FINACLE 7.0

DBA Training Manual

SCRIPTING USERHOOKS

ExpertEdge Software

Info.training@cwlgroup.com

Document number:		VersionRev:	1.1
Authorized by:	Olawuwo Shade	Signature/Date:	

Document revision list

Ver.Rev	**Date**	**Author**	**Description**
1.00	01-02-2011	Arije Abayomi	*Original version*

TABLE OF CONTENTS

1 SCRIPT ENGINE BASED CUSTOMIZATION 1

1.1 WHAT IS 'SCRIPT ENGINE BASED CUSTOMISATION'? 1

1.2 WHAT IS A FINACLE™ USER HOOK? 2

1.3 REPOSITORIES AND CLASSES 2

1.4 RETURN STATUS OF THE FUNCTION. 2

2 GENERAL SCRIPTING FUNCTIONS 3

2.1 GET THE ACCOUNT DETAILS INTO BANCS REPOSITORY 3

2.2 GET THE SCHEME DETAILS INTO BANCS REPOSITORY 7

2.3 BACID TO ACCT ID AND NAME 8

2.4 CONVERT AN AMOUNT FROM ONE CURRENCY TO ANOTHER 10

2.5 ROUNDOFF AN AMOUNT 11

2.6 VALIDATE DATE 13

2.7 VALIDATE REFERENCE CODE:: VRP1.6.38.01 13

2.8 SELECTION OF FIELDS FROM DATABASE 14

2.9 USING A SQL STATEMENT IN A SCRIPT 15

2.10 ACCOUNT BALANCE AS ON A GIVEN DATE: 16

2.11 VALIDATE CRNCY CODE 17

3 ONLINE SCRIPTING FUNCTIONS 17

3.1 SET THE DEFAULT ATTRIBUTE OF A FIELD IN THE FORM 17

3.2 SET THE DEFAULT VALUE FOR A FIELD IN THE FORM DURING FORM LOAD. 19

3.3 CHANGE THE FIELD VALUE IN THE FORM. 21

3.4 INQUIRE THE FIELD VALUE IN THE FORM. 22

3.5 CHANGE THE ATTRIBUTE OF A FIELD IN THE FORM. 25

3.6 GET THE 'POST-COMMIT' VALUE FROM A FIELD IN THE FORM. 27

3.7 EXECUTING A KEY-BASED EVENT FROM THE SCRIPT. 28

3.8 CALLING GENERAL PARAMETER ACCEPTANCE FORM 32

3.9 DISPLAY A MESSAGE AT THE BOTTOM LINE IN A FORM. 37

3.10 SPECIFY ADDITIONAL ACTIONS FOR A FUNCTION KEY AT ANY FIELD 38

4 WORKFLOW SCRIPTING FUNCTIONS 40

4.1 EXECUTING A MENU OPTION 40

4.2 CALLING INPUT ACCEPTANCE FORM 41

5 INDEX OF USER HOOKS 43

1 SCRIPT ENGINE BASED CUSTOMIZATION

1.1 WHAT IS 'SCRIPT ENGINE BASED CUSTOMISATION'?

Finacle™ is a Core Banking package which supports a number of deployment topologies (Fully Distributed to Fully Centralized) and Implementation scenarios (Single currency, Multiple currency, Various Retail and Trade finance banking products and Interest methods etc.). In order to support these multitude of features, it is essential that Finacle™ is open to 'Customization' by the Bank so that it can implement the features that it requires and in the way that it desires. Also, the Bank will need to interface some of their 'peripheral applications' (like Treasury, Consumer Finance etc.) which are not supported directly by Finacle™, to Finacle™ in various ways. Another area of 'Customizability' is in the interface to various special purpose equipment like ATM switches, VRUs, MICR encoders etc. which are supplied by many vendors with some differences in actual implementation.

So far, Finacle™ had addressed this issue by defining 'parameters' which was set up by the Bank depending upon its requirements. However, this has the limitation of requiring that the developers of Finacle™ know all the possible business logic before hand and then implement them using parameters. However, a far more powerful approach is to allow the Bank to 'take control ' at certain 'events', apply their own logic on the data associated with that event and be able to either influence the outcome of that event or communicate the occurrence of the event to other 'Custom' applications. The Finacle™ 'Script Engine based customization' allows the bank to do just this.

The Script Engine Based Customization (**SEBC**) consists of 3 parts:

1) The Script Engine – This is an 'interpretive language' processor which allows a wide range of key programming constructs, while allowing the Bank to 'custom develop' additional functions that can be called from the 'scripts'.
2) The Finacle™ functions – Certain standard functions and certain 'module specific' functions have been developed by the Finacle™ developers which can be called from the 'scripts'. This document contains more details about this.
3) The Finacle™ 'script events' – Right across Finacle™, there are several events that have been 'opened up' by defining script events. What it means is that for a 'script-enabled event', the Finacle™ application, checks for the existence of a defined script and if present, transfers control to the Script Engine to execute that script after populating all the data required by that event in 'input fields'. It is the script's responsibility to apply whatever logic it wants and provide the necessary 'output fields' at the end of the script for Finacle™ to continue its processing. Please refer to Scripting Events document for more details.

1.2 WHAT IS A FINACLE™ USER HOOK?

The document on scripting Syntax describes the Finacle™ Script Engine capabilities and the Finacle™ scripting language syntax. This document describes all such functions that are provided in Finacle™, which can be used while writing a Finacle™ script. Some of these functions are general and can be used in the context of any of the Script Events described in scripting Events document or in any Workflow script. However, certain functions are specific either to the Workflow context or to a specific event. In such a case it will be documented as such along with its description below.

The general format for describing each function is as follows:

Brief description of the function and context in which it can be used.

Function syntax – (Also gives change log between Finacle™ versions)

Detailed functionality –

Description of the Input fields and Output fields –

Example of the usage of the function –

1.3 REPOSITORIES AND CLASSES

Please refer to the document on scripting syntax for a general discussions on Repositories and classes.

All Finacle™ user hooks (except for Workflow related functions and MTT related functions) interface with the scripts using a standard repository called "BANCS". There are 2 classes that have been predefined in this repository, INPARAM and OUTPARAM each of which hold 'String' type of fields. INPARAM is the class that is used to populate specified fields (depending upon the function) in the script so that the function can access those as parameters. All the fields (depending upon the function) that are output by the function are populated in the OUTPARAM class.

In addition to INPARAM class, the input to the functions can also be provided as arguments to the function. Please note that the INPARAM class is cleared out once the user hook is executed and needs to be repopulated by the script when calling another user hook or the same user hook in a loop.

1.4 RETURN STATUS OF THE FUNCTION.

All functions return a value of either SUCCESS (0) or FAILURE (1).

2 GENERAL SCRIPTING FUNCTIONS

These scripting user hooks can be used in the context of any script.

2.1 GET THE ACCOUNT DETAILS INTO BANCS REPOSITORY

This function returns information about a given account number in the FINACLE™ database. It can be used in the context of any Script Event or Workflow.

SYNTAX

sv_a = urhk_getAcctDetailsInRepository(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("SBGEN1").

FUNCTIONALITY

This function checks whether the given account number exists in the datacenter database. If the account exists (irrespective of the state of the account), then some attributes of the account, as detailed below, are returned in repository fields.

INPUT

The input to this function is an account number (FORACID).

OUTPUT

Return value is 0, if account found, else 1.

This function will return the following Account details into OUTPARAM class of BANCS repository if the account exists in the datacenter database.

Field Name in GAM table	Field name in BANCS repository
sol_id	BANCS.OUTPARAM.acctSolId
Bacid	BANCS.OUTPARAM.acctPlaceHolder
acct_name	BANCS.OUTPARAM.acctName
cust_id	BANCS.OUTPARAM.custId
emp_id	BANCS.OUTPARAM.empId
gl_sub_head_code	BANCS.OUTPARAM.glSubHeadCode
acct_ownership	BANCS.OUTPARAM.acctOwnerShip (C'ustomer, E'mployee, O'ffice account)
schm_code	BANCS.OUTPARAM.schmCode
schm_type	BANCS.OUTPARAM.schmType
acct_opn_date	BANCS.OUTPARAM.acctOpenDate
acct_cls_date	BANCS.OUTPARAM.acctCloseDate
acct_cls_flg	BANCS.OUTPARAM.acctCloseflg (Y/N)
mode_of_oper_code	BANCS.OUTPARAM.modeOprnCode
acct_locn_code	BANCS.OUTPARAM.acctLocnCode
acct_crncy_code	BANCS.OUTPARAM.acctCrncyCode
system_only_acct_flg	BANCS. OUTPARAM.systemOnlyAcctFlg (Y/N)

