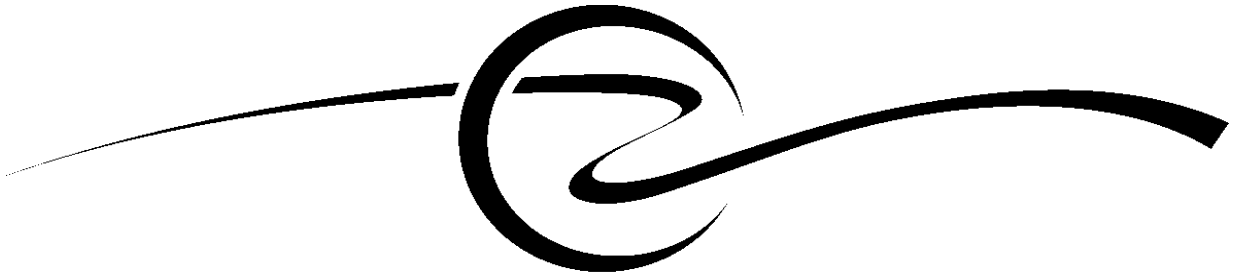


OPENPORT™
technology

ADDRESS PARSING IN IP LAUNCHPAD 1.4

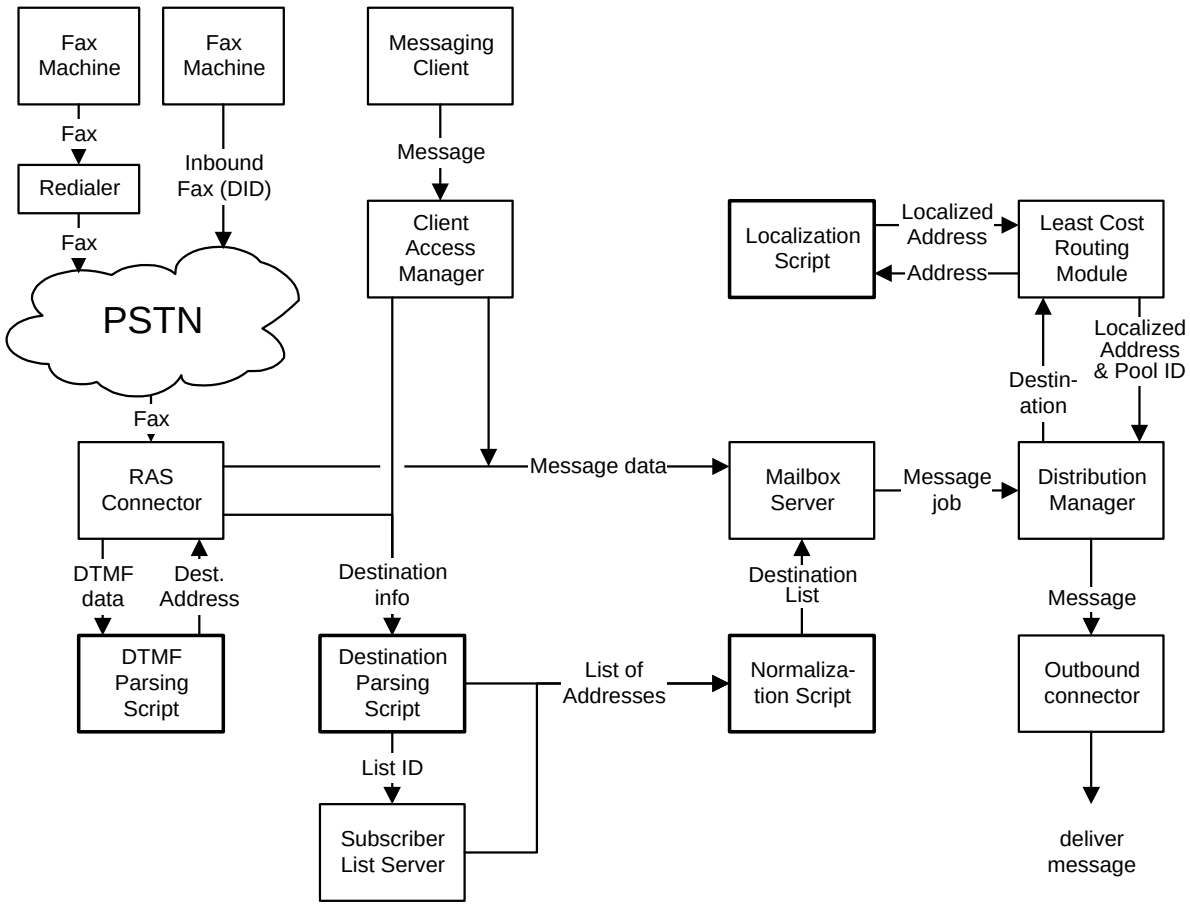
Version 2

Part Number D5DTM140-02

Parsing Narratives

Introduction to Narratives

The following diagram highlights the role of the IP LaunchPad Address Parsing scripts in the transmission of inbound and outbound messages.-



This diagram does not show all the modules and exchanges that are involved in an inbound or outbound message transmission. This depiction of the process has been simplified to emphasize the roles of the Address Parsing scripts.

One or more of the Tcl scripts documented in this Guide are called in the following parts of IP LaunchPad message processing:

A: Submitting a Message from a Messaging Client

B: Submitting a Message from a Fax Machine

C: Processing the Address String

D: Processing a Broadcast List

E: Normalizing Message Destinations

F: Processing and Delivering Messages

G: Receiving an Inbound Message from Off-Net

The following sections detail the steps involved in each of these processes.

A: Submitting a Message from a Messaging Client

1. The Subscriber uses a messaging client program to create a message (a fax, an EMail, a voice message, etc.). A message consists of two components:
   - The body of the message.
   - A list of one or more destination addresses. At this point in the process, the destination list can include one or more simple destination addresses (fax numbers, EMail addresses, etc.) and/or one or more List IDs (the IDs of mailing lists created by this Subscriber using the IP LaunchPad Subscriber List Management application).

   The message is submitted to a Client Access Manager (CAM) within the IP LaunchPad system.
2. After the CAM authenticates the message sender, it submits the actual message content to the local Mailbox Server (MBS). At the same time, the CAM passes the destination list to the Destination Parsing script.

Go to *C: Processing the Address String* on page 13 for the next steps.

B: Submitting a Message through a Redialer

1. The Subscriber puts a document in the fax machine, enters one or more addresses (and/or one or more List IDs) and presses the Start button.
