Cisco Pre-Paid Debitcard Multi-Language Programmer’s Reference

Version: 9/29/00

This document contains information about the Multi-Language feature, including the following sections:

- Overview of the Multi-Language Feature, page 1
- Prerequisites, page 1
- Multi-Language Feature Implementation, page 3
- Cisco Connection Online, page 16
- Call Flow Example, page 17

Overview of the Multi-Language Feature

The baseline prepaid debitcard application supports the playing of pre-recorded static and dynamic prompts for English, Spanish, and Chinese (Mandarin). The building of the dynamic prompts for these languages is performed as part of the Cisco Internetwork Operating System (IOS). If an additional language is needed, it has to be delivered as a change to the Cisco IOS and follow the Cisco IOS release process. This can be a very time-consuming and costly process. The Multi-Language feature provides the user with the ability to accommodate languages, other than those originally implemented in the Cisco IOS, by making changes within the debitcard Tool Command Language (TCL) application. Because the modifications are at the script level, the user can implement any defined language. Script changes are relatively easy when the language’s dynamic prompt rules are supplied and translated into the script.

Language exceptions are defined as languages outside the originally supported three languages. This document concentrates on the building of dynamic prompts for language exceptions, rather than static prompts, because the script handles static prompts the same way for all languages. When a language is selected, the script prepends a language prefix and passes that prompt to the interactive voice response (IVR) Application Programming Interface (API) playPrompt verb.

Prerequisites

To use the Multi-Language feature you need the following:

- The Multi-Language feature script debitcard_multi-lang_Cisco.1.1.0.0.tcl or later
- Complete set of professionally recorded language prompts for each configured language. Languages available with the Multi-Language feature debitcard script are English (en), Spanish (sp), Mandarin (ch), Thai (th), Cantonese (ca), Japanese (ja), and Russian (ru).
- Cisco IOS Release 12.1(3)xi or later.
- The Multi-Language feature is based on the debitcard application; therefore, it requires a RADIUS billing server for AAA.

Restrictions and Configuration

- When you configure the languages, you must configure them sequentially. You cannot skip language numbers. However, the languages themselves can be configured in any order. Refer to the following example:

```
router# call application voice debit
tftp://dirt/script/debitcard_multi-lang_Cisco.1.1.0.0.tcl
router# call application voice debit uid-len 6
router# call application voice debit language 1 en
router# call application voice debit language 2 th
router# call application voice debit language 3 ru
router# call application voice debit language 4 ja
router# call application voice debit language 5 ca
router# call application voice debit language 6 sp
router# call application voice debit language 7 ch
router# call application voice debit set-location en 0 tftp://dirt/au/en/
router# call application voice debit set-location th 0 tftp://dirt/au/th/
router# call application voice debit set-location ru 0 tftp://dirt/au/ru/
router# call application voice debit set-location ja 0 tftp://dirt/au/ja/
router# call application voice debit set-location ca 0 tftp://dirt/au/ca/
router# call application voice debit set-location sp 0 tftp://dirt/au/sp/
router# call application voice debit set-location ch 0 tftp://dirt/au/ch/
```

- When setting up the language selection prompts associated with the configured languages, you must put all the actual files in the language 1 configured directory. Refer again to the above example. The files th_lang_sel2.au, ru_lang_sel3.au, ja_lang_sel4, ca_lang_sel5, sp_lang_sel6 and ch_lang_sel7 must all reside in directory tftp://dirt/au/en/. All other language prompts reside in the individual language directories.
- The maximum amount that the script handles for $amt and creditTime converted to hours and minutes is 999,999.99.

Related Documentation

- *TCL IVR API Version 1.0 Programmer's Guide*
  Version 1.0 commands for writing TCL scripts for the Cisco IVR feature on the Cisco AS5300 gateway.
- *TCL IVR API Version 2.0 Programmer's Guide*
  Version 2.0 commands for writing TCL scripts for the Cisco IVR feature on the Cisco AS5300 gateway.
- *RADIUS Vendor-Specific Attributes Voice Implementation Guide*
  Reference of Cisco RADIUS VSAs for use with VoIP call authorization.

Developer Support

Developers using this programmer's guide may be interested in joining the Cisco Developer Support Program. This new program has been developed to provide you with a consistent level of support that you can depend on while leveraging Cisco interfaces in your development projects.

A signed Developer Support Agreement is required to participate in this program. For more details, and access to this agreement, please visit us at **http://www.cisco.com/warp/public/779/servpro/programs/ecosystem/devsup/** or contact **developer-support@cisco.com**.

Multi-Language Feature Implementation

The Multi-Language feature is built on the application infrastructure of the debitcard script. The script is rewritten, but the call flow from debitcard is essentially unmodified. The original call flow is followed through all procedures for selecting a language, collecting a card number, performing authorization, and placing a call. The modifications necessary for the Multi-Language feature were made to procedures do_get_dest and do_second_authorization to accommodate the playing of pre-recorded dynamic creditAmount and creditTime for the new languages.

How to Use the Multi-Language Feature

This section of the document includes information on how to use the feature:

Step 1 Build the language's master prompt spreadsheet. This will be used to determine the language exceptions and how the language should build dynamic prompts.

Step 2 Determine what numbering structure the language belongs to. Is it *SSDHTT*, *SDDHTT*, or another form? Refer to the example in Table 1 to determine if modifications in procedure do_whole_part are required.

Step 3 Based on the numbering structure, modify the necessary procedures. If a language has a dynamic prompt exception for a digit, as defined in Table 1, then add the language prefix to the switch in the necessary procedure (do_tens, do_hundreds, do_thousands, or do_hthousands). See "Modifying the Existing Script" on page 4. Under most circumstances, the switch default case should not have to be modified.

Table 1 Language Exception Number Translation Form

	Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6
Cantonese (ca)	S	S	D	H	T	T
Thai (th)	S	D	D	H	T	T
Japanese (ja)	S	S	D	H	T	T
Russian (ru)	S	D	D	H	T	T

Testing and Debugging Your Multi-Language Script

Sample test tools and cases are bundled with the script.

Modifying the Existing Script

Modification of the existing script is based on the specific constructs of the required language. Each modification is different but is made to some or all of the procedures:

- do_get_dest
- do_second_authorization
- do_whole_part
- do_decimal_part
- do_creditTime_prompt
- do_tens
- do_hundreds
- do_thousands
- do_hthousands

The modifications are made by adding a new TCL switch case for the language prefix with the TCL statements that define the language dynamic prompt to the switch statement.

New TCL Procedures

Included in this guide are seven new TCL procedures that can be used to build the language-specific dynamic prompts, while modifications were made in procedures do_get_dest and do_second_authorization. To implement a language exception, the procedures do_get_dest and do_second_authorization, as well as at least some of the other seven procedures, must be modified.

Proc do_get_dest

This modified procedure is used after successful first authorization. A TCL switch statement is included that directs the flow to the language exception procedures or uses the default for Cisco IOS implemented languages. For the TCL implemented language exceptions, this procedure separates the billing server returned amount (creditAmt) into two parts and calls the prompt building procedures of do_whole_part and do_decimal_part.

Note Some TCL script implementations must be based on the returned values from the RADIUS billing server for creditAmt and creditTime.

Example 1 Sample Modifications to Procedure do_get_dest

```
switch -regexp $prefix {
  {ru} {set newlist [split $amt .]
        set gender m

        do_whole_part

        if {$len == 1} {
            switch -regexp $numbers(tens) {
                {0}        { }
                {1}        {lappend prompt "[set prefix]_dollar.au"}
                {[2-4]}    {lappend prompt "[set prefix]_2-4_dollars.au"}
                {[5-9]}    {lappend prompt "[set prefix]_5-20_dollars.au"}
               }
        } else {
               switch -regexp $numbers(tens) {
                  {[0-9]*[0|2-9][2-4]} {lappend prompt "[set prefix]_2-4_dollars.au"}
                  {[0-9]*[2-9][1]}     {lappend prompt "[set prefix]_dollar.au"}
                  default              {lappend prompt "[set prefix]_5-20_dollars.au"}
               }
        }

        set gender f
        do_decimal_part

        if {[string length [lindex $newlist 1]] == 1} {
               switch -regexp [lindex $newlist 1] {
                  {0}        { }
                  default    {lappend prompt "[set prefix]_5-20_cents.au"}
               }
        } else {
               switch -regexp [lindex $newlist 1] {
                  {00}           { }
                  {[0|2-9][1]}   {lappend prompt "[set prefix]_cent.au"}
                  {[0|2-9][2-4]} {lappend prompt "[set prefix]_2-4_cents.au"}
                  default        {lappend prompt "[set prefix]_5-20_cents.au"}
               }
        }
        puts "\t\t**** playPrompt param3 info [set prefix]_you_have.au $prompt [set prefix]_enter_dest.au"
        set ev [eval [list playPrompt param3 info [set prefix]_you_have.au] $prompt [list %s1000 [set
prefix]_enter_dest.au]]
  }
  default {set ev [playPrompt param3 info [set prefix]_you_have.au %a$amt %s1000 [set
prefix]_enter_dest.au]}
}
```

Proc do_second_authorization

This is a modified procedure, used after successful authorization to play the dynamic prompt for creditTime. A TCL switch statement is included that directs the flow to the language exception procedures or uses the default for Cisco IOS implemented languages.