Field Name in GAM table	Field name in BANCS repository
Foracid	BANCS.OUTPARAM.AcctId
dr_bal_lim	BANCS.OUTPARAM.debitBalanceLimit
frez_code	BANCS.OUTPARAM. freezeCode
frez_reason_code	BANCS.OUTPARAM. freezeReasonCode
clr_bal_amt	BANCS.OUTPARAM. clearBalance
un_clr_bal_amt	BANCS.OUTPARAM. unclearBalance
ledg_num	BANCS.OUTPARAM. ledgerNumber
drwng_power	BANCS.OUTPARAM. drawingPower
sanct_lim	BANCS.OUTPARAM. sanctionLimit
adhoc_lim	BANCS.OUTPARAM. adhocLimit
emer_advn	BANCS.OUTPARAM. emergencyAdvance
dacc_lim	BANCS.OUTPARAM. DACCLimit
system_reserved_amt	BANCS.OUTPARAM. systemReservedAmt
single_tran_lim	BANCS.OUTPARAM. singleTranLimit
clean_adhoc_lim	BANCS.OUTPARAM. cleanAdhocLimit
clean_emer_advn	BANCS.OUTPARAM. cleanEmergencyAdvance
clean_single_tran_lim	BANCS.OUTPARAM. cleanSingleTranLimit
system_gen_lim	BANCS.OUTPARAM. systemGeneratedLimit
chq_alwd_flg	BANCS.OUTPARAM. chequeAllowed
cash_excp_amt_lim	BANCS.OUTPARAM. cashExceptionAmtLimit
clg_excp_amt_lim	BANCS.OUTPARAM. clearingExceptionAmtLimit
xfer_excp_amt_lim	BANCS.OUTPARAM. transferExceptionAmtLimit
cash_cr_excp_amt_lim	BANCS.OUTPARAM. cashCrExceptionAmtLimit
clg_cr_excp_amt_lim	BANCS.OUTPARAM. clearingCrExceptionAmtLimit
xfer_cr_excp_amt_lim	BANCS.OUTPARAM. transferCrExceptionAmtLimit
cash_abnrml_amt_lim	BANCS.OUTPARAM. cashAbnormalAmtLimit
clg_abnrml_amt_lim	BANCS.OUTPARAM. clearingAbnormalAmtLimit
xfer_abnrml_amt_lim	BANCS.OUTPARAM. TransferAbnormalAmtLimit
acrd_cr_amt	BANCS.OUTPARAM. accruedCreditAmt
pb_ps_code	BANCS.OUTPARAM. passbookOrPasssheet
serv_chrg_coll_flg	BANCS.OUTPARAM. serviceChrgCollectedFlg
int_paid_flg	BANCS.OUTPARAM. interestPaidFlag
int_coll_flg	BANCS.OUTPARAM. interestCollectedFlag
limit_prefix	BANCS.OUTPARAM. limitPrefix
limit_suffix	BANCS.OUTPARAM. limitSuffix
drwng_power_ind	BANCS.OUTPARAM. drawingPowerIndicator
drwng_power_pcnt	BANCS.OUTPARAM. drawingPowerPercentage
notional_rate	BANCS.OUTPARAM. notionalRate
notional_rate_code	BANCS.OUTPARAM. notionalRateCode
fx_clr_bal_amt	BANCS.OUTPARAM. FXClearBalance
crncy_code	BANCS.OUTPARAM. FCNRCrncyCode
wtax_flg	BANCS.OUTPARAM. TaxFlg
wtax_amount_scope_flg	BANCS.OUTPARAM. TaxAmtScopeFlg
lien_amt	BANCS.OUTPARAM. lienAmt
acct_mgr_user_id	BANCS.OUTPARAM. accountManagerUserId
schm_type	BANCS.OUTPARAM. schemeType
Partitioned_flg	BANCS.OUTPARAM. partitionedFlg
Partitioned_type	BANCS.OUTPARAM. partitionedType
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM AvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FxAvailableAmt

Field Name in GAM table	Field name in BANCS repository
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FFDAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FxFFDAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM EffAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FxEffAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FullAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FxFullAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FullEffAvailableAmt
Not in GAM table (Sum of different limits)	BANCS.OUTPARAM FxFullEffAvailableAmt

EXAMPLE:

Let us say that the bank does not want the account balance of 'Office accounts' to be displayed in the lower block of the 'Transaction Maintenance' screen when doing a transaction (*eg: this information should not be available to the Teller*). Then the CheckAndDispAcctBal.scr script can be coded as follows:

```
<--start

sv_a = BANCS.INPUT.AcctId

# --------------------------------------------------------------------------------
-------
# - sv_a (account id) is the input to user hook getAcctDetailsInRepository
# - Call user hook getAcctDetailsInRepository to put the Account details into BANCS
repository.
# --------------------------------------------------------------------------------
-------

sv_b = urhk_getAcctDetailsInRepository(sv_a)

# if getAcctDetailsInRepository function returns failure exit out of script.
# This condition should not occur since BANCS.INPUT.AcctId would
# already have been validated to exists when this script is invoked.

if( sv_b == 1 ) then
    exitscript
endif

sv_c="O"
# acctOwnership flag is "O" for office accounts

# ---------------------------------------------------------------------------
# If the acctOwnerShip flag of ACCOUNT class is equal to "O"
# then don't display dispblk2.acct_bal field.
# ---------------------------------------------------------------------------

if (BANCS.OUTPARAM.acctOwnerShip == sv_c) then
   sv_z = "dispblk2.acct_bal|H"
           sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
else
   sv_z = "dispblk2.acct_bal|U"
sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
       endif
 exitscript
 end-->
```

Note: Refer to urhk_TBAF_ChangeFieldAttrib given separately for is usage

```
# ---------------------------------------------------------------------------
# If the tod amt changes at the time of posting (InstantTodAmt
# and TodLvlIntFlg are inputs to the script),
# raise exception if the difference is more than 100
# ---------------------------------------------------------------------------
sv_e = cdouble(BANCS.INPUT.InstantTodAmt)
    if (sv_e != 0.00 ) then
    sv_a = BANCS.INPUT.AcctNum
        sv_b = urhk_getAcctDetailsInRepository(sv_a)
        if (sv_b == 1) then
            BANCS.OUTPUT.OutString = "Account number is invalid"
            BANCS.OUTPUT.successOrFailure="F"
            exitscript
        endif
        sv_c = cdouble(BANCS.OUTPARAM.AvailableAmt)
        sv_d = cdouble(BANCS.OUTPARAM.FullEffAvailableAmt)
        if ( sv_d < 0 ) then
            sv_d = 0
        endif
        sv_f = cdouble(BANCS.OUTPARAM.FullAvailableAmt)
        sv_g = BANCS.INPUT.TodLvlIntFlg
        sv_h = cdouble(BANCS.INPUT.TranAmt)
        sv_i = ( sv_h - sv_d )
        sv_j = ( sv_e - 100 )
        sv_k = ( sv_e + 100 )
    if ( ( sv_i < sv_j ) OR ( sv_i > sv_k ) ) then
            BANCS.OUTPUT.successOrFailure = "F"
```

```
          BANCS.OUTPUT.ErrorDesc = "Error: TOD Amt. is different from that of
specified value"
endif
   endif
```

2.2 GET THE SCHEME DETAILS INTO BANCS REPOSITORY

This function returns scheme details information in the FINACLE™ database. It can be used in the context of any Script Event or Workflow.

SYNTAX

sv_a = urhk_getSchemeDetailsInRepository (Variable)

The variable can be a scratch pad variable (sv_b) or a string("SBGEN").

FUNCTIONALITY

This function will return some attributes of the given scheme in the repository fields as detailed below if the scheme is found. If the scheme does not exist, then a fatal error occurs. Hence the scheme code passed must be a valid scheme code.

INPUT

The input to this function is a valid (existing) SCHEME CODE.

OUTPUT

Return value is always 0. If scheme not found then Fatal Error will result.

This function will return the following Scheme details into OUTPARAM class of BANCS repository if the scheme exists in the datacenter database.

Field Name in GSP table	Field name in BANCS repository
schm_desc	BANCS.OUTPARAM. SchemeDescription
int_paid_bacid	BANCS.OUTPARAM. InterestPaidPlaceHolder
int_coll_bacid	BANCS.OUTPARAM. InterestCollPlaceHolder
serv_chrg_bacid	BANCS.OUTPARAM. ServiceChargePlaceHolder
chq_alwd_flg	BANCS.OUTPARAM. SchmChequeAllowed
schm_supr_user_id	BANCS.OUTPARAM. SchemeSupervisorUserId
new_acct_duration	BANCS.OUTPARAM. NewAccountDuration
dorm_acct_criteria	BANCS.OUTPARAM. DormantAccountCriteria
min_posting_workclass	BANCS.OUTPARAM. MinPostingWorkClass
schm_type	BANCS.OUTPARAM. SchemeType
int_paid_flg	BANCS.OUTPARAM. InterestPaidFlg
int_coll_flg	BANCS.OUTPARAM. InterestCollectedFlg
staff_schm_flg	BANCS.OUTPARAM. StaffSchemeFlg
nre_schm_flg	BANCS.OUTPARAM. NRE_SchemeFlg
fd_cr_bacid	BANCS.OUTPARAM. FD_CreditPlaceHolder
fd_dr_bacid	BANCS.OUTPARAM. FD_DebitPlaceHolder

stp_cr_dr_ind	BANCS.OUTPARAM. StopCrDrInd
fcnr_flg	BANCS.OUTPARAM. IsFCNR
int_pandl_bacid	BANCS.OUTPARAM. Int_P_and_L_PlaceHolder
Parking_bacid	BANCS.OUTPARAM. ParkingPlaceHolder
Advance_int_bacid	BANCS.OUTPARAM. AdvanceIntPlaceHolder
tds_parking_bacid	BANCS.OUTPARAM. TDS_ParkingPlaceHolder
penal_pandl_bacid	BANCS.OUTPARAM. Penal_P_and_L_PlaceHolder
dflt_clg_tran_code	BANCS.OUTPARAM. DefaultClearingTranCode
dflt_inst_type	BANCS.OUTPARAM. DefaultInstrumentType
int_pandl_bacid_cr	BANCS.OUTPARAM. Int_P_and_L_CrPlaceHolder
int_pandl_bacid_dr	BANCS.OUTPARAM. Int_P_and_L_DrPlaceHolder

EXAMPLE:

Let us say that banker wants to change a certain scheme which is fcnr to change later to non fcnr so the fcnr flg field should be unprotected and if it is non fcnr the field should be protected.

```
<--start

sv_a = BANCS.INPUT. SchmCode

# ---------------------------------------------------------------------------
# sv_a (SchmCode) is the input to user hook getSchemeDetailsInRepository
# Call user hook getSchemeDetailsInRepository to put the Scheme details
# into BANCS repository.
# ---------------------------------------------------------------------------

sv_b = urhk_getSchemeDetailsInRepository (sv_a)

# FCNR flag is "Y" for fcnr type of schemes.

# ---------------------------------------------------------------------------
# If the FCNR flag of scheme code class is equal to "Y"
# then protect dispblk1.fcnr_flg field.
# ---------------------------------------------------------------------------

if (BANCS.OUTPARAM. IsFCNR == "Y") then
    sv_z = "dispblk1.fcnr_flg|U"
    sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
else
    sv_z = "dispblk1. fcnr_flg|P"
    sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
endif
exitscript

end-->
```

☞ *Note:* *Refer to urhk_TBAF_ChangeFieldAttrib given separately for its usage.*

2.3 BACID TO ACCT ID AND NAME

This function returns the Account Id and name of the account that is identified via a BACID, Currency code and Sol Id, if such an account exists. This is a subset of getAcctDetailsInRepository and may be removed in future versions. It is usually used as a part of a script where the Bacid, Currency and Sol Id are known (or are obtained) before this user hook is called.