2. The fax machine communicates with the Redialer, which intercepts the address string and dials the number for an IP LaunchPad Remote Access Server (RAS). The Redialer sends a series of DTMF tones to the RAS. This series contains the address string from the fax machine plus the Redialer's own identification information. The RAS converts these tones to a character string, which it passes to an IP LaunchPad RAS Connector.
3. The RAS connector passes the address/ID string to the DTMF Parsing script. This script extracts the following information from the string:
   - The address string
   - Redialier identification information (RedialerID, UserID, password, realm, etc.–what information gets passed depends on the capabilities of the Redialer) used by IP LaunchPad to authenticate the Redialer.
4. The RAS connector passes the address string to the Destination Parsing script for further processing.
5. The RAS Connector uses the ID information to authenticate the Redialer. Once the Redialer has been authenticated, the RAS connector receives the fax transmission from the Fax Machine via the Redialer.

Go to *C: Processing the Address String* on page 13 for the next steps.

C: Processing the Address String

1. The Destination Parsing Script assigns an address type to each address in the list.
2. The addresses of type "harmony/broadcast" (these are Subscriber-created mailing lists) are forwarded to the Subscriber List Server (SLS) (see *D: Processing a Broadcast List* on page 14).
3. Addresses of other types are compiled into a list. then the list is passed to the Normalization script. Go to *E: Normalizing Message Destinations* on page 14 for the next steps.

D: Processing a Broadcast List

1. When the SLS receives a list ID from the Destination Parsing script, the server retrieves the list from the User Server (US). The Broadcast List has the same format as the list compiled by the Destination Parsing script: each list item consists of an address and an address type.
2. The SLS passes each name in the Subscriber List to the Normalization script for further processing.

Go to *E: Normalizing Message Destinations* on page 14 for the next steps.