Proc do_whole_part

This new procedure separates the whole number string into its numeric prompt components for tens, hundreds, thousands, and hundred thousands. This is explained in more detail in Table 1 on page 3. This procedure places these values in the numbers array and calls the appropriate procedure based on the arrays index containing a value, as shown in the following example.

Example 2 Example of Procedure do_whole_part

```
global amt
    global prefix
    global prompt
    global numbers
    global len
    global newlist

    set numbers(tens) ""
    set numbers(hundreds) ""
    set numbers(thousands) ""
    set numbers(hthousands) ""

    set len [string length [lindex $newlist 0]]

    set separate [split [lindex $newlist 0] ""]

    puts "\t\t**** TTS whole part: [lindex $newlist 0]"

# Do the number group translation

    switch -regexp $len {
        {1}  { set numbers(tens) [lindex $separate 0]}
        {2}  { set numbers(tens) [join [list [lindex $separate 0] [lindex $separate 1]] "" ]}
        {3}  { set numbers(hundreds) [lindex $separate 0]
              set numbers(tens) [join [list [lindex $separate 1] [lindex $separate 2]] "" ]}
        {4} { set numbers(thousands) [lindex $separate 0]
              set numbers(hundreds) [lindex $separate 1]
              set numbers(tens) [join [list [lindex $separate 2] [lindex $separate 3]] "" ]}
        {5}  {switch -regexp $prefix {
                 {ca} -
                 {ja}    {set numbers(hthousands) [lindex $separate 0]
                          set numbers(thousands) [lindex $separate 1]}
                 default {set numbers(thousands) [join [list [lindex $separate 0] [lindex $separate 1]] ""
]}
              }
              set numbers(hundreds) [lindex $separate 2]
              set numbers(tens) [join [list [lindex $separate 3] [lindex $separate 4]] "" ]
        }
        {6}  {switch -regexp $prefix {
               {ca} -
               {ja} {set numbers(hthousands) [join [list [lindex $separate 0] [lindex $separate 1]] "" ]
                     set numbers(thousands)  [lindex $separate 2]}
               default {set numbers(hthousands) [lindex $separate 0]
                        set numbers(thousands) [join [list [lindex $separate 1] [lindex $separate 2]] "" ]}
              }
              set numbers(hundreds) [lindex $separate 3]
              set numbers(tens) [join [list [lindex $separate 4] [lindex $separate 5]] "" ]
        }
   }
   foreach index {hthousands thousands hundreds tens} {

       if {[string compare $numbers($index) ""] != 0} {

           puts "\t\t**** Executing do_$index $numbers($index)"
           do_$index
       }
```

Proc do_tens

This new procedure is used to build the language dynamic prompt for the value stored in the numbers (tens) array index. In this procedure, the language exception code is written if the language handles the tens numbers different than the default method.

Example 3 Example of Procedure do_tens

```
proc do_tens {} {
    global prompt
    global prefix
    global numbers
    global len
    global gender

    if {$numbers(tens) != 0} {
          switch -regexp $prefix {
             {th} {if {$len == 1} {
                         lappend prompt "[set prefix]_$numbers(tens).au"
                 } else {set separate [split $numbers(tens) ""]
                              switch -regexp $numbers(tens) {
                                 {00}        { }
                                 {10}        {lappend prompt "[set prefix]_$numbers(tens).au"}
                                 {[0][1-9]}  {lappend prompt "[set prefix]_[lindex $separate 1].au"}
                                 {[1][1-9]}  {lappend prompt "[set prefix]_10.au" "[set prefix]_[lindex
$separate 1].au" }
                                 {[2][0]}    {lappend prompt "[set prefix]_20.au"}
                                 {[2][1-9]}  {lappend prompt "[set prefix]_20.au" "[set prefix]_[lindex
$separate 1].au"}
                                 {[3-9][0]}  {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set
prefix]_10.au"}
                                 default     {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set
prefix]_10.au" "[set prefix]_[lindex $separate 1].au"}
                              }
                }
            }
            default {if {$len == 1} {
                       switch -regexp $numbers(tens) {
                                {0}     { }
                                default {lappend prompt "[set prefix]_$numbers(tens).au"}
                             }
                        } else {set separate [split $numbers(tens) ""]
                                   switch -regexp $numbers(tens) {
                                      {00}        { }
                                      {[0][1-9]}  {lappend prompt "[set prefix]_[lindex $separate 1].au"}
                                      default     {lappend prompt "[set prefix]_$numbers(tens).au"}
                                   }
                        }
           }
         }
      }
      return 0
}
```

Proc do_hundreds

This new procedure is used to build the language dynamic prompt for the value stored in the numbers (hundreds) array index. In this procedure, the language exception code is written if the language handles the hundreds numbers different than the default method, as shown in the following example.

Example 4 Example of Procedure do_hundreds

```
proc do_hundreds {} {
    global prompt
    global prefix
    global numbers

    if {$numbers(hundreds) == 0} {
          switch -regexp $prefix {
             {ca} { if {$numbers(tens) >= 10 } {
                         lappend prompt "[set prefix]_0.au"
                     } else {
                         return 0
                     }
            }
             default {return 0}
         }
    } else {
          switch -regexp $prefix {
             {ru} {lappend prompt "[set prefix]_$numbers(hundreds)00.au"}
             {ja} {switch -regexp $numbers(hundreds) {
                        1 {lappend prompt "[set prefix]_hyaku.au"}
                        3 {lappend prompt "[set prefix]_san_byaku.au"}
                        6 {lappend prompt "[set prefix]_ro_pyaku.au"}
                        8 {lappend prompt "[set prefix]_happyaku.au"}
                        default {lappend prompt "[set prefix]_$numbers(hundreds).au" "[set
prefix]_hyaku.au"}
                     }
            }
           default {lappend prompt "[set prefix]_$numbers(hundreds).au" "[set prefix]_hundred.au"}
        }
    }

    return 0
}
```

Proc do_thousands

This new procedure is used to build the language dynamic prompt for the value stored in the numbers (thousands) array index. In this procedure, the language exception code is written if that language handles the thousands numbers different than the default method. The language form explained in Table 1 on page 3 has an effect on how the language is implemented in this procedure. The procedure has to know if the numbers (thousands) array index contains a one- or two-digit number.

Example 5 Example of Procedure do_thousands

```
proc do_thousands {} {
    global prompt
    global prefix
    global numbers
    global len

    if {$numbers(thousands) == 0} {
            switch -regexp $prefix {
               {ca} {if {($numbers(hundreds) == 0) && ($numbers(tens) >= 10) } {
                      lappend prompt "[set prefix]_0.au"
                      }
               }
               {th} -
               {ja}    { }
               default {lappend prompt "[set prefix]_thousand.au"}
         }
    } else {
         switch -regexp $prefix {
            {th} {if {$len == 4} {
                          lappend prompt "[set prefix]_$numbers(thousands).au" "[set prefix]_thousand.au"
                    } else {
                          set separate [split $numbers(thousands) ""]
                          switch -regexp $numbers(thousands) {
                             {00}        { }
                             {[0][1-9]}  {lappend prompt "[set prefix]_[lindex $separate 1].au" "[set 
prefix]_thousand.au"}
                             {[1-9][0]}  {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set 
prefix]_ten_thousand.au"}
                             default     {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set 
prefix]_ten_thousand.au" "[set prefix]_[lindex $separate 1].au" "[set prefix]_thousand.au"}
                          }
                    }
            }
            {ja} {switch -regexp $numbers(thousands) {
                      1 {lappend prompt "[set prefix]_thousand.au"}
                      3 {lappend prompt "[set prefix]_san_zen.au"}
                      8 {lappend prompt "[set prefix]_ha_ssen.au"}
                      default {lappend prompt "[set prefix]_$numbers(thousands).au" "[set 
prefix]_thousand.au"}
                    }
            }
            default {lappend prompt "[set prefix]_$numbers(thousands).au" "[set prefix]_thousand.au"}
        }
    }

    return 0
}
```

Proc do_hthousands

This new procedure is used to build the language dynamic prompt for the value stored in the numbers (hthousands) array index. In this procedure, the language exception code is written if that language handles the hundred thousand numbers different than the default method. The language form explained in Table 1 on page 3 has an effect on how the language is implemented in this procedure. The procedure has to know if the numbers (hthousands) array index contains a one- or two-digit number.