SYNTAX

The Syntax for calling the function is

sv_r = urhk_B2k_BacidToAcctId(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("PURCH|INR|123456").

FUNCTIONALITY

Get the account id for passed Bacid, Currency and Sol Id.

INPUT

Input String contains three parts separated by a '|'

1. Bacid
2. Currency code
3. Sol id

Example – "PURCH|INR|123456"

OUTPUT

Return value will be 1 if no account is found, else it will be 0 and fields AcctName and AcctId of OUTPARAM class of BANCS repository will be populated for the deduced account.

Field Name in GAM table	Field name in BANCS repository
sol_id	BANCS.OUTPARAM.acctSolId
Bacid	BANCS.OUTPARAM.Bacid
CrncyCode	BANCS.OUTPARAM.CrncyCode
AcctName	BANCS.OUTPARAM.AcctName
AcctId	BANCS.OUTPARAM.AcctId

EXAMPLE:

This user hook is a subset of getAcctDetailsInRepository user hook. It is used to manipulate only acctName and acctId (Foracid). In the example below, the bank has named all system cash accounts as "SYSTEM CASH" and wants these accounts to be skipped in the script.

```
<--start

….
….
# The above statements set the context of Bacid, Currency Code and Foracid
# in variables sv_b, sv_c and sv_f respectively for this example.

sv_a = sv_b + “|” + sv_c + “|” sv_f

# ---------------------------------------------------------------------------------
-------
# - sv_a (see Input section above) is the input to user hook
#  B2k_BacidToAcctId
# - Call user hook B2k_BacidToAcctId to put the account name and foracid
#   in the repository
# ---------------------------------------------------------------------------------
-------

sv_r = urhk_B2k_BacidToAcctId (sv_a)

# if B2k_BacidToAcctId function returns failure exit out of script.

if( sv_r == 1 ) then
    exitscript
endif

sv_s ="SYSTEM CASH"
# All cash accounts in various currencies have this name
# --------------------------------------------------------------------------
# acctName is compared
# If system cash, the script skips the account
# --------------------------------------------------------------------------

if (BANCS.OUTPARAM.acctName == sv_s) then
   GOTO lblSkipSystemCash
else
   …..
…..
# do the rest of the processing as desired
endif
end-->
```

2.4 CONVERT AN AMOUNT FROM ONE CURRENCY TO ANOTHER

Given an input amount, ‘from currency’, ‘to currency’ and either a rate code or conversion rate, this function gives the equivalent amount in the ‘to currency’.

SYNTAX

The Syntax for calling the function is

sv_r = urhk_B2k_ConvertAmount("")

FUNCTIONALITY

This function applies the same logic as Finacle™ in determining equivalent amounts in different currencies.

INPUT

All the input variables should be populated in the INPARAM class of the BANCS repository. The variables are (all STRINGS):

InputAmount	The amount that needs to be converted
FromCurrency	The currency that the input amount is in
ToCurrency	The currency in which the equivalent amount is needed
RateCode	The rate code to be used during conversion. Will be used only if Rate is "0"
Rate	The conversion rate to be used for the equivalence Calculation, if not "0".

OUTPUT

Function returns 0 or 1. If 1, then there was failure in conversion. If 0, all output variables will be populated in the OUTPARAM class of the BANCS repository. The variables are (all STRINGS):

OutputAmount -	The equivalent amount in 'to currency'
Rate -	The conversion rate actually used (in case INPARM.RateCode is used, INPARM.Rate is zero).

EXAMPLE:

```
<--start
sv_a = "USD"
#From Currency
sv_b = "GBP"
#To Currency
sv_c = "TCB"
# Want to get the rate for Travellers Cheque buying
….
….
# sv_p contains the amount to be converted

BANCS.INPARM.InputAmount = sv_p
BANCS.INPARM.FromCurrency = sv_a
BANCS.INPARM.ToCurrency = sv_b
Sv_r = urhk_B2k_ConvertAmount("")
if (sv_r ==1) then
   # Error processing
   …
   …
endif
sv_q = BANCS.OUTPARAM.OutputAmount
# sv_q contains the converted amount
sv_r = BANCS.OUTPARAM.Rate
# sv_r contains the TC buying rate
end-->
```

2.5 ROUNDOFF AN AMOUNT

Given an input amount, 'round off to' amount, and roundoff flag, this function gives the roundedoff amount.

SYNTAX

The Syntax for calling the function is

sv_r = urhk_B2k_RoundOff ("")

FUNCTIONALITY

This function applies the same logic as Finacle™ in roundingoff the amount.

INPUT

All the input variables should be populated in the INPARAM class of the BANCS repository. The variables are(all STRINGS) :

InputAmount - The amount that needs to be converted

RoundOffAmt - The amount to be rounded off to

Example : If the amount is in Rupees …

100 – 100 Rupees

.01 – 1 Paise

RoundOffFlag - Valid values are:

H - roundoff to the next higher amount

L - roundoff to the previous lower amount

N - roundoff to the nearest amount

OUTPUT

Function returns 0 or 1. If 1, then there was failure in roundingoff. If 0,

All output variables will be populated in the OUTPARAM class of the BANCS repository. The variables are(all STRINGS) :

OutputAmount - The roundedoff amount

EXAMPLE:

```
<--start
trace on
sv_a = 10986792.2358
#Input Amount
sv_b = 100
# Round off to 100 Rupees
sv_c = "L"
# Lower amount

BANCS.INPARAM.InputAmount = sv_a
BANCS.INPARAM.RoundOffAmt = sv_b
BANCS.INPARAM.RoundOffFlag = sv_c
sv_r = urhk_B2k_RoundOff ("")
if (sv_r == 1) then
# Error processing
endif
sv_q = BANCS.OUTPARAM.OutputAmount
# sv_q contains the rounded amount. This should be 10986700.0000
end-->
```

2.6 VALIDATE DATE

This user hook will validate if the string passed to it is a valid date. Only the date component is validated not the time component. However when passing the string either a string with only date or string with date and time may be passed. The user hook truncates the time component and validates the date.

SYNTAX:

sv_a = urhk_ B2k_valDate(Variable)

sv_a: scratch pad variable to store the return value

Variable: Scratch pad variable containing string to be validate.

FUNCTIONALITY:

This user hook is used to validate if a given string is a valid date. The first 10 characters of the string are taken and validated to check if it is a valid date. The time component is not validated.

INPUT:

String to be validated whether it is a valid date or not

OUTPUT

Return value will be success ('0') if string is a valid date else failure.

EXAMPLE:

```
sv_r=urhk_B2k_valDate(PASTDUE.PDC.dateofreckoning)
 if (sv_r != 0) then
           sv_a="Invalid Date given for field Date Of Reckon "
 else
            sv_a=""
 endif
```

2.7 VALIDATE REFERENCE CODE:: VRP1.6.38.01

This user hook will validate if the string passed to it is a valid reference code. The input to the user hook is reference code type and reference code. The user hook validates that the reference code is valid and of type reference code type as passed to it.

SYNTAX:

sv_a = urhk_ B2k_valRefCode(Variable)

sv_a: scratch pad variable to store the return value

Variable: Scratch pad variable containing string consisting of reference code type and reference code delimited by '|'.

FUNCTIONALITY:

This user hook will validate if the string passed to it is a valid reference code. The input to the user hook is reference code type and reference code. The user hook validates that the reference code is valid and of type reference code type as passed to it.

INPUT:

String consisting of reference code type and reference code to be validated delimited by '|'.

OUTPUT

Return value will be success ('0') if string is a valid reference code of the given reference code type else failure.

EXAMPLE:

```
sv_z="12|"+PASTDUE.PDL.authpermwaiver
sv_r=urhk_B2k_valRefCode(sv_z)
if (sv_r != 0) then
      sv_a="Auth. Perm. Waiver"
else
      sv_a=""
endif
```

2.8 SELECTION OF FIELDS FROM DATABASE

SYNTAX

sv_r = urhk_dbSelect(variable)

The variable can be a scratch pad variable (sv_a) or a string ("user_id | select user_id from upr where session_id = 'xxxx'").

Here note that everything before '|' (pipe symbol) is taken as the title value where as post pipe symbol is taken as query.

FUNCTIONALITY

This user hook allows script writer to select data from database tables. The user specifies the title of each selected field separated by comma before pipe symbol and specifies the complete query after that. It stores the output error code of the query in BANCS.OUTPARAM.DB_ERRCODE and message in BANCS.OUTPARAM.DB_ERRMSG.

INPUT

Comma separated Field titles followed by the Query, separated by pipe symbol. The field symbols should not have spaces.

OUTPUT

The output fields are stored in BANCS.OUTPARAM.<title field>. An error free condition is represented by 0 value of BANC.OUTPARAM.DB_ERRCODE.

LIMITATION

- In case the query returns more than single row, it gives a FATAL error hence one need to be very meticulous during query writing.
- There is a size limitation of the query string size. In case where query size is very large make use of scratch pad variables.

EXAMPLE

```
sv_a = getenv("B2K_SESSION_ID")
sv_b = "user_id | select user_id from lgi where session_id = '" + sv_a + "'"
sv_r = usrhk_dbSelect(sv_b)
```

The above will fetch the value of **user_id** in **BANC.OUTPARAM.user_id** field.

2.9 USING A SQL STATEMENT IN A SCRIPT

SYNTAX

sv_r = urhk_dbSQL(variable)

The variable can be a scratch pad variable (sv_a) or a string ("update Upr Set User_logged_on_flg = 'N' where session_id = 'xxxx'").

FUNCTIONALITY

This user hook allows script writer to perform DML Statements on the database tables.

INPUT

The SQL Statement

OUTPUT

The output fields are stored in BANCS.OUTPARAM.<title field>. An error free condition is represented by 0 value of BANC.OUTPARAM.DB_ERRCODE.

This user hook must be used with utmost caution since none of the Finacle™ Application logic will apply here.

2.10 ACCOUNT BALANCE AS ON A GIVEN DATE:

This user hook will return end of day account balance as on a given date from FINACLE™. This user hook can be called from any script event.

