amount. Note for FOREX contracts
0015: ** 2 amounts separated by @VM will be required for both sides of the deal
0016: ** Errors are returned in ETEXT
0017: **
0018: ** The routine will
0019: ** Look at the LD record. The customer will be checked to extract the
0020: ** sector code, which will in turn be checked against the LOCAL.REFEERNCE
0021: ** field LIMIT.SECTOR in LIMIT.REFERENCE. If found the associated LIMIT
0022: ** PERC will be used, if not 100% is assumed.
0023: ** The sector will also be checked against the LOCAL.REFERENCE item
0024: ** BLOCKED.SECTOR in the LD contract. If it matches an error is returned
0025: ** (Note this would probably be better as a Version Validation routine in
0026: ** practice)
0027: *
0028:
0029:       RETURNED.AMOUNT = ""                ; * Initialise
0030: *
0031: ** Check first that the update applies to the LD application
0032: *
0033:       IF LIMIT.ARGUMENTS<LI.SUBR.TXN.REF>[1,2] = "LD" THEN
0034: *
0035: ** Open files etc
0036: ** Argument TXN.REF.MNE contains the deal number | company code
0037: ** Open the LD file in the correct company
0039:          CO.CODE = LIMIT.ARGUMENTS<LI.SUBR.COMPANY.MNE>
0040:          CONTRACT.ID = LIMIT.ARGUMENTS<LI.SUBR.TXN.REF>
0041:          LIMIT.REF.ID = LIMIT.ARGUMENTS<LI.SUBR.REF.NO> + 0   ; * Strip leading zeros
0042: *
0043:          F.CUSTOMER = ""
0044:          CALL OPF("F.CUSTOMER", F.CUSTOMER)
0045: *
0046:          LD.FILE = "F":CO.CODE:".LD.LOANS.AND.DEPOSITS"
0047:          F.LD.FILE = ""
0048:          CALL OPF(LD.FILE, F.LD.FILE)
0049: *
0050:          F.LIMIT.REFERENCE = ""
0051:          CALL OPF("F.LIMIT.REFERENCE", F.LIMIT.REFERENCE)
0052: *
0053: ** Extract customer number, read record to get sector
0054: *
0055:          CUSTOMER.NO = LIMIT.ARGUMENTS<LI.SUBR.CUST.NO>
0056:          CUST.REC = ""
0057:          CALL F.READ("F.CUSTOMER", CUSTOMER.NO, CUST.REC, F.CUSTOMER, "")
0058:          SECTOR.CODE = CUST.REC<EB.CUS.SECTOR>
0059: *
0060: ** Get the LD record
0061: ** If running on-line it will be in R.NEW in common. In end of day
0062: ** the record should be read from the file
0063: *
0064:          IF RUNNING.UNDER.BATCH THEN
0065:             LD.REC = ""
0066:             CALL F.READ(LD.FILE, CONTRACT.ID, LD.REC, F.LD.FILE, "")
0067:          END ELSE
0068:             MATBUILD LD.REC FROM R.NEW
0069:          END
```

Figure 45 - Example routine for LIMIT REFERENCE

```
0070: *
0071: ** Check that the SECTOR is not blocked in the LD record
0072: *
0073:           LOCATE SECTOR.CODE IN LD.REC<LD.LOCAL.REF,1,1> SETTING BLOCKED.POS THEN
0074:              ETEXT = "CUSTOMER SECTOR CODE IS BLOCKED"
0075:           END ELSE
0076: *
0077: ** Now check the sector against the limit ref record
0078: *
0079:              LIMIT.REF.REC = ""
0080:              CALL F.READ("F.LIMIT.REFERENCE", LIMIT.REF.ID, LIMIT.REF.REC,
 F.LIMIT.REFERENCE, "")
0081: *
0082:              LOCATE SECTOR.CODE IN LIMIT.REF.REC<LI.REF.LOCAL.REF,1,1> SETTING
 SECTOR.POS THEN
0083:                 SECTOR.PERC = LIMIT.REF.REC<LI.REF.LOCAL.REF,2,SECTOR.POS>
0084:              END ELSE
0085:                 SECTOR.PERC = 100          ; * default
0086:              END
0087: *
0088: ** Calculate amount percentage
0089: *
0090:              TXN.AMT = LIMIT.ARGUMENTS<LI.SUBR.TXN.AMT>
0091:              TXN.CCY = LIMIT.ARGUMENTS<LI.SUBR.TXN.CCY>
0092:              RETURNED.AMOUNT = TXN.AMT * SECTOR.PERC / 100
0093:                 CALL EB.ROUND.AMOUNT(TXN.CCY, RETURNED.AMOUNT, "", "")     ; * Round to
currency
0094: *
0095:           END
0096: *
0097:        END
0098: *
0099:        LIMIT.ARGUMENTS = RETURNED.AMOUNT
0100: *
0101: PGM.EXIT:
0102:        RETURN
0103: *
0104:     END
```

Figure 46 - Example routine for LIMIT REFERENCE

Company Customisation

Introduction

In order to allow for different account number structures and check-digit calculations, the check-digit type may be defined as a user routine, which will perform the desired formatting and check-digit validation. This routine should also return the next available id if called from the *ACCOUNT* application with F3 or F2 entered at awaiting id. The routine is specified in the *COMPANY* record.

Company

ACCT.CHECKDIG.TYPE

Format: @Subroutine Name

Subroutine contains the name of the subroutine to be executed. Must exist as a VOC entry.

Invoked: From I_CHECK.ACCT.NO and GET.NEXT.ID.

Arguments: None

The following common variables should be used in the subroutine.

COMI - contains the account number, or portion of the account number to be validated. When called from GET.NEXT.ID the following

additional elements are supplied:
COMI<1> - Next account number from locking
COMI<2> - “NEW” to denote new number required
COMI<3> - “F” is F3 requested, “B” if F2 requested

ETEXT - returned if there is an error in the account number supplied.

Details:

The following example formats an account number to the number of digits in the ACCOUNT.MASK in the company record. There is no check-digit required.

```
0001:       SUBROUTINE TEST.ACCOUNT.CHECK
0002: *
0003: $INSERT I_COMMON
0004: $INSERT I_EQUATE
0005: $INSERT I_F.COMPANY
0006: *
0007: ** Test routine to allow an account of any checkdigit formatted to the
0008: ** account mask length
0009: *
0010:       IN.ACC.NO = COMI<1>                ; * Supplied A/c No
0011:        NEW.IND = COMI<2>                   ; * Will be set if next id       0012:
BACK.FORW = COMI<3>              ; * Set to F if F3 or B id F2
0013:       COMI = COMI<1>                     ; * Strip of other items
0014: *
0015:       MASKLEN = COUNT(R.COMPANY(EB.COM.ACCOUNT.MASK),"#")     ; * Length
0016: *
0017:       IF NEW.IND THEN                    ; * Get next account number
0018:          IF BACK.FORW = "F" THEN
0020:          END ELSE
0021:             IN.ACC.NO -= 1
0022:          END
0023:       END
0024: *
0025: ** Format the account number to the number in the mask
0026: *
0027:       IF LEN(COMI) GT MASKLEN THEN
0028:          ETEXT = "ACCOUNT NUMBER TOO LONG"
0029:       END ELSE
0030:          COMI = STR('0',MASKLEN-LEN(IN.ACC.NO)):IN.ACC.NO
0031:       END
0032: *
0033:       RETURN
0034:    END
```

Figure 47 - Example subroutine TEST.ACCOUNT.CHECK