Example 6 Example of Procedure do_hthousands

```
proc do_hthousands {} {
    global prompt
    global prefix
    global numbers

    set hlen [string length $numbers(hthousands)]
    switch -regexp $prefix {
       {th}  {lappend prompt "[set prefix]_$numbers(hthousands).au" "[set prefix]_hundred_thousand.au"}
       {ca}  {if {$hlen == 1} {
                    lappend prompt "[set prefix]_$numbers(hthousands).au" "[set prefix]_ten_thousand.au"
               } else {set separate [split $numbers(hthousands) ""]
                         switch -regexp $numbers(hthousands) {
                            {[1][1-9]}  {lappend prompt "[set prefix]_10.au" "[set prefix]_[lindex
$separate 1].au" "[set prefix]_ten_thousand.au"}
                            {[2-9][0]}  {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set
prefix]_10.au" "[set prefix]_ten_thousand.au"}
                            default     {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set
prefix]_10.au" "[set prefix]_[lindex $separate 1].au" "[set prefix]_ten_thousand.au"}
                         }
             }
       }
       {ja} {switch $numbers(hthousands) {
                  1        {lappend prompt "[set prefix]_ichi_man.au"}
                  default  {lappend prompt "[set prefix]_$numbers(hthousands).au" "[set prefix]_man.au"}
               }
       }
       {ru} {lappend prompt "[set prefix]_$numbers(hthousands)00.au"}
       default {lappend prompt "[set prefix]_$numbers(hthousands).au" "[set prefix]_hundred.au"}
    }

    return 0
}
```

Proc do_decimal_part

This new procedure separates the decimal number string into its numeric prompt components following the languages rules for tens. This is explained in more detail later in this section.

Example 7 Example of Procedure do_decimal_part

```
proc do_decimal_part {} {
   global newlist
   global prompt
   global prefix
   global gender

# Do the decimal translation

    puts "\t\t*** TTS decimal part: [lindex $newlist 1]"

    switch -regexp $prefix {
        {th} {if {[string length [lindex $newlist 1]] == 1} {
               switch -regexp [lindex $newlist 1] {
                  {0}     { }
                  default {lappend prompt "[set prefix]_[lindex $newlist 1]0.au"}
               }
              } else {set separate [split [lindex $newlist 1] ""]
                          switch -regexp [lindex $newlist 1] {
                             {00} { }
               {[0][1-9]} {lappend prompt "[set prefix]_[lindex $separate 1].au"}
               {10}       {lappend prompt "[set prefix]_[lindex $newlist 1].au"}
               {[1][1-9]} {lappend prompt "[set prefix]_10.au" "[set prefix]_[lindex $separate 1].au" }
               {[2][0]}   {lappend prompt "[set prefix]_20.au"}
               {[2][1-9]} {lappend prompt "[set prefix]_20.au" "[set prefix]_[lindex $separate 1].au"}
               {[3-9][0]} {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set prefix]_10.au"}
               default    {lappend prompt "[set prefix]_[lindex $separate 0].au" "[set prefix]_10.au" "[set
prefix]_[lindex $separate 1].au"}
           }
              }
        }
        default {if {[lindex $newlist 0] > 0 && [lindex $newlist 1] > 0} {
                     lappend prompt "[set prefix]_and.au"
                 }
                 if {[string length [lindex $newlist 1]] == 1} {
                     switch -regexp [lindex $newlist 1] {
           {0}     { }
           default {lappend prompt "[set prefix]_[lindex $newlist 1]0.au"}
              }
                } else {set separate [split [lindex $newlist 1] ""]
                         switch -regexp [lindex $newlist 1] {
              {00}       { }
              {[0][1-9]} {lappend prompt "[set prefix]_[lindex $separate 1].au"}
              default    {lappend prompt "[set prefix]_[lindex $newlist 1].au"}
           }
          }
        }
    }
   return 0
}
```

In procedures do_get_dest and do_second_authorization, a TCL switch statement was added to the branch if the selected language is a language exception (refer to Modifying the Existing Script, page 4). If not, the script executes the default case for playPrompt as previously implemented in the debitcard script. If the selected language is a language exception the switch statement in procedure do_get_dest starts the prompt building process.

Example 1 on page 5 shows the switch statement and language exception (ru). The default statement for the switch is the standard execution of playPrompt from previous debitcard applications. The switch statement is used in the else part of nested if for amount. It is implemented if the creditAmt returned from the billing server is greater than 0 and less than or equal to the maximum amount allowed (999,999.99).

Following the Flow

- In Example 1, the exception code shown in the first statement splits the value for creditAmt ($amt) returned from the billing server into two parts, a whole part and a decimal part. After the split function, we are left with a two-element list. The string contained in list element 0 represents the whole part of the $amt variable, and the string contained in list element 1 represents the decimal portion.
- Next, as shown in Example 2 on page 6, we start the dynamic prompt building process with a procedure call to do_whole_part. The do_whole_part procedure makes a determination on the size and characteristic of the number being passed. The string representing the whole part of the creditAmt can be said to have a four-part characteristic. based on its length and characters representing a tens, a hundreds, a thousands, and a hundred thousands place.

  Depending on the language form, a dynamic prompt for a language can be shown as a 6-digit whole number of the form *SDDHTT*, where *S* represents the hundred thousands position, *DD* represents the thousands position, *H* represents the hundreds position and *TT* represent the tens position. Another form of dynamic prompt for a language also can be shown as a 6-digit whole number of the form *SSDHTT*, where *S* represents the ten thousands position, *D* represents the thousands position, *H* represents the hundreds position, and *TT* represents the tens position.
- Example 4 on page 8 shows where the switch builds the numbers array differently based on a string length of 5 or 6 (refer to Table 1 on page 3 regarding the difference of language exception number translation). In this example, the default case builds the numbers array around the *SDDHTT* form, while the exceptions are built for Cantonese (ca) and Japanese (ja), where the numbers array uses the form *SSDHTT.*
- When the whole part number form is established, a value is entered for the numbers array indexes. If a value is stored in the numbers index for tens, hundreds, thousands, or hthousands, the procedure is called to build the dynamic prompt. Example 3 on page 7 shows one way to handle the *TT* numbers using the procedure do_tens. In this example, the exception is for Thai (th). In the procedure the default case appends to the prompt list the language prefix and the value stored in numbers (tens) with some exceptions for string length. For Thai (th) there are exceptions for several different string combinations. The procedures do_hundreds, do_thousands, and do_hthousands are similar in design.

  After the procedures do_tens, do_hundreds, do_thousands, and do_hthousands append the number files to build the dynamic prompt the call flow returns to the do_get_dest switch, where the script appends the language's currency denomination.
- The flow then proceeds to build the dynamic prompt for the decimal part. The do_decimal_part procedure shown in Example 7 on page 11, follows most of the same rules as do_tens procedure for most languages; but in some cases, there are different word gender rules. There are also some differences in handling the split decimal part strings. The string returned from the billing server for decimal part for numbers of ".10", ".20", ".30", and so on, is returned as .1, .2 and .3, and so on. Some special handling is needed when you append the necessary prompt. You can see this in the switch where string lengths are 1.