SYNTAX:

sv_a = urhk_getAcctBalanceAsOnDate("")

sv_a: scratch pad variable to store the return value.

FUNCTIONALITY

- Validates account number and as on date. Will return failure if account number is invalid or if date format is not correct.
- If inputs are valid, populates end of day account balance into FINACLE™ repository.

INPUT

- Valid account number
- As on Date

The above inputs are to be populated into the following repositories.

Valid Account Number	BANCS.INPARAM.acctId
As on Date	BANCS.INPARAM.asOnDate (in DD-MM-YYYY format)

OUTPUT

If the input values are invalid return value will be '1'.

End of day account balance based on transaction date will be populated into the repository specified below.

Account balance	BANCS.OUTPARAM.acctBal

EXAMPLE

To obtain account balance as on BOD date for an account, the following may be implemented.

```
<--start

# Populate BOD date and into as on date and account number

    BANCS.INPARAM.acctId = BANCS.INPUT.acctId
    BANCS.INPARAM.asOnDate = BANCS.STDIN.BODDate

    sv_a = urhk_getAcctBalanceAsOnDate("")

    if ( sv_a == 1 ) then
         exitscript
    endif

  sv_i = cdouble(BANCS.INPARAM.balance)
    end-->
```

2.11 VALIDATE CRNCY CODE.

SYNTAX:

sv_a=urhk_B2k_valCrncy(crncyCode)

FUNCTIONALITY:

This user hook will take the crncyCode as the input and validates it.

INPUT :

crncy code as shown in the syntax.

OUTPUT:

This user hook return 0 if the input is valid crncy code , otherwise it returns a non zero value.

3 ONLINE SCRIPTING FUNCTIONS

Can be used in the context of any scripts invoked by Online programs only

3.1 SET THE DEFAULT ATTRIBUTE OF A FIELD IN THE FORM

This function will provide a facility to change the attributes of a field in the form. It can be used in the context of WorkFlow or in the script executed during the Pre Form Load event (xxxxxxxxPreLoad.scr) only (refer to Scripting Events document for more details about Preload scripts)

For setting the attribute of a field in any other online event refer to urhk_TBAF_ChangeFieldAttrib in this document.

SYNTAX

sv_a = urhk_TBAF_SetAttrib(Variable)

Variable needs to be formatted as "<formname>.<blockname>.<fieldname>|Attribute"

Attribute	M	Mandatory
	P	Protect (Cursor and data entry not allowed)
	H	Hide
	E	called as **Entry Allowed** Cursor entry allowed, Data Entry not allowed

The variable can be a scratch pad variable (sv_b) or a string ("baff0009.datablk2.cust_comu_addr1|M")

☞ *Note: Use dispfields utility for finding this information. This utility is documented in Workflow manual.*

FUNCTIONALITY

This function can be used to change the attribute of any field in a form that is currently being loaded or about to be called for loading (in the case of WorkFlow). By default, all fields in a form that is loaded carry the attribute assigned to it by Infosys. However, a Bank may wish to customise the attributes of certain fields in certain forms which is different from the Infosys setting. For example, a Bank may decide that the 'Communication address line 1' field in the Customer master maintenance screen is to be made mandatory (default setting is non-mandatory). Then this function can be used in the form load event of the form baff0009 to set the attribute of the field 'cust_comu_addr1' as mandatory.

Please note that when a field has a default setting of 'Mandatory', setting 'Protect' or 'Entry allowed' attributes to that field is not allowed.

Also, once an attribute for a field has been set, setting the attribute of that field again (even with a different attribute) before the form is unloaded, will not have any effect.

INPUT

Input String contains two parts separated by a '|'

1. Field name

2. Attribute

EXAMPLE

- "baff0009.datablk2.cust_comu_addr1|M"

Field name should be given as FormName.BlockName.FieldName.

In the above example the attribute is given as M to cust_comu_addr1 field which makes it mandatory to enter a value in the field.

Valid values for Attribute are :

P – Protect – The field is made non-enterable and tab will skip past this field

M – Mandatory – The field is made mandatory and something needs to be entered in this field before proceeding

H – Hide – The field value is not displayed

E – Entry allowed – The cursor can be positioned in the field but no data can be entered. Useful if function keys need to be used in a protected field (to scroll the field or to explode for example)

OUTPUT

Always returns 0 as return value

The form will display the new attribute after it is loaded if the specified field exists in the form and the attribute does not conflict with the default setting. Otherwise, nothing happens.

EXAMPLE:

The logic discussed in the 'functionality section' can be implemented by coding baff0009PreLoad.scr as follows:

```
<--start
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk2.cust_comu_addr1|M")

exitscript
end-->
```

3.2 SET THE DEFAULT VALUE FOR A FIELD IN THE FORM DURING FORM LOAD.

This function will populate the given value for a field in a form that is currently being loaded. It can be used in the context of WorkFlow or in the script executed during the Form Load event (xxxxxxxxPreLoad.scr) only (refer to Scripting Events document for more details about Preload scripts)

For changing the value of a field in any other online event refer to urhk_TBAF_ChangeFieldValue in this document.

SYNTAX

sv_a = urhk_TBAF_SetValue (Variable)

As discussed for TBAF_SetAttrib above.

The variable can be a scratch pad variable (sv_b) or a string ("tdfe3201.datablk1.ledg_num|1")

FUNCTIONALITY

This function can be used to set the default value of any field in a form that is currently being loaded or about to be called for loading(in the case of WorkFlow). By default, many fields in a form that is loaded carry a NULL value assigned to it by Infosys. However, a Bank may wish to assign a default values of certain fields in certain forms. For example, the Customer master maintenance function assigns no value to the 'Customer Non Resident', 'Customer staff' and 'Customer minor' fields. A Bank may decide that a default value of 'N' needs to be assigned to these 3 fields. Then this function can be used in the form load event of the form baff0009 to set the default value as 'N' for these three fields.

It is also used in conjunction with **urhk_TBAF_GetValue** to get back values from a form after 'Commit'. In order to do that, this function should be used to specify all the form fields whose values (as they exist after commit processing) is to be accessed, and then after commit processing use **urhk_TBAF_GetValue** to access their values.

Please note that once a value for a field has been set, setting the value of that field again (even with a different value) before the form is unloaded, will not have any effect.

INPUT

Input String contains three parts separated by a '|'

1. Field name

2. Value - Max. size is 255 characters

3. Repository [Optional]

EXAMPLE –

"tdfe3201.datablk1.ledg_num|1"

Field name should be given as FormName.BlockName.FieldName.

Value is the data to be populated in the field. It can have a NULL value to set the value of a field to NULL. Ex. "tdfe3201.datablk1.ledg_num|"

Repository name is meant for advanced use(in the context of Workflow. See discussion on TEMP repositories under **urhk_WFS_TransferUser** and

urhk_WFS_CreateWFItem sections) and typically should not be specified as shown in the example above. In this case the repository INTBAF is assumed. However, when we intend to specify the list of fieldnames whose value we need to access after 'commit' processing, this repository should be specified as OUTTBAF.

OUTPUT

Always returns 0 as return value.

When the repository is not specified then the form will display the new value after it is loaded if the specified field exists in the form, otherwise, nothing happens.

When the repository OUTTBAF is specified, nothing happens when the form is loaded, but you can then use **urhk_TBAF_GetValue** in post-commit events to access the value of the field.

EXAMPLE:

The logic discussed in the 'functionality section' can be implemented by coding baff0009PreLoad.scr as follows:

```
<--start
sv_a = urhk_TBAF_SetValue("baff0009.datablk1. cust_nre_flg|N")
sv_a = urhk_TBAF_SetValue("baff0009.datablk1. cust_emp_flg|N")
sv_a = urhk_TBAF_SetValue("baff0009.datablk1. cust_minor_flg|N")


exitscript
end-->

The logic for getting the assigned Customer Id (after commit) can be implemented by
coding a Workflow script which calls CUMM menu Option, as follows:

<--start
sv_a = urhk_TBAF_SetValue("baff0009.datablk6.cust_id| |OUTTBAF")
sv_b = urhk_WFS_CallMenuOption("CUMM|0|3")
if(sv_b == 1) then
   exitscript
endif
sv_d = CLASSEXISTS("OUTTBAF","baff0009")
if (sv_d == 1) then
    sv_a = urhk_TBAF_GetValue("baff0009.datablk6.cust_id ")
    STDWFS.GLBDATA.cust_id = B2KTEMP.TEMPSTD.TBAFRESULT
    DELETECLASS("OUTTBAF","baff0009")
endif


exitscript
end-->
```

3.3 CHANGE THE FIELD VALUE IN THE FORM.

This function will change the value of a field in the current form to the specified value.

SYNTAX

sv_a = urhk_TBAF_ChangeFieldValue(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("optionblk.option_code|P")

FUNCTIONALITY

This user hook will change the value of a field in the form. This user hook should not be used in Pre Form load or Post Form Load scripts. In the Pre Form load scripts use the function **urhk_TBAF_SetValue.**

INPUT

Input string has two parts separated by a '|'

1. Field name.
2. Value - Max. size is 255 characters

Example - "optionblk.option_code|P"

Field name should be given as BlockName.FieldName.

Value is the data to be populated in the field. It can have a NULL value to change the value of a field to NULL. Ex. "datablk1.ledg_num|"

OUTPUT

Always returns 0 as return value.

The form will display the new value immediately, if the field is found in the current form, else nothing happens.

EXAMPLE:

Please refer to Example section of **urhk_TBAF_SetReplayKey.**

In **tm1.scr** this function is used to populate the option code as 'P'to request for posting the transaction.

3.4 INQUIRE THE FIELD VALUE IN THE FORM.

This function will get the value of a field in the current form.

SYNTAX

sv_a = urhk_TBAF_InquireFieldValue(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("optionblk.option_code")

FUNCTIONALITY

This function will get the value of a field in the form in the repository field B2KTEMP.TEMPSTD.TBAFRESULT. This function should not be used during Pre Form load or Post Form Load events.

INPUT

Input string has one part

Field Name

EXAMPLE –

"optionblk.option_code"

Field name should be given as BlockName.FieldName.