E: Normalizing Message Destinations

1. The Normalization script takes each address in the list and converts it to the standardized format IP LaunchPad uses for that address type. (IP LaunchPad uses the E.164 standard for fax numbers and voice phone numbers.) The output of the Normalization script is the same list of address/type pairs, except that each address has been "normalized."
2. The Normalization script passes the list of addresses to the MBS. At this point, the MBS has the message content and the complete list of destinations for the message.

Go to *F. Processing and Delivering Messages* on page 14 for the next steps.

F. Processing and Delivering Messages

1. For each destination in the list, the MBS creates a message job. One by one, each message job is passed to the Distribution Manager (DM).
2. For each message job, the DM passes the destination address to the Least Cost Routing (LCR) module. The LCR consults a table to find the Pool ID associated with the IP LaunchPad connector can deliver the message as quickly and inexpensively as possible. (In the case of a fax message, for example, this would be a RAS connector that interfaces with the public telephone network at a location close to the destination.)

3. Once the LCR determines where the message is to be delivered from, it sends the destination address to the Localization script. This script uses address characteristics of the designated connector to convert the normalized address to the most appropriate form for local delivery. (Example: the normalized form of any fax number includes the country code and an area code; if IP LaunchPad is going to deliver a message from a connector in the same area code as the destination, the Localization script removes the country code and area code from the destination fax number.)
4. After the LCR has returned the localized phone number and the Pool ID, the DM forwards the message to the appropriate connector for delivery.

G: Receiving an Inbound Message from Off-Net

Each IP LaunchPad Subscriber can be assigned a unique fax number or voice phone number, at which he or she can receive incoming messages. This IP LaunchPad feature is called Direct Inward Dial (DID).

When someone from outside the IP LaunchPad system sends a message to one of these numbers, IP LaunchPad receives the transmission at a local RAS Connector, and routes it by running the destination phone number through the Address Parsing process described below:

1. A non-IP LaunchPad Subscriber sends a message (a fax, in this example) to a destination number defined within the IP LaunchPad system.
2. An IP LaunchPad RAS device receives the destination address.
3. The RAS Connector runs the destination address through the DTMF Parsing script, which tests the destination address to see whether the transmission came from a Redialer. When the destination string fails this test, the script returns a generic username and password which the RAS connector uses to log in to IP LaunchPad.
4. Once the RAS connector has logged in, it receives the fax transmission from the Fax Machine, and passes the message content to the MBS.
5. The RAS connector passes the address string to the Destination Parsing script for further processing.
6. The Destination Parsing script assigns it the default address type "fax-phone/local" to the address, and passes the address string and address type to the Normalization script.

7. The Normalization script converts the address to the standardized format. It sends the address type and normalized address to the MBS.
8. The MBS creates a message job for this message, and passes the job information to the DM.
9. The DM compares the normalized address string to the list of addresses defined for IP LaunchPad Subscribers. When it finds a match, it forwards the message job to the appropriate Subscriber Connector for on-net delivery.

DESTINATION PARSING

The Destination Parsing script receives a list containing one or more destination addresses that have been received from the RAS Connector or Client Access Manager. The script parses each destination address and assigns an Address Type to it. After this, each address/Address Type pair is routed to the Subscriber List Server (if it is a Broadcast List ID) or the Mail Box Server (if it is a fax/EMail/Voice number).

This chapter provides detailed information about the script's operation, and offers some information about customizing the script.

About This Chapter

This Chapter contains the following sections:

The Destination Parsing Script Overview of this script--its name, its inputs, and its outputs.

Printout of OptParseDestination.tcl A printed version of the default Destination Parsing script.

How the Destination Parsing String Works A step-by-step description of how the Destination Parsing script processes a destination address.

Modifying the Script How the Destination Parsing script might be modified.

The Destination Parsing Script

This section provides details about how the Destination Parsing script operates.

The Main Script File

The Main Destination Parsing script is named *OptParseDestination.tcl*. It is stored on any IP LaunchPad server in the directory *SOPT_ROOT/lib/tcl*, where *$OPT_ROOT* is the directory in which the IP LaunchPad software has been installed.

See page 42 for a printout of the default Destination Parsing script.