Example 8 Example of Procedure do_second_authorization Modifications

```
switch -regexp $ev {
        {authorized} {if {[string compare $creditTime uninitialized] == 0} {
                          set ev [ playPrompt param2 info [set prefix]_no_aaa.au]
                          set state end
                      } elseif {$creditTime == "umlimited"} {
                          set noTimeLimit 1
                          # play mesg only if time left is  20 secs > warntime
                      } elseif {[expr $creditTime - $warnTime] < 20} {
                          set noPlay 1
                          switch -regexp $prefix {
                 {th} -
                 {ca} -
                 {ru} -
                 {ja} { # Convert creditTime to hours and minutes
                                     # Build prompt and play creditTime

                                     do_creditTime_prompt

             }
                 default {set ev [playPrompt param2 info [set prefix]_you_have.au %t$creditTime ]}
             }
               } else {
                          switch -regexp $prefix {
                 {th} -
                 {ca} -
                 {ru} -
                 {ja} { # Convert creditTime to hours and minutes
                                     # Build prompt and play creditTime

                                     do_creditTime_prompt

             }
                 default {set ev [playPrompt param2 info [set prefix]_you_have.au %t$creditTime ]}
             }
               }
                      set state do_place_call
                      return 0
        }
```

After the procedure do_decimal_part appends the number files to build the decimal number portion of the dynamic prompt, the call flow returns and the script appends the language's currency denomination for the decimal currency.

The billing server value for creditTime is handled in the procedure do_second_authorization, see Example 8 on page 13. The language modification is simply a call to the procedure do_creditTime_prompt. This was made a separate procedure to minimize lines of code. The procedure do_second_authorization had two places where almost identical operations were performed, so a separate procedure was created to handle the operations.

Proc do_creditTime_prompt

This new procedure is used after successful second authorization. It converts the billing server returned creditTime value of seconds to a two-element list whose whole part is hours and whose decimal part is minutes. The procedure follows the same process described previously for building the language dynamic prompts.

Example 9 Example of Procedure do_creditTime_prompt

```
proc do_creditTime_prompt {} {
    global creditTime
    global prefix
    global param2
    global len
    global prompt
    global newlist
    global numbers
    global gender

 # Convert creditTime to hours and minutes

       set prompt ""
       set newlist [list [expr {int([expr {$creditTime / 3600}])}] [expr {int([expr {$creditTime % 3600} /
60])}] ]

# Gender is used to determine correct word structure in Russian prompt building

       set gender m

       do_whole_part

       switch $prefix {
          {ru} {if {$len == 1 } {
                     switch -regexp $numbers(tens) {
           {0} { }
           {1} {lappend prompt "[set prefix]_hour.au"}
           {[2-4]} {lappend prompt "[set prefix]_2-4_hours.au"}
           {[5-9]} {lappend prompt "[set prefix]_5-20_hours.au"}
             }
               } else {
                     switch -regexp $numbers(tens) {
           {[0-9]*[0|2-9][2-4]} {lappend prompt "[set prefix]_2-4_hours.au"}
           {[0-9]*[2-9][1]}     {lappend prompt "[set prefix]_hour.au"}
                 default              {lappend prompt "[set prefix]_5-20_hours.au"}
             }
          }
          }
          default {if {[lindex $newlist 0] == 1 } {
                         lappend prompt "[set prefix]_hour.au"
                   } elseif {[lindex $newlist 0] > 1 } {
                         lappend prompt "[set prefix]_hours.au"
                   }
          }
       }
```

Example 10 Example of Procedure do_creditTime_prompt (continued)

```
set list_index_1 [lindex $newlist 1]

  # Do the decimal translation to build prompt for minutes

      switch -regexp $prefix {
        {ru} {set gender f

              do_decimal_part

              if {[string length [lindex $newlist 1]] == 1} {
                  switch -regexp [lindex $newlist 1] {
            {0}     { }
            default {lappend prompt "[set prefix]_5-20_minutes.au"}
               }
           } else {
               switch -regexp [lindex $newlist 1] {
            {00}           { }
            {[0|2-9][1]}   {lappend prompt "[set prefix]_minute.au"}
            {[0|2-9][2-4]} {lappend prompt "[set prefix]_2-4_minutes.au"}
            default        {lappend prompt "[set prefix]_5-20_minutes.au"}
               }
           }
                 puts "\t\t*** playPrompt param2 info [set prefix]_you_have.au $prompt"
                 set ev [eval [list playPrompt param2 info [set prefix]_you_have.au] $prompt]
           }
           default {do_decimal_part

                    if {[lindex $newlist 1] == 1 } {
                       lappend prompt "[set prefix]_minute.au"
                     } elseif {[lindex $newlist 1] > 1 } {
                       lappend prompt "[set prefix]_minutes.au"
                    }
                    puts "\t\t*** playPrompt param2 info [set prefix]_you_have.au $prompt"
                    set ev [eval [list playPrompt param2 info [set prefix]_you_have.au] $prompt]
            }
       }
}
```

Cisco Connection Online

Cisco Connection Online (CCO) is Cisco Systems' primary, real-time support channel. Maintenance customers and partners can self-register on CCO to obtain additional information and services.

Available 24 hours a day, 7 days a week, CCO provides a wealth of standard and value-added services to Cisco's customers and business partners. CCO services include product information, product documentation, software updates, release notes, technical tips, the Bug Navigator, configuration notes, brochures, descriptions of service offerings, and download access to public and authorized files.

CCO serves a wide variety of users through two interfaces that are updated and enhanced simultaneously: a character-based version and a multimedia version that resides on the World Wide Web (WWW). The character-based CCO supports Zmodem, Kermit, Xmodem, FTP, and Internet e-mail, and it is excellent for quick access to information over lower bandwidths. The WWW version of CCO provides richly formatted documents with photographs, figures, graphics, and video, as well as hyperlinks to related information.

You can access CCO in the following ways:

- WWW: http://www.cisco.com
- WWW: http://www-europe.cisco.com
- WWW: http://www-china.cisco.com
- Telnet: cco.cisco.com
- Modem: From North America, 408 526-8070; from Europe, 33 1 64 46 40 82. Use the following terminal settings: VT100 emulation; databits: 8; parity: none; stop bits: 1; and connection rates up to 28.8 kbps.

For a copy of CCO's Frequently Asked Questions (FAQ), contact cco-help@cisco.com. For additional information, contact cco-team@cisco.com.

Note If you are a network administrator and need personal technical assistance with a Cisco product that is under warranty or covered by a maintenance contract, contact Cisco's Technical Assistance Center (TAC) at 800 553-2447, 408 526-7209, or tac@cisco.com. To obtain general information about Cisco Systems, Cisco products, or upgrades, contact 800 553-6387, 408 526-7208, or cs-rep@cisco.com.

Documentation CD-ROM

Cisco documentation and additional literature are available in a CD-ROM package, which ships with your product. The Documentation CD-ROM, a member of the Cisco Connection Family, is updated monthly. Therefore, it might be more current than printed documentation. To order additional copies of the Documentation CD-ROM, contact your local sales representative or call customer service. The CD-ROM package is available as a single package or as an annual subscription. You can also access Cisco documentation on the World Wide Web at http://www.cisco.com, http://www-china.cisco.com, or http://www-europe.cisco.com.