OUTPUT

Always returns 0 as return value

Required value will be available in B2KTEMP.TEMPSTD.TBAFRESULT if the field is found in the current form, else B2KTEMP.TEMPSTD.TBAFRESULT will contain "".

Example:

In this example, a sample script has been written which can be used to invoke all the mandatory forms to be visited while opening a deposit account automatically.

First we will code the PreLoad script of the deposit form to execute an 'accept-key' when the option block is visited.

Then we will code the pre-replay script of the above key to populate the appropriate option code into the option field.

Then we will code the post-replay script of the above key to reset the replay key so that when the option block is visited again the next option is chosen.

```
Script Name :- tdfe3201PreLoad.scr

<--start
sv_a = urhk_TBAF_SetReplayKey("optnblk.key-f2|tdoptpre.scr|0|tdoptpost.scr|0")
exitscript
end-->


Script Name :- tdoptpre.scr

<--start

sv_b = cint(INTBAF.INTBAFC.TbafEventStep)

if (sv_b == 0) then
    GOTO STEP0
endif

if (sv_b == 1) then
    GOTO STEP1
endif

if (sv_b == 2) then
    GOTO STEP2
endif

if (sv_b == 3) then
    GOTO STEP3
endif

exitscript

# Populate option as G to visit general details.

STEP0:
sv_a = urhk_TBAF_ChangeFieldValue("optnblk.acct_opn_option|G")
exitscript

# Populate option as N to visit Nominee details

STEP1:
                  sv_a = urhk_TBAF_ChangeFieldValue("optnblk.acct_opn_option|N")
exitscript

# Populate option as F to visit Flow details.

STEP2:
sv_a = urhk_TBAF_ChangeFieldValue("optnblk.acct_opn_option|F")
exitscript

# Populate option as V to visit Advance details.

STEP3:
sv_a = urhk_TBAF_ChangeFieldValue("optnblk.acct_opn_option|V")
exitscript

end-->

Script Name :- tdoptpost.scr

<--start

sv_b = cint(INTBAF.INTBAFC.TbafEventStep)

if (sv_b == 0) then
    GOTO STEP0
endif

if (sv_b == 1) then
    GOTO STEP1
endif

if (sv_b == 2) then
    GOTO STEP2
```

```
endif


exitscript

# Populate Replay key to execute Step 1 or Step 2 of pre-script

STEP0:
sv_a = urhk_TBAF_InquireFieldValue (datablk1.nom_available_flg)
if (B2KTEMP.TEMPSTD.TBAFRESULT = "Y")
                  sv_a = urhk_TBAF_SetReplayKey("optnblk.key-
f2|tdoptpre.scr|1|tdoptpost.scr|1")
                  else
                  sv_a = urhk_TBAF_SetReplayKey("optnblk.key-
f2|tdoptpre.scr|2|tdoptpost.scr|2")
exitscript

# Populate Replay key to execute Step 2 of pre-script

STEP1:
                  sv_a = urhk_TBAF_SetReplayKey("optnblk.key-
f2|tdoptpre.scr|2|tdoptpost.scr|2")
exitscript

# Populate Replay key to execute Step 3 of pre-script

STEP2:
                  sv_a = urhk_TBAF_SetReplayKey("optnblk.key-f2|tdoptpre.scr|3")
exitscript


end-->
```

3.5 CHANGE THE ATTRIBUTE OF A FIELD IN THE FORM.

This function will change the attribute of a field in the form to the specified value.

SYNTAX

sv_a = urhk_TBAF_ChangeFieldAttrib(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("dispblk2.acct_bal|H")

FUNCTIONALITY

This user hook will change the attribute of a field in the form. This user hook should not be used in Pre Form Load and Post Form Load scripts. In the Pre Form Load scripts use the function **urhk_TBAF_SetAttrib.**

INPUT

Input string has two parts separated by a '|'

1. Field Name
2. Attribute

Example – "dispblk2.acct_bal|H"

Field name should be given as BlockName.FieldName.

Valid values for Attribute are:

P – Protect – The field is made non-enterable and tab will skip past this field

M – Mandatory – The field is made mandatory and something needs to be entered in this field before proceeding

H – Hide – The field value is not displayed

U - UnHide – The field value is displayed

E – Entry allowed – The cursor can be positioned in the field but no data can be entered. Useful if function keys need to be used in a protected field (to scroll the field or to explode for example)

OUTPUT

Always returns 0 as return value

The form will display the changed attribute immediately if the field is found in the current form, else nothing happens.

EXAMPLE:

Let us assume that a Bank wants to hide the account balance field in the Transaction maintenance screen for Customer accounts. This is achieved by coding the script as under.

```
Script Name - CheckAndDispAcctBal.scr

<--start

sv_a = BANCS.INPUT.AcctId

sv_b = urhk_getAcctDetailsInRepository(sv_a)

# if getAcctDetailsInRepository function returns failure exit out of script.

if( sv_b == 1 ) then
    exitscript
endif

                   sv_c="E"

# ----------------------------------------------------------------------------
                  # If the acctOwnerShip flag of ACCOUNT class is equal to "E"
                  # then hide the dispblk2.acct_bal field
                  # -----------------------------------------------------------------
-----------

if (BANCS.OUTPARAM.acctOwnerShip == sv_c) then
   sv_z = "dispblk2.acct_bal|H"
           sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
else
   sv_z = "dispblk2.acct_bal|U"
sv_z = urhk_TBAF_ChangeFieldAttrib(sv_z)
       endif
 exitscript
 end-->
```

3.6 GET THE 'POST-COMMIT' VALUE FROM A FIELD IN THE FORM.

This function will get the value of a field in the form, whose 'post-commit' value has been requested by using the **urhk_TBAF_SetValue** as described above. This function should be used only in a post-commit event of a form(Post Form Load event for example) or a Workflow script after a 'CallMenuOption' statement.

If the value of a form field is required to be accessed in any other event, then the function **urhk_TBAF_InquireFieldValue** should be used.

SYNTAX

sv_a = urhk_TBAF_GetValue (Variable)

The variable can be a scratch pad variable (sv_b) or a string ("baff0009.datablk6.cust_id")

FUNCTIONALITY

This function will typically be used in the context of a WorkFlow script, eventhough it can theoretically be used in any post-commit event. This function (along with **urhk_TBAF_SetValue)** is used to chain multiple menu options into a single business task by passing data from one step to another. For example, we could use this function in a 'Account Opening WorkFlow script' which executes 'CUMM', and 'OAAC' in sequence, to pass the Customer Id assigned in Customer creation step to the Account opening step.

INPUT

Input string has two parts separated by a '|'

1. Field name.
2. Repository [Optional]

Example – "baff0009.datablk6.cust_id"

Field name should be given as FormName.BlockName.FieldName.

Repository name is meant for advanced use (in the context of Workflow. See discussion on TEMP repositories under **urhk_WFS_TransferUser** and **urhk_WFS_CreateWFItem** sections) and typically should not be specified as shown in the example below. If not specified OUTTBAF is the repository that is assumed.

OUTPUT

Always returns 0 as return value.

The value of the required field will be available in the repository field B2KTEMP.TEMPSTD.TBAFRESULT if the field exists in the specified form and was correctly specified in **urhk_TBAF_SetValue** (with OUTTBAF repository) prior to commit. Otherwise, B2KTEMP.TEMPSTD.TBAFRESULT will contain "".

EXAMPLE:

In this example a sample script written to get the value of customer id from CUMM and put in account opening menu option OAAC.

```
<--start
STEP0
sv_a = urhk_TBAF_SetValue("baff0009.datablk6.cust_id| |OUTTBAF")
# we have told the system that we are interested in the post-commit value of
#baff0009.datablk6.cust_id
sv_a = urhk_WFS_CallMenuOption("CUMM|0|1")
if(sv_b == 1) then
   exitscript
endif
STEP1
sv_d = CLASSEXISTS("OUTTBAF","baff0009")
if (sv_d == 1) then
# We have asked the system to give us the post-commit value of
#baff0009.datablk6.cust_id
    sv_a = urhk_TBAF_GetValue("baff0009.datablk6.cust_id ")
    DELETECLASS("OUTTBAF","baff0009")
endif

sv_v = "bafe3013.funcblk.cust_id|" + B2KTEMP.TEMPSTD.TBAFRESULT
sv_a = urhk_TBAF_SetValue(sv_v)
#We have told the system that it should take the customer id assigned above as the
#customer id in the account opening form (bafe3013)
STEP2
sv_a = urhk_WFS_CallMenuOption("OAAC|2|")
if(sv_b == 1) then
   exitscript
endif

STDWFS.WFSINPUT.WfsStatus = "D"
exitscript
end-->
```

3.7 EXECUTING A KEY-BASED EVENT FROM THE SCRIPT.

This function will cause the execution of the Infosys defined logic associated with the specified key-based event, along with execution of a specified script before and after the execution of the specified key-based logic (refer to Finacle™ Forms Customisation manual for more details regarding how to determine what key based events are associated with a form), when the system encounters the specified Block. Only those key-based events that are already defined in the Infosys supplied form can be specified in the context of that form. Though this function can be used in any form-based event, its typical usage is in the context of a WorkFlow script.

SYNTAX

sv_a = urhk_TBAF_SetReplayKey (Variable)

The variable can be a scratch pad variable (sv_b) or a string ("funcblk.key-f2|TM1.scr|2|TM1.scr|3")

FUNCTIONALITY

This function gives the capability of automating certain operations in a form. Most Finacle™ forms follow a standard sequence of execution punctuated by data accepted from the user. For example a standard sequence in a maintenance form is 'enter values for field(s) in function block', hit the 'accept-key', 'enter values for field(s) in data block1', hit the 'next-blk' key(or accept-key and enter option in option block)', enter values for field(s) in data block2, hit the 'accept-key' and finally hit the 'commit key'. Now in certain 'WorkFlow' operations, values for a field in a form block maybe determined without any input from the user (either because default values can be supplied by the script or because values from a previous form can be supplied by the script or there are no values to be supplied). In such cases, this function can be used to automate the operation.

This function works like a 'stack' in the sense that multiple calls to this function will stack-up the 'Replay keys' one after another.