Inputs

When the Destination parsing script starts, it accesses an array called *OptArgs*. This array has already been populated by the calling program with the following information:

Attribute Name	Value Type	Reqd	Description / Example
destinationStr	String	Yes	The address string as received from the DTMF Parsing script. Example: `5551002`
action	String	Yes	Description of the type of operation to be performed. Example: `parseDestination`
poolname	String	Yes	The name of the pool associated with the RAS Connector that received the transmission. (If the pool name is unknown, a blank string is passed.) Example: `Poo13`
defaultAddrType	String	Yes	The default address type to be used if the script cannot determine the address type. Example: `fax-phone/local`
Areacode	Numeric String	Yes	This is the area code or city code. Example: `312`
CentralOffice	Numeric String	No	Not currently used.

Attribute Name	Value Type	Reqd	Description / Example
InternationalEscape	Numeric String	Yes	The international escape sequence. Example: `011`
CountryCode	Numeric String	Yes	The country code. Example: `1`
NumDigitsLocalCall	Integer	Yes	The number of digits required to make a local call. Example: `7`
NumDigitsLDCall	Integer	Yes	The number of digits required to make a long distance call within this country. Example: 10
LongDistanceEscape	Numeric String	Yes	The sequence that is appended to a long distance phone. Example: `1`
NoCharacteristics	Numeric String	No	Not used in this script.
DestinationScript	String	No	Path and filename of an Override Destination Parsing script defined for this Pool.

Outputs

The IP LaunchPad program that calls the Destination Parsing Script expects the script to output a list of name/value pairs. Since the script returns both an Address string and an Address Type string for each input address, the calling program expects each item in the returned list to conform to the following format:

```
address, addressType
```

Printout of *OptParseDestination.tcl*

> *In the following printout, the line numbers at the left margin are included in this document for reference. They are NOT part of the actual file.*

```
1   # ----------------------------------------------------------------------
2   # OptParseDestination.tcl
3   # ----------------------------------------------------------------------
4
5   proc OPTMain { pOptArgs } {
6   upvar $pOptArgs optArgs
7
8   #puts "OptParseDestination::OPTMain"
9
10  if { ! [ info exists optArgs ] } {
11  error "optArgs does not exist"
12  }
13
14  #puts "optArgs:"
15  #foreach elem [ array names optArgs ] {
16  #puts "   optArgs($elem) = $optArgs($elem)"
17  #}
18
19  if [ catch { set optArgs(destinationStr) } destinationStr ] {
20  error "optArgs(destinationStr) is not defined"
21  }
22
23  if [ catch { set optArgs(defaultAddrType) } defaultAddrType ] {
24  error "optArgs(defaultAddrType) is not defined"
25  }
26
27  #If there is a script associated with this element then invoke it now.
28  if [ info exists optArgs(DestinationScript) ] {
29  if [ catch { source $optArgs(DestinationScript) } result ] {
30  error $result
31  }
32
33  return [ OPTSubMain optArgs ]
34  }
35
36  set addrList [ list ]
```

```
37
38  while { [ string length $destinationStr ] > 0 } {
39  if [ regexp {^0[*]([0-9][0-9]+)[*]?} $destinationStr junk listName ] {
40  lappend addrList "harmony/broadcast"
41  lappend addrList $listName
42  set destinationStr [ string range $destinationStr [ string length $junk ]
                         end ]
43  } elseif [ regexp {^1[*]([0-9][0-9]+)[*]?} $destinationStr junk listName ]
                         {
44  lappend addrList "fax-phone/local"
45  lappend addrList $listName
46  set destinationStr [ string range $destinationStr [ string length $junk ]
                         end ]
47  } else {
48  lappend addrList $defaultAddrType
49  lappend addrList $destinationStr
50  break
51  }
52  }
53
54  return $addrList
55  }
56
```

How the Destination Parsing String Works

The following table summarizes the Destination Parsing script's activity. the line numbers in the first column refer to the line numbers in the printout of the script starting on page 42.

Lines	Activity
5-25	Retrieve values from *OptArgs*; make sure crucial values have been passed.
27-34	If an override script has been passed, execute it.
36	Create the list that will receive the address/addressType pairs.
38	Start of the loop that goes through the input list of addresses.
39-42	If the list item conforms to the format of a broadcast list ID, append it to the new list followed by the AddressType "harmony/broadcast" and remove it from the original list.
43-46	If the list item conforms to the format of a normal phone number, append it to the new list followed by the AddressType