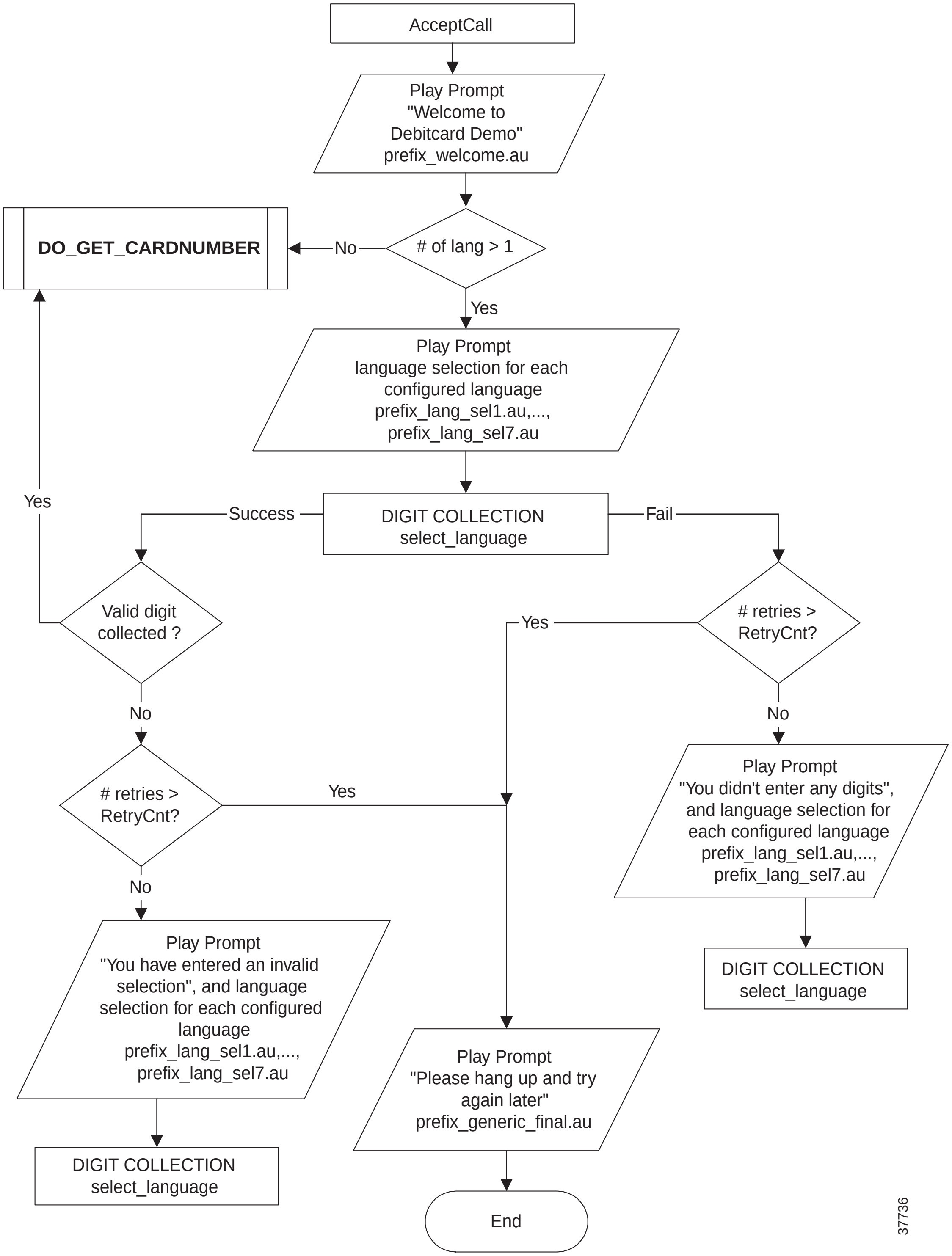
If you are reading Cisco product documentation on the World Wide Web, you can submit comments electronically. Click **Feedback** in the toolbar and select **Documentation**. After you complete the form, click **Submit** to send it to Cisco. We appreciate your comments.

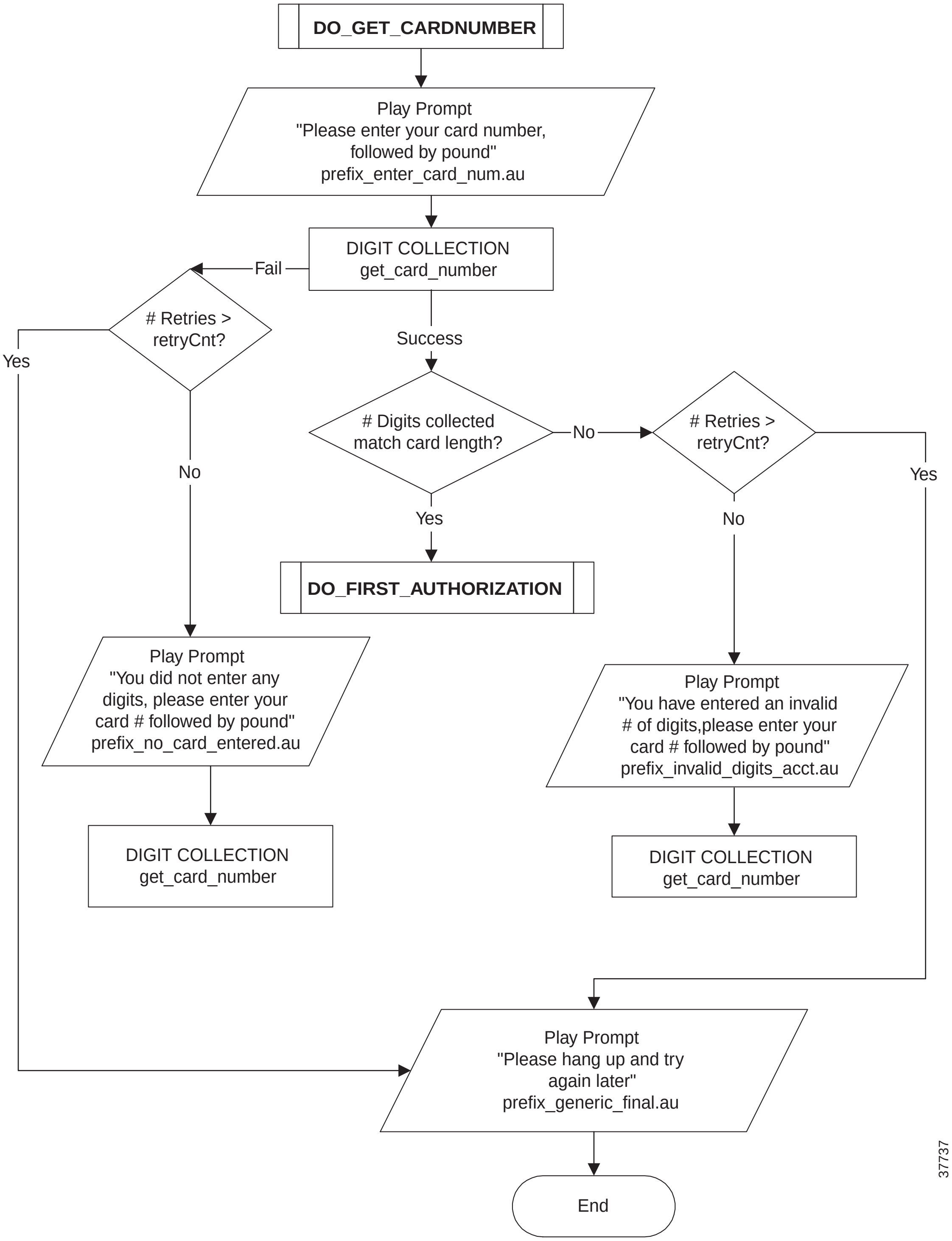
Glossary

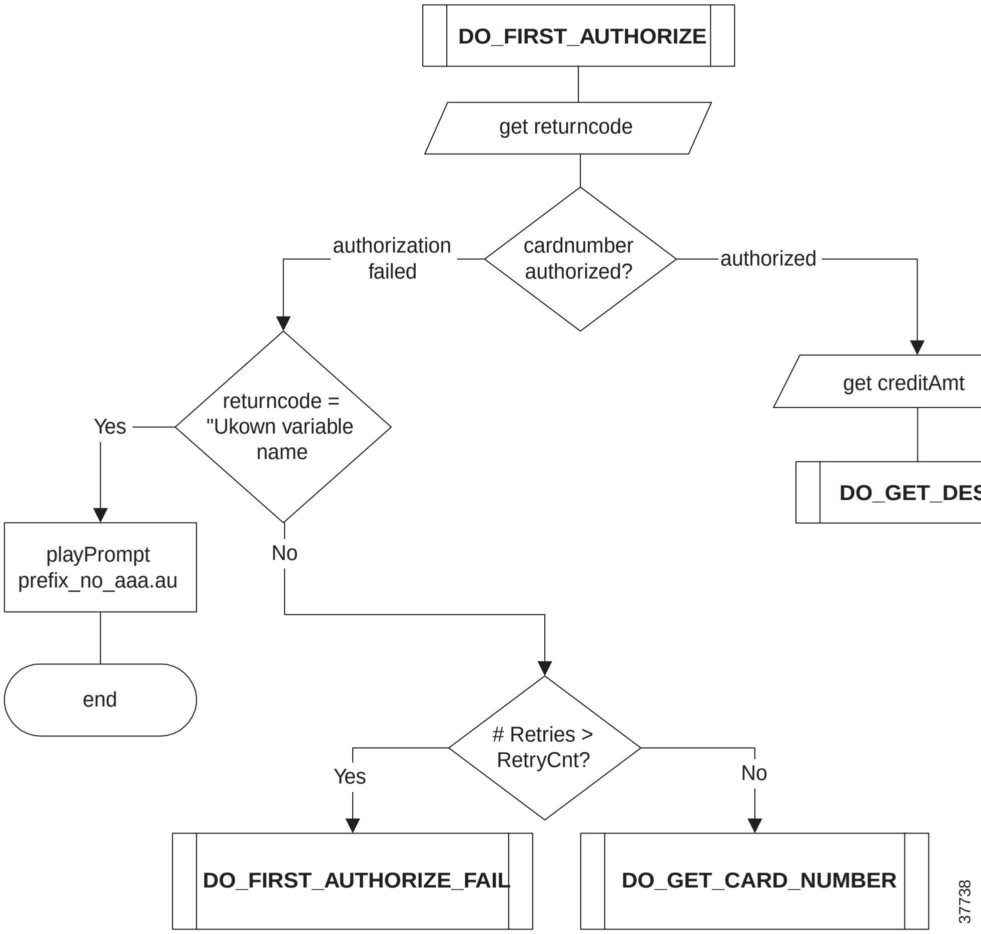
For a list of other internetworking terms, see *Internetworking Terms and Acronyms* document that is available on the Documentation CD-ROM and Cisco Connection Online (CCO) at the following URL: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ita/index.htm.