Having used this function, the logic of execution will be as follows:

1. Before accepting any user input in the current block, the system checks whether there is a 'Replay key' at the top of the stack, that has been specified for that block.
2. If so, then it executes the pre-script associated with that 'Replay key'(if any specified), executes the key-based event logic (which has been defined by Infosys), and then executes the post-script associated with that 'Replay key'(if any specified). It then removes the entry from the 'stack' so that the next 'Replay key' comes onto the top.
3. If no 'Replay key' has been specified it goes into the normal user input acceptance mode.

INPUT

Input string has three parts separated by a '|'

Block name.Key

PreScriptName|Step [Optional]

PostScriptName|Step [Optional]

Example - "funcblk.key-f2|TM1.scr|2|TM1.scr|3"

If the block name is not specified then key will be executed in the first block that is encountered after this 'Replay key' comes to the top of the 'stack'.

If Prescript name is specified then before executing the key-event the specified script will be executed

If Postscript is specified then specified script will be executed after the key-based event logic(as defined by Infosys) is executed.

Key can take on any valid values as documented in the Finacle™ Forms Customisation manual for 'Logical Keys'.

OUTPUT

Always returns 0 as return value.

When the PreScript or PostScript is executed, all the standard repository fields that is given out in any Finacle™ script event (Refer Scripting Events document)

will be given. In addition the specified step value will be available in the repository field INTBAF.INTBAFC.TbafEventStep.

EXAMPLE:

Let us assume that a Bank frequently opens a specific type of fixed deposit using a cash deposit from the Customer. So it decides that it needs a specific menu option for this, which will accept only the min basic details, create the account and post the cash transaction. The WorkFlow script shown below is an example of how to do this.

```
Script file name :- Opndepac.scr.
```

```
<--start

sv_d = CLASSEXISTS("STDWFS", "GLBDATA")
if (sv_d == 0) then
    CREATECLASS("STDWFS", "GLBDATA", 5)
endif
sv_b = cint(STDWFS.WFSINPUT.NextStep)

if (sv_b == 0) then
    GOTO STEP0
endif

if (sv_b == 1) then
    GOTO STEP1
endif

if (sv_b == 2) then
    GOTO STEP2
endif
exitscript

# upto now standard WorkFlow statements have been executed

STEP0
# Now accept the min. basic information regarding the deposit

STDWFS.WFSINPARAMVAL.custId="Customer Id"
STDWFS.WFSINPARAMVAL.SchemeCode="Scheme Code"
STDWFS.WFSINPARAMVAL.GlSubHeadCode="GL Sub Head Code"
STDWFS.WFSINPARAMVAL.depositAmount="Deposit Amount"
STDWFS.WFSINPARAMVAL.depPeriodMonths="Dep Prd Months"
STDWFS.WFSINPARAMVAL.depPeriodDays="Dep Prd Days"
sv_a = urhk_WFS_ShowParamAcptFrm("")

STEP1:

# Populate Scheme code, GL Sub Head Code, deposit Amount, Deposit Period in
# OAAC forms.

sv_v = "bafe3013.funcblk.schm_code|" +STDWFS.WFSOUTPARAM.schemeCode
sv_a = urhk_TBAF_SetValue(sv_v)
sv_v = "bafe3013.funcblk.gl_sub_head_code|" + STDWFS.WFSOUTPARAM.glSubHead
sv_v = "tdfe3201.datablk1.deposit_amount|" + STDWFS.WFSOUTPARAM.depositAmount
sv_a = urhk_TBAF_SetValue(sv_v)
sv_a = urhk_TBAF_SetValue(sv_v)
sv_v = "tdfe3201.datablk1.deposit_period_mths|" + STDWFS.WFSOUTPARAM.depPeriodMonths
sv_a = urhk_TBAF_SetValue(sv_v)
sv_v = "tdfe3201.datablk1.deposit_period_days|" + STDWFS.WFSOUTPARAM.depPeriodDays
sv_a = urhk_TBAF_SetValue(sv_v)

# Define field for getting the Account Number in OUTTBAF.

sv_a = urhk_TBAF_SetValue("tdfe3201.funcblk.acct_num| |OUTTBAF")
sv_a = 0
sv_a = urhk_WFS_CallMenuOption("OAAC|1|2")
if (sv_a == 1) then
   exitscript
endif

STEP2

# Get the account number and store in GLBDATA class.
sv_d = CLASSEXISTS("OUTTBAF","tdfe3201")
if (sv_d == 1) then
    sv_a = urhk_TBAF_GetValue("tdfe3201.funcblk.acct_num")
    STDWFS.GLBDATA.tmpAcctNum = B2KTEMP.TEMPSTD.TBAFRESULT
    DELETECLASS("OUTTBAF","tdfe3201")
endif

sv_a = urhk_TBAF_SetValue("bafe3012.funcblk.func_code|A")
sv_a = urhk_TBAF_SetValue("bafe3012.funcblk.tran_type|C")
sv_a = urhk_TBAF_SetValue("bafe3012.funcblk.tran_sub_type|NR")

sv_v = "bafe3012.datablk.tran_amt|" + STDWFS.WFSOUTPARAM.depositAmount
sv_a = urhk_TBAF_SetValue(sv_v)
```

```
sv_v = "bafe3012.datablk.acct_num|" + STDWFS.GLBDATA.tmpAcctNum
sv_a = urhk_TBAF_SetValue(sv_v)

# Play the key strokes so that Transaction will be created and posted.
# Play F2 to go to data block.
sv_a = urhk_TBAF_SetReplayKey("funcblk.key-f2")
# Play F2 to go to Option  block.
sv_a = urhk_TBAF_SetReplayKey("datablk.key-f2")
sv_a = urhk_TBAF_SetReplayKey("datablk1.key-f2")
sv_a = urhk_TBAF_SetReplayKey("dummyblk.key-f2")

# Play F2 at Option  block. and write tm1.scr to populate the 'P' - Post request
value
# and pass as pre script. Pre script will be executed before firing F2.
sv_a = urhk_TBAF_SetReplayKey("optionblk.key-f2|tm1.scr|0")
sv_a = urhk_TBAF_SetReplayKey("optionblk.key-nxtrec")
sv_a = urhk_TBAF_SetReplayKey("optionblk.key-f2")

# Play COMMIT at Option  block to post transaction.

sv_a = urhk_TBAF_SetReplayKey("optionblk.key-commit")

# Play KEY - F1 to exit form TM menu option.

sv_a = urhk_TBAF_SetReplayKey("dispblk3.key-f1")
sv_a = urhk_TBAF_SetReplayKey("funcblk.key-f1")

#Call the TM menu option for posting the cash transaction
sv_a = urhk_WFS_CallMenuOption("TM|2|")

STDWFS.WFSINPUT.WfsStatus = "D"
exitscript
end-->
```

In **tm1.scr** script

```
<--start
sv_b = cint(INTBAF.INTBAFC.TbafEventStep)
if (sv_b == 0) then
    GOTO STEP0
Endif
Exitscript

# Populate option code as P for posting the transaction.

STEP0:
sv_a = urhk_TBAF_ChangeFieldValue("optionblk.option_code|P")

exitscript
end-->
```

3.8 CALLING GENERAL PARAMETER ACCEPTANCE FORM

This function will invoke a specified form. The form should be script-enabled in the sense that it should operate only of repository variables, both for input and output. bafi2028 is an example of such a form.

SYNTAX

sv_a = urhk_B2K_ShowParamAcptFrm("bafi2028").

FUNCTIONALITY

This function will bring up the form specified in the argument. As of now only the form 'bafi2028' is supported. Each supported form will have its own inputs and outputs.

INPUT

The call to the function requires a form name which is to be brought up for accepting data from the user. As of now only 'bafi2028' is supported.

Inputs for bafi2028

The form 'bafi2028' requires certain inputs for bringing up the form, displaying the literals and accepting data. The inputs are to be given by way of a repository field in FRMATTRIB class of BANCS repository, for each field that needs to be displayed/accepted. For each field the following is required to be given

1) The name of the field - assigned by referring to this name while populating the rest of the information for the field.

2) The literal for the field – first part of the value assigned to the above variable. Maximum length for literal is 30 characters

3) Length of the value of the field - second part of the value assigned to the above variable separated by '|'. Maximum length for value is 100 characters

4) Field to be protected or not (P/N) – third part of the value assigned to the above variable separated by '|' Protected field

Ex: BANCS.FRMATTRIB.senderSwiftCode="Sender Swift Code|12|P"

```
                    |                    |                  |
                Field name            Literal         Field length
```

If a default value needs to be specified for any field then that value needs to be assigned to the corresponding output field. For ex. To assign a default value of 'XXX' for the senderSwiftCode, we should do the following before calling this function :

BANCS.FRMVALUE.senderSwiftCode = "XXX"

A well known field BANCS.INPARAM.ErrorMesg can be used to force the form to display an Error message.

A well known field BANCS.FRMVALUE.FormTitle can be used to display the title in the form.

OUTPUT

Always returns 0 as return value.

Output from bafi2028

The values entered in the screen for the specified fields will be available in fields of FRMVALUE class of BANCS repository, after the user hits the <ACCEPT> or <COMMIT> key in the form. The name of the field will be the same as the name assigned in the input described above. i.e. the value that the user entered for the 'senderSwiftCode' field will be available in BANCS.FRMVALUE.senderSwiftCode field.