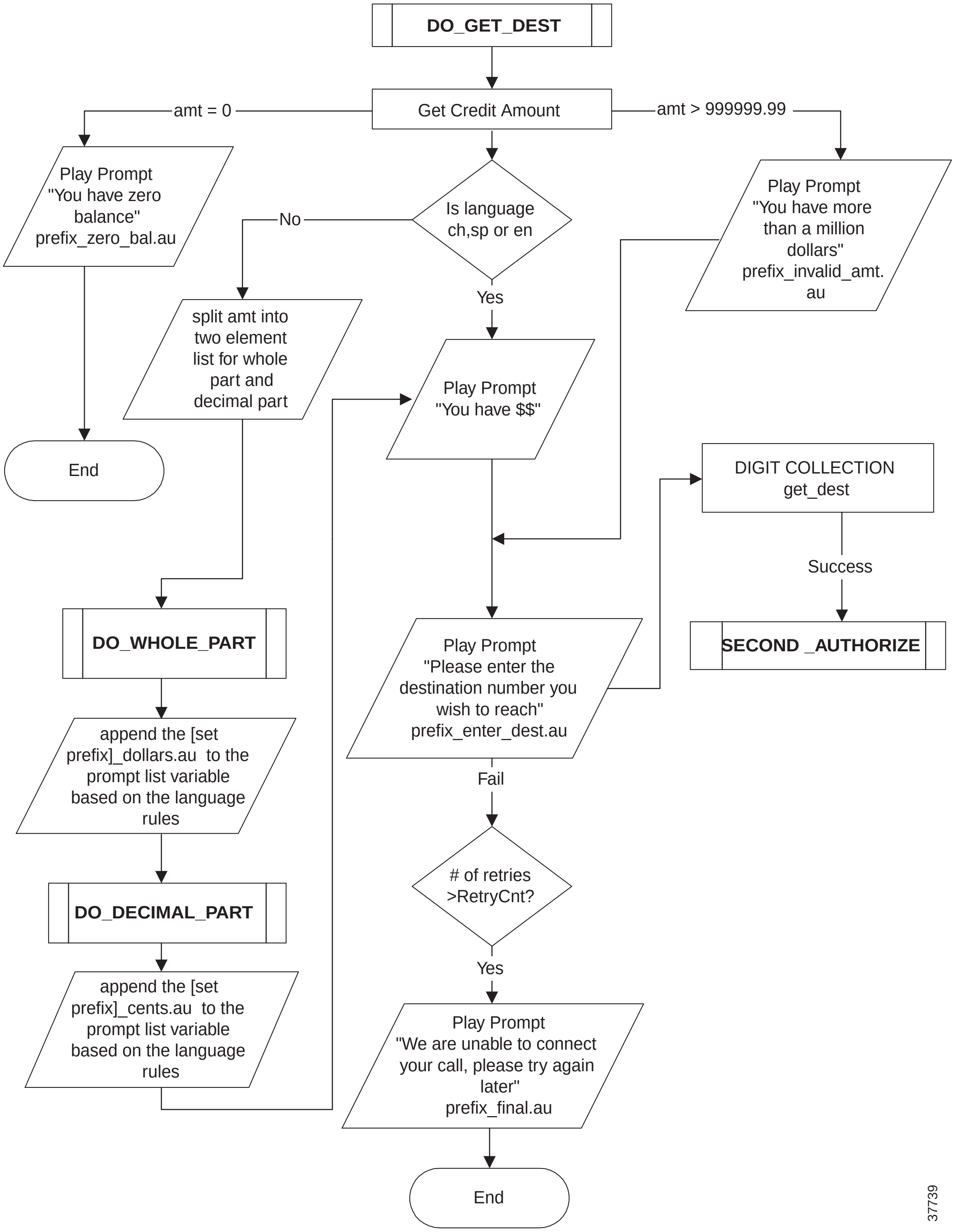
Call Flow Example

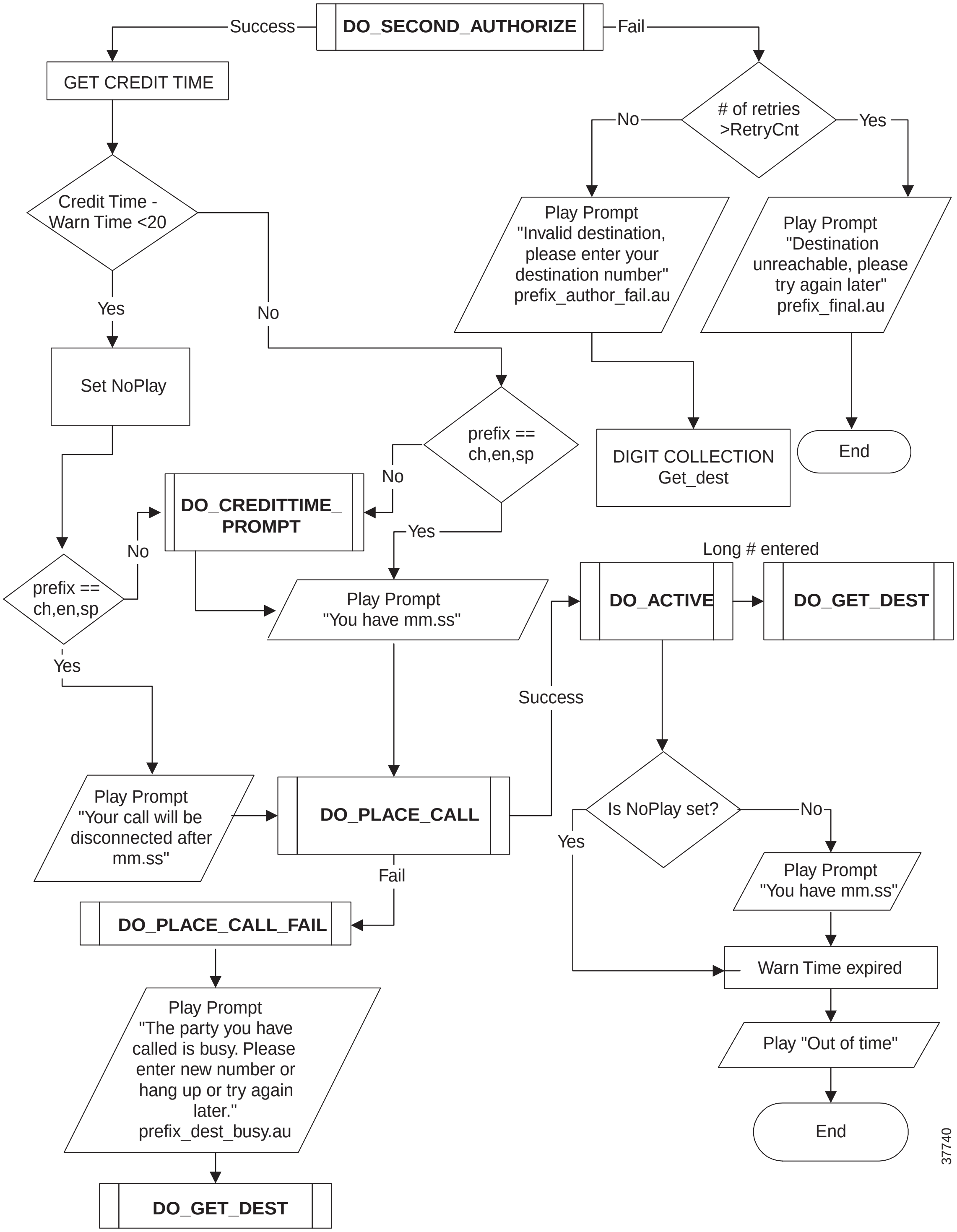
Following is an example of the call flow for this feature:

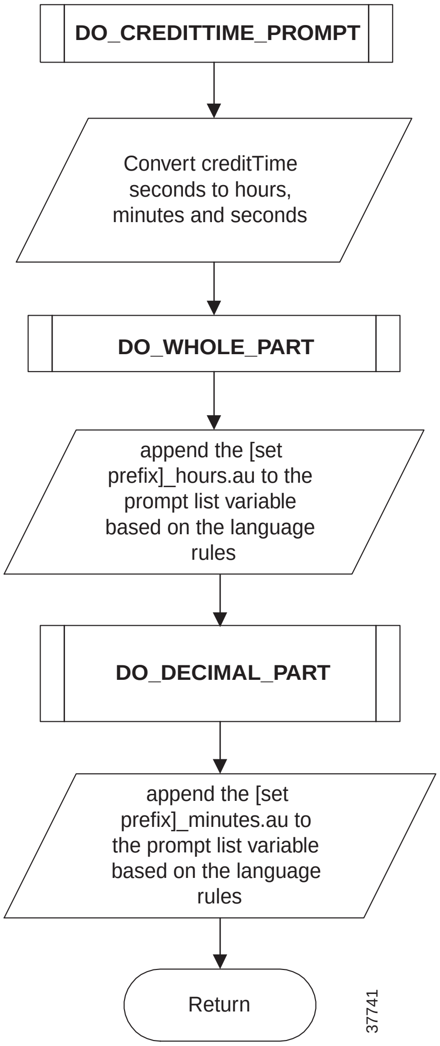

AcceptCall
Play Prompt "Welcome to Debitcard Demo" prefix_welcome.au
# of lang > 1
No
DO_GET_CARDNUMBER
Yes
Play Prompt language selection for each configured language prefix_lang_sel1.au,..., prefix_lang_sel7.au
DIGIT COLLECTION select_language
Success
Fail
Yes
Valid digit collected ?
# retries > RetryCnt?
Yes
No
No
# retries > RetryCnt?
Yes
No
Play Prompt "You didn't enter any digits", and language selection for each configured language prefix_lang_sel1.au,..., prefix_lang_sel7.au
DIGIT COLLECTION select_language
Play Prompt "You have entered an invalid selection", and language selection for each configured language prefix_lang_sel1.au,..., prefix_lang_sel7.au
DIGIT COLLECTION select_language
Play Prompt "Please hang up and try again later" prefix_generic_final.au
End
37736

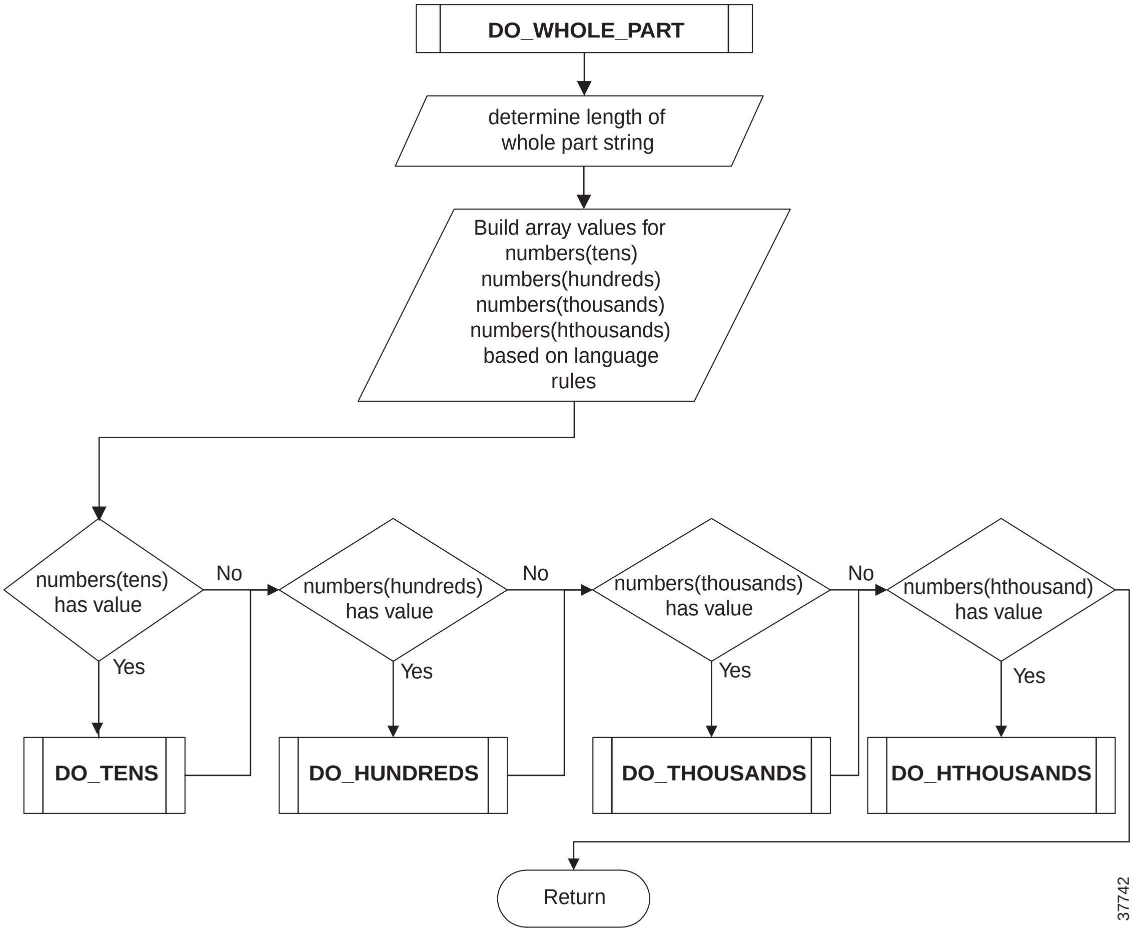

DO_GET_CARDNUMBER
Play Prompt "Please enter your card number, followed by pound" prefix_enter_card_num.au
DIGIT COLLECTION get_card_number
Fail
# Retries > retryCnt?
Yes
No
Success
# Digits collected match card length?
No
# Retries > retryCnt?
Yes
Yes
No
DO_FIRST_AUTHORIZATION
Play Prompt "You did not enter any digits, please enter your card # followed by pound" prefix_no_card_entered.au
Play Prompt "You have entered an invalid # of digits,please enter your card # followed by pound" prefix_invalid_digits_acct.au
DIGIT COLLECTION get_card_number
DIGIT COLLECTION get_card_number
Play Prompt "Please hang up and try again later" prefix_generic_final.au
End
37737