EXAMPLE:

Example for bafi2028

The following is a sample script which is used to manage a Swift message(mt410) in Finacle™. Amoung other things it requires user input for certain fields like sender swift code, Sender to receiver info etc. These user input are taken by using the **urhk_B2K_ShowParamAcptFrm("bafi2028")** function.

```
<--start
trace on
#************************************************************************
# Script for generation of MT410 swift message
#********************************************************************
#
#   While forming the message, even if the optional fields are not
#   concatenated, a new line separator"|"(pipe character)
#   should be concatenated.
#
#   In Generate mode, apart from other INPUT repository variables
#   (addOrMod) will be populated by the calling program.
#
#   In Modify mode, INPUT repository variables (status, addOrMod)
#   will  be populated by the calling program. Also, all the field[n]
#   INPUT variables will be populated.
#
#********************************************************************

# *******************************************************
# Populate the FRMVALUE fields for display in bafi2028
# in modify mode.
# *******************************************************

if (BANCS.INPUT.addOrMod == "M") then
    BANCS.FRMVALUE.senderSwiftCode      =BANCS.INPUT.field1
    BANCS.FRMVALUE.mtNumber             =BANCS.INPUT.field2
    BANCS.FRMVALUE.receiverSwiftCode    =BANCS.INPUT.field3
    BANCS.FRMVALUE.billId               =BANCS.INPUT.field4
    BANCS.FRMVALUE.referenceNum         =BANCS.INPUT.field5
    BANCS.FRMVALUE.amountAcknowledged   =BANCS.INPUT.field6
    BANCS.FRMVALUE.senderToReceiverInfo1=BANCS.INPUT.field7
    BANCS.FRMVALUE.senderToReceiverInfo2=BANCS.INPUT.field8
    BANCS.FRMVALUE.senderToReceiverInfo3=BANCS.INPUT.field9
    BANCS.FRMVALUE.senderToReceiverInfo4=BANCS.INPUT.field10
    BANCS.FRMVALUE.senderToReceiverInfo5=BANCS.INPUT.field11
    BANCS.FRMVALUE.senderToReceiverInfo6=BANCS.INPUT.field12
endif

# *******************************************************
#   Massage the field values present in OUTPARAM
#   when in ADD mode
# *******************************************************
if (BANCS.INPUT.addOrMod == "A") then

    # ------------------------------
    # populate MT number
    # ------------------------------
    BANCS.FRMVALUE.mtNumber="410"

    # ------------------------------
    # Populate amount Acknowledged
    # ------------------------------
    sv_s=BANCS.FRMVALUE.billAmount
    call("swiftAmount.scr")
    BANCS.FRMVALUE.billAmount= sv_t
    sv_w=BANCS.FRMVALUE.dueDate
    call("swiftDate.scr")
    BANCS.FRMVALUE.dueDate = sv_x
    sv_a = sv_x + BANCS.FRMVALUE.billCrncy
    sv_a = sv_a + BANCS.FRMVALUE.billAmount
    BANCS.FRMVALUE.amountAcknowledged = sv_a

    # ------------------------------
    # Sender to Receiver information
    # ------------------------------
    BANCS.FRMVALUE.senderToReceiverInfo1=""
    BANCS.FRMVALUE.senderToReceiverInfo2=""
    BANCS.FRMVALUE.senderToReceiverInfo3=""
    BANCS.FRMVALUE.senderToReceiverInfo4=""
    BANCS.FRMVALUE.senderToReceiverInfo5=""
    BANCS.FRMVALUE.senderToReceiverInfo6=""

    # ------------------------------
    # default value for status of message
```

```
        # -------------------------------
        BANCS.FRMVALUE.status="N"
    endif
    # Massaging of data in ADD mode complete

    # ******************************************************
    #   Populate the FRMATTRIB fields - Provide definition for  the bafi2028 form
    # ******************************************************

    BANCS.FRMVALUE.FormTitle="MT410 SWIFT Message Generation"

    BANCS.FRMATTRIB.senderSwiftCode="Sender Swift Code|12|P"
    BANCS.FRMATTRIB.mtNumber="MT|3|P"
    BANCS.FRMATTRIB.receiverSwiftCode="Receiver Swift Code|12|P"
    BANCS.FRMATTRIB.billId="Sending Bank's TRN|16|P"
    BANCS.FRMATTRIB.referenceNum="Related reference|16|P"
    BANCS.FRMATTRIB.amountAcknowledged="Amount Acknowledged|24|P"
    BANCS.FRMATTRIB.senderToReceiverInfo1="Sender to Receiver Info|35|N"
    BANCS.FRMATTRIB.senderToReceiverInfo2=" |35|N"
    BANCS.FRMATTRIB.senderToReceiverInfo3=" |35|N"
    BANCS.FRMATTRIB.senderToReceiverInfo4=" |35|N"
    BANCS.FRMATTRIB.senderToReceiverInfo5=" |35|N"
    BANCS.FRMATTRIB.senderToReceiverInfo6=" |35|N"
    # -------------------------------
    # status field
    # -------------------------------
    BANCS.FRMATTRIB.status="Status of Message|1|N"


    # ******************************************************
    # Call general acceptance form
    # Conditional validations can be done here.
    # ******************************************************
    BANCS.INPARAM.ErrorMesg = " "
    while(BANCS.INPARAM.ErrorMesg != "")
        sv_a = urhk_B2K_ShowParamAcptFrm("bafi2028")
        sv_b = BANCS.FRMVALUE.status
        if ((sv_b == "N") or (sv_b == "R") or (sv_b == "D")) then
            BANCS.INPARAM.ErrorMesg = ""
        else
            BANCS.INPARAM.ErrorMesg = "Status can have only the following values. [N-Not
    ready, R
    -Ready, D-Delete]"
        endif
    do

    # ******************************************************
    # Form the swift message
    # For optional tags check if the value is null or not
    # ******************************************************
    # -------------------------------
    # Form MT410 message
    # -------------------------------
    sv_a = BANCS.FRMVALUE.senderSwiftCode
    sv_a = sv_a + "|" + BANCS.FRMVALUE.mtNumber
    sv_a = sv_a + "|" + BANCS.FRMVALUE.receiverSwiftCode
    sv_a = sv_a + "|" + ":20:" + BANCS.FRMVALUE.billId
    sv_a = sv_a + "|" + ":21:" + BANCS.FRMVALUE.referenceNum
    sv_a = sv_a + "|" + ":32A:" + BANCS.FRMVALUE.amountAcknowledged

    if ((BANCS.FRMVALUE.senderToReceiverInfo1 != "") or
    (BANCS.FRMVALUE.senderToReceiverInfo2 !=
    "") or (BANCS.FRMVALUE.senderToReceiverInfo3 != "") or
    (BANCS.FRMVALUE.senderToReceiverInfo4
    != "")or (BANCS.FRMVALUE.senderToReceiverInfo5 != "") or
    (BANCS.FRMVALUE.senderToReceiverInfo
    6 != "")) then
        sv_a = sv_a + "|" + ":72:" + BANCS.FRMVALUE.senderToReceiverInfo1
        sv_a = sv_a + "|" + BANCS.FRMVALUE.senderToReceiverInfo2
        sv_a = sv_a + "|" + BANCS.FRMVALUE.senderToReceiverInfo3
        sv_a = sv_a + "|" + BANCS.FRMVALUE.senderToReceiverInfo4
        sv_a = sv_a + "|" + BANCS.FRMVALUE.senderToReceiverInfo5
        sv_a = sv_a + "|" + BANCS.FRMVALUE.senderToReceiverInfo6
    else
        sv_a = sv_a + "||||||"
```

```
endif

BANCS.OUTPUT.swiftMessage = sv_a

exitscript
end-->
```

3.9 DISPLAY A MESSAGE AT THE BOTTOM LINE IN A FORM.

This function will display a given message at the bottom line (the place where messages are generally shown in a form) in a form.

SYNTAX

sv_a = urhk_TBAF_DispMesg (Variable)

The variable can be a scratch pad variable (sv_b) or a string ("Value cannot be greater than 1000")

FUNCTIONALITY

This user hook will display a given message at the bottom line of a form when invoked. This can be used to specify any error messages that the user would want to during some processing in a script.

INPUT

Input string has only one part

The message to be displayed.

Ex : – "Value cannot be greater than 1000"

OUTPUT

Always returns 0 as return value

The message will be displayed at the bottom line of the form once the control is returned back.

EXAMPLE

Let us assume that a Bank wants to display a message "Value cannot be greater than 1000" in case it is so during a script processing.

```
          <--start

sv_a = BANCS.INPUT.tran_amount

if (sv_a > 1000) then
   sv_b = urhk_TBAF_DispMesg(sv_e)
   exitscript
else
   # go ahead further
endif
exitscript
end-->
```

3.10 SPECIFY ADDITIONAL ACTIONS FOR A FUNCTION KEY AT ANY FIELD

This function will enable the user to specify a script name that he would like to be invoked at the pressing of a specified function key at a specified field on a form, in addition to the normal action for that key.

SYNTAX

sv_a = urhk_TBAF_SetKeyScript(Variable)

The variable can be a scratch pad variable (sv_b) or a string ("baff0009.decision_blk.cust_option_code.key-nxtfld| CUMMadet.scr|1”)

FUNCTIONALITY

This user hook will set the script name specified, as the script to be invoked when a particular function key is pressed on a particular field. Two scripts can be specified, one to be executed before processing the function key pressed and one to be executed after processing the function key. The step number for each of these scripts can also be specified.

Once set in this manner, whenever the specified function key is hit on the speicifed field of a form, the specified pre script gets executed (at specified step) followed by the normal processing for that function key followed by the specified post script (at specified step).

At least one of the pre or post scripts is mandatory, though one can specify both.

INPUT

Input string has five (5) parts separated by a ‘|’

<formname>.<blockname>.[<fieldname>.]<function-key-name>

Name of the script to be executed before key processing

Step number of above script to be executed

Name of the script to be executed after the key processing

Step number of the above script to be executed

The function-key-name must be a valid logical key name like key-f2 or key-nxtfld. Also the fieldname is not mandatory. If not specified, then it means for any field.

Following are examples of some valid input strings :

"baff0009.funcblk.key-f2|||CUMMadet.scr|3"

This means that the script CUMMadet.scr at step 3 must be executed whenever key-f2 is pressed on form baff0009 at block funcblk. The script is a post script (4th part of input) and hence must be executed after the normal processing for the key-f2 happens.

"baff0009.decision_blk.cust_option_code.key-nxtfld|CUMMadet.scr|1"

This means that the script CUMMadet.scr at step 1 must be executed whenever key-nxtfld is pressed on the field cust_option_code of decision_blk in form baff0009. Note that in this example, the script is executed before the key-nxtfld logic is processed since the script was specified as a prescript (2nd part of input).

Formname and blockname is mandatory. Field name is optional (if not specified, it would mean for any field). Function Key must be specified. The step numbers are not mandatory parts. One of pre or post scripts must be specified.

Valid values for function keys are all valid keys specified in the crd file that begin with a 'key-'

OUTPUT

Always returns 0 as return value

Once the key script is set, the execution of the form will ensure that the appropriate scripts are run at appropriate times.