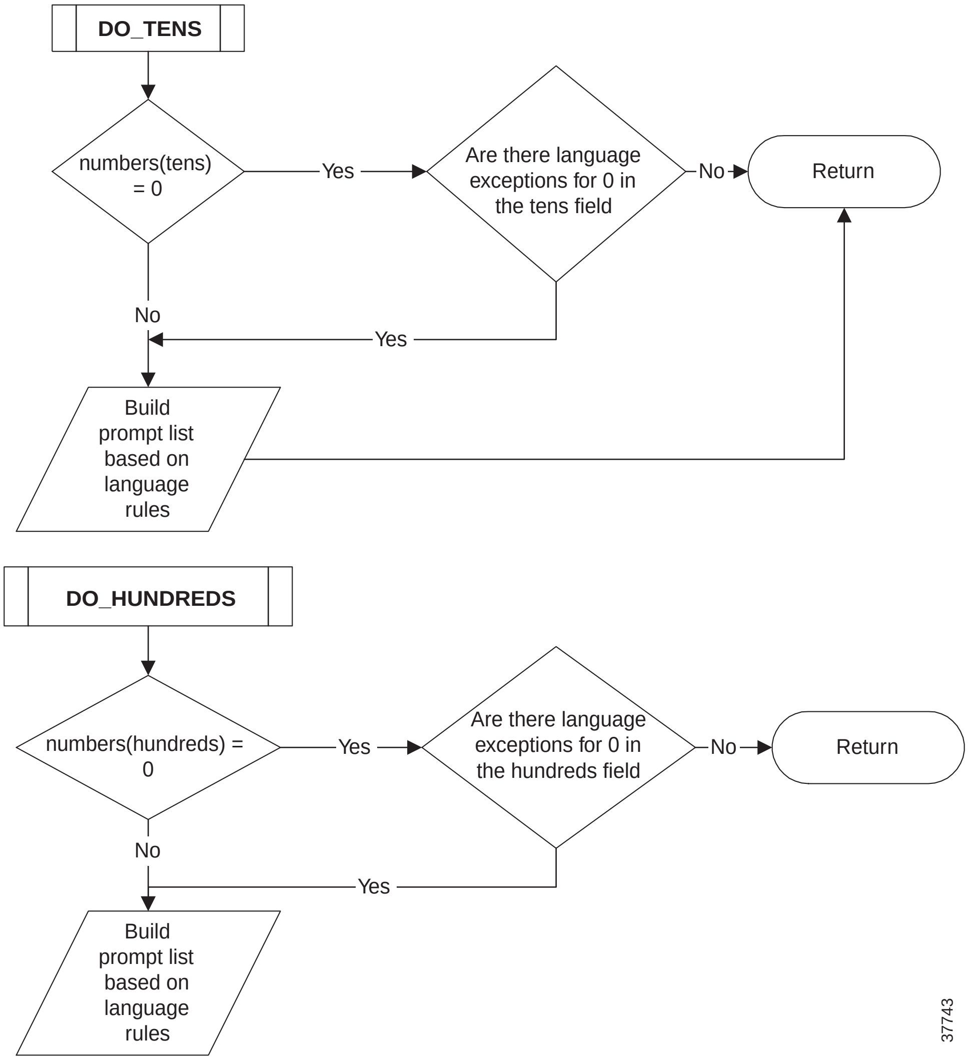

DO_FIRST_AUTHORIZE
get returncode
cardnumber authorized?
authorization failed
authorized
get creditAmt
DO_GET_DEST
returncode = "Ukown variable name
Yes
No
playPrompt prefix_no_aaa.au
end
# Retries > RetryCnt?
Yes
No
DO_FIRST_AUTHORIZE_FAIL
DO_GET_CARD_NUMBER
37738

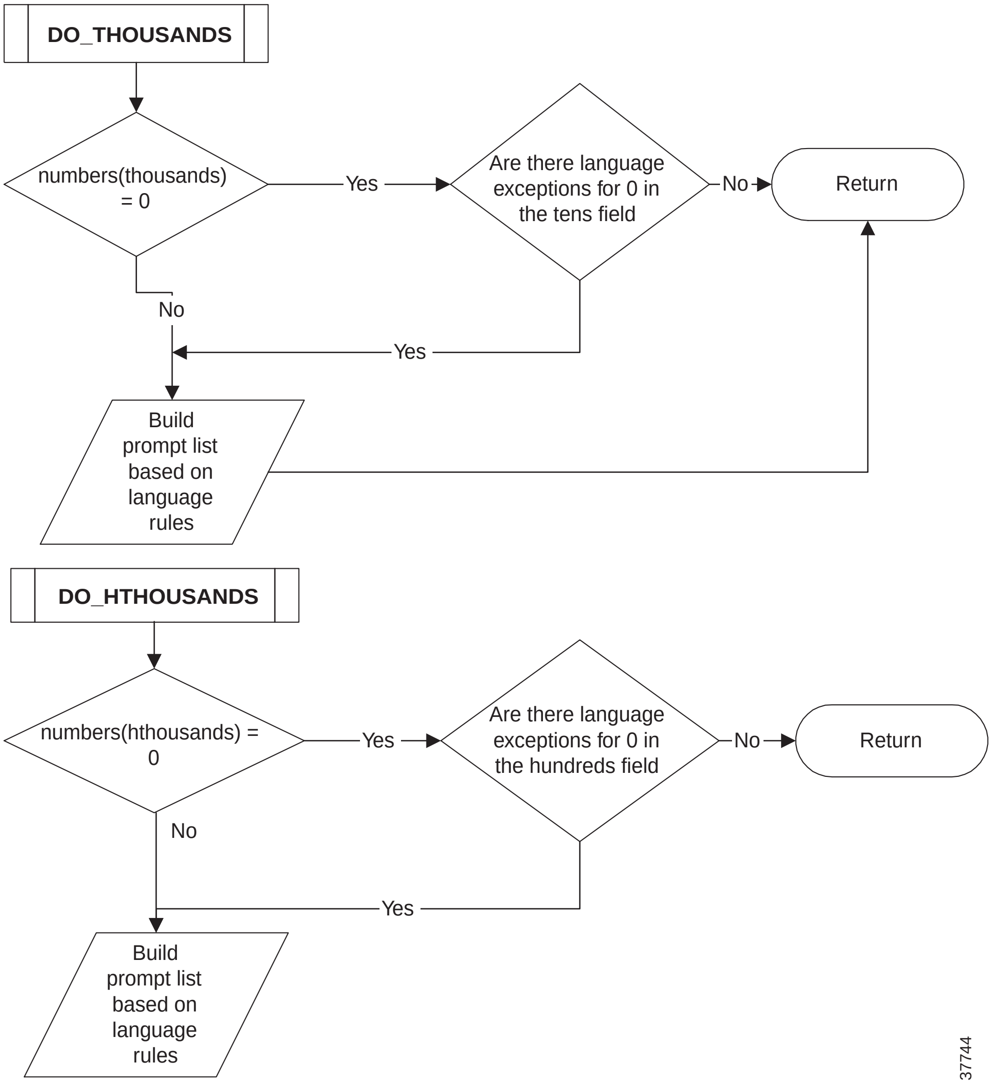

DO_GET_DEST
Get Credit Amount
amt = 0
amt > 999999.99
Play Prompt "You have zero balance" prefix_zero_bal.au
End
Is language ch,sp or en
No
Yes
split amt into two element list for whole part and decimal part
Play Prompt "You have more than a million dollars" prefix_invalid_amt. au
Play Prompt "You have $$"
DIGIT COLLECTION get_dest
Success
SECOND _AUTHORIZE
DO_WHOLE_PART
append the [set prefix]_dollars.au to the prompt list variable based on the language rules
Play Prompt "Please enter the destination number you wish to reach" prefix_enter_dest.au
Fail
DO_DECIMAL_PART
# of retries >RetryCnt?
Yes
append the [set prefix]_cents.au to the prompt list variable based on the language rules
Play Prompt "We are unable to connect your call, please try again later" prefix_final.au
End
37739


DO_SECOND_AUTHORIZE
Success
Fail
GET CREDIT TIME
# of retries >RetryCnt
No
Yes
Credit Time - Warn Time <20
Play Prompt "Invalid destination, please enter your destination number" prefix_author_fail.au
Play Prompt "Destination unreachable, please try again later" prefix_final.au
Yes
No
Set NoPlay
prefix == ch,en,sp
DIGIT COLLECTION Get_dest
End
No
DO_CREDITTIME_ PROMPT
Yes
No
Long # entered
prefix == ch,en,sp
Play Prompt "You have mm.ss"
DO_ACTIVE
DO_GET_DEST
Yes
Success
Play Prompt "Your call will be disconnected after mm.ss"
DO_PLACE_CALL
Is NoPlay set?
No
Yes
Fail
Play Prompt "You have mm.ss"
DO_PLACE_CALL_FAIL
Warn Time expired
Play Prompt "The party you have called is busy. Please enter new number or hang up or try again later." prefix_dest_busy.au
Play "Out of time"
End
DO_GET_DEST
37740


DO_CREDITTIME_PROMPT
Convert creditTime seconds to hours, minutes and seconds
DO_WHOLE_PART
append the [set prefix]_hours.au to the prompt list variable based on the language rules
DO_DECIMAL_PART
append the [set prefix]_minutes.au to the prompt list variable based on the language rules
Return
37741


DO_WHOLE_PART
determine length of whole part string
Build array values for numbers(tens) numbers(hundreds) numbers(thousands) numbers(hthousands) based on language rules
numbers(tens) has value
No
numbers(hundreds) has value
No
numbers(thousands) has value
No
numbers(hthousand) has value
Yes
Yes
Yes
Yes
DO_TENS
DO_HUNDREDS
DO_THOUSANDS
DO_HTHOUSANDS
Return
37742


DO_TENS
numbers(tens) = 0
Yes
Are there language exceptions for 0 in the tens field
No
Return
No
Yes
Build prompt list based on language rules
DO_HUNDREDS
numbers(hundreds) = 0
Yes
Are there language exceptions for 0 in the hundreds field
No
Return
No
Yes
Build prompt list based on language rules
37743


DO_THOUSANDS
numbers(thousands) = 0
Yes
Are there language exceptions for 0 in the tens field
No
Return
No
Yes
Build prompt list based on language rules
DO_HTHOUSANDS
numbers(hthousands) = 0
Yes
Are there language exceptions for 0 in the hundreds field
No
Return
No
Yes
Build prompt list based on language rules
37744