EXAMPLE

In the preload script of CUMM menu option one could incorporate this feature as shown in the example below :

baff0009PreLoad.scr :

```
<--start
# baf0009PreLoad.scr
# This sets up key events for passing control to scripts
# when the baff0009 form is loaded either thru the CUMM option or
# 'Customer explode'. This allows the "X" option in the option code to be
# processed by the CUMMadet.scr . It also sets certain fields as mandatory
sv_a = urhk_TBAF_SetKeyScript("baff0009.decision_blk.cust_option_code.key-
nxtfld|CUMMadet.scr|1")
sv_a = urhk_TBAF_SetKeyScript("baff0009.decision_blk.cust_option_code.key-
commit|||CUMMadet.scr|2")
sv_a = urhk_TBAF_SetKeyScript("baff0009.funcblk.key-f2|||CUMMadet.scr|3")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk1.cust_sex|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk1.cust_const|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk1.cust_introd_stat_code|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk2.cust_comu_addr1|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk2.cust_comu_city_code|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.minor_datablk.minor_guard_name|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.minor_datablk.minor_guard_addr1|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.minor_datablk.minor_guard_city|M")
sv_a = urhk_TBAF_SetAttrib("baff0009.datablk.nre_nationality|M")
exitscript
end-->
```

4 WORKFLOW SCRIPTING FUNCTIONS

(Can be used only in the context of any WorkFlow script) (A WorkFlow script is any script that is tied to a menu option using the 'Customisation' feature of the Finacle™ menu. Refer to WorkFlow manual for more details)

4.1 EXECUTING A MENU OPTION

This function provides a facility to execute a menu option available in BANCS200 from a WorkFlow script.

SYNTAX

```
sv_a = urhk_WFS_CallMenuOption (Variable)
if(sv_a == 1) then
        exitscript
endif
```

The variable can be a scratch pad variable (sv_b) or a string ("CUMM|0|1")

FUNCTIONALITY

This function will execute the menu option specified in the input string. This will be useful while writing a script for workflow to chain different menu options to execute one after the other to complete the workflow operation.

INPUT

Input String contains three parts separated by a '|'

Menu option – As displayed in the Finacle™ menu screen

Current Step.

Next Step [Optional].

Example - "CUMM|0|1"

Menu option can be any FINACLE™ menu option other than a Workflow type of menu option.

Current Step is the step in which this statement is being executed. This is required so that even if the called menu option is in a different executable, the script's execution continues correctly.

'Next step' is the step that a workflow script should be restarted at, if the system aborts during the workflow after the currently called menu option commits its changes. It is required to be specified only for 'update' menu options, and typically should specify the next step in the script. In effect it is equal to calling the function **urhk_WFS_CheckPoint(Next Step)** immediately after calling the 'update' menu option.

OUTPUT

The return value will be ZERO (0) in case the menu option is in the same executable, after executing the menu option. The return will be 1 if the menu option is in another executable, before executing the menu option. In this case the script should exit, so that the system can switch exe and continue the execution of the script in the context of the new executable.

Example:

Please see example of **urhk_WFS_ShowParamAcptFrm function below.**

4.2 CALLING INPUT ACCEPTANCE FORM

This function will invoke a standard user acceptance form, which can be called in workflow script to accept additional data during the workflow. See also **urhk_B2K_ShowParamAcptFrm("bafi2028").**

SYNTAX

sv_a = urhk_WFS_ShowParamAcptFrm("").

FUNCTIONALITY

This function will bring a standard form with all fields defined in WFSINPARAMVAL class to accept additional data that may be required during workflow.

A maximum of 34 fields can be accepted in one call to this function.

INPUT

NULL

This function requires certain inputs for bringing up the form, displaying the literals and accepting data. The inputs are to be given by way of a repository fields in WFSINPARAMVAL and WFSINPARAMLEN class of STDWFS repository, for each field that needs to be displayed/accepted. For each field the following is required to be given

1) The name of the field - assigned by referring to this name while populating the literal and length of the field.

2) The literal for the field – Maximum length for literal is 20 characters

The length of the field - Maximum length for value is 70 characters

```
Ex: STDWFS.WFSINPARAMVAL.ledgerNo ="Ledger No."
      STDWFS.WFSINPARAMLEN.ledgerNo ="5"
```

```
If a default value needs to be specified for any field then that value needs to be
assigned to the corresponding output field. For ex. To assign a default value of
'XXX' for the ledgerNo, we should do the following before calling this function :
STDWFS.WFSOUTPARAM.ledgerNo = "XXX"
A well known field STDWFS.WFSINPARAMVAL.ErrorMesg can be used to force the form to
display an Error message.
A well known field STDWFS.WFSINPARAMVAL.FormTitle can be used to display the title in
the form.
```

OUTPUT

Always returns 0 as return value.

The values entered on the screen will be available in WFSOUTPARAM class under the corresponding name in WFSOUTPARAM class, after the user presses <ACCEPT> or <COMMIT> key in the form.

5 INDEX OF USER HOOKS

urhk B2k_GetAccountsForInstrument
  Get List of Amounts based on an Instrument....16
urhk_ B2k_CdciPopulateFeeInfoLL 114
urhk_ B2k_valBankCode
  validate a Bank Code 17
urhk_ generatePastDueReport 119
urhk_ MTTS_DefineAsPerCustomerInstruction....101
urhk_ MTTS_DefinePartTran 97
urhk_B2k_BacidToAcctId
  bacid to Acccount Id and Name....11
urhk_B2k_checkForPipeChar 42
urhk_B2k_ConvertAmount
  Convert an Amount from One Currency to another....12
urhk_B2k_date_add
  date Add....23
urhk_B2k_GetASTMAmount
  Get amount calcul 15
urhk_B2k_getDateDiffFromBODDate....45
urhk_B2k_GetUserAdditionalDetails....121
urhk_B2k_getYear
  get 4 Digit year 23
urhk_B2k_isStringNumeric
  Validate Numeric....20
urhk_B2k_PutUserAdditionalDetails....122
urhk_B2k_PutUserException 123
urhk_B2k_RoundOff
  Roundoff An Amount 14
urhk_B2K_ShowParamAcptFrm....65
urhk_B2k_valBankBranchCode
  Validate Bank and Branch Code....18
urhk_B2k_valCashAccount....42
urhk_B2k_valCrncy 45
urhk_B2k_valFCNRAccount 43
urhk_B2k_valHOAccount....44
urhk_B2k_valHoTranCode
  Validate HO Transaction Code....19
urhk_B2k_validateAmtWithCrncy....44
urhk_B2k_validateInstrmntDate 48
urhk_B2k_valSystemAccount....43
urhk_B2k_valTranRptCode....46
urhk_dbSelect
  Selection of Fields from database 24
urhk_dbSQL
  Run a SQL Statement....25
urhk_EVTSCR_CreateWFItem....113
urhk_EVTSCR_InitWFRepos....111
urhk_FTS_RaiseException....108
urhk_getAcctBalanceAsOnDate
  Get Account Balance as on a date....35
urhk_getAcctDetailsInRepository
  get Account details into BANCS Repository.... 4
urhk_getAdvanceDetailsForAccount
  get Advance Details for an account 26
urhk_getAssetClassfnDetailsForAccount
  get Asset Classification for an Account....32

urhk_GetCTCDetailsFromTranCode 87
urhk_getCustomerDetails
  Get customer details into BANCS repository 28
urhk_getDDSchemeCrncyDetails 46
urhk_getFileLocation
  Getting the location of a File in the path 9
urhk_getInterestDetailsForAccount
  Get Interest Details for an account..........33
urhk_getInterestRateForAccount
  get Interest rate for an Account..........36
urhk_getOutClgInstDetails..........91
urhk_getOutClgPtranDetails 94
urhk_getOutZoneHeaderDetails..........90
urhk_getPastDueRecord..........115
urhk_getSchemeDetailsInRepository
  get Scheme Details into BANCS Repository.......... 8
urhk_getSecurityDetailsForAccount 40
urhk_getSumCrDrTranAmtsForPeriod
  Sum credit & Debit Transactions over a period..........37
urhk_getSumTranAmtForAcctLabel..........39
urhk_GetZoneHeaderDetails..........88
urhk_MTTS_CheckIfCustInstExists 100
urhk_MTTS_DefineChargePartTran..........103
urhk_MTTS_DefineTransaction 96
urhk_MTTS_EndDefinition 99
urhk_MTTS_GetPttRect..........105
urhk_MTTS_InitPtt..........104
urhk_MTTS_PopulateInputDetails 96
urhk_MTTS_PopulateSubInputDetails 123
urhk_MTTS_ReportErrorCondition..........100
urhk_OWCLG_RaiseException 109
urhk_putPastDueDispPrintString 116
*urhk*_PutUDTXN_into_IBR..........49
urhk_SE_DispRepositories
  Display Repository Fields..........26
urhk_TBAF_ChangeFieldAttrib..........58
urhk_TBAF_ChangeFieldValue..........55
urhk_TBAF_DispMesg 70
urhk_TBAF_GetCurFldName..........70
urhk_TBAF_GetValue 60
urhk_TBAF_InquireFieldValue 55
urhk_TBAF_SetAttrib..........51
urhk_TBAF_SetKeyScript 71
urhk_TBAF_SetReplayKey 61
urhk_TBAF_SetValue..........53
urhk_valAcctNumber
  validating the Account Number 3
urhk_WFS_CallMenuOption..........74
urhk_WFS_CheckPoint..........84
urhk_WFS_CreateWFItem..........75
urhk_WFS_SetWorkGroup 86
urhk_WFS_ShowParamAcptFrm..........80
urhk_WFS_TransferUser 79
GetDiffBetweenTwoFreq..........4
  9
B2K_IsCheckSumRight..........4
  9
URHK_
  GETACCTTURNOVERDETAILS..........49

urhk_ getDSATurnoverDetails……………………………………………………………………….123
urhk_ getDSAMasterDetails………………………………………………………………………….123
urhk_
getDSAProductDetails……………………………………………………………………………123
urhk_ getDSAParamDetails………………………………………………………………………...123
urhk_getPastDueCycles………………………………………………………………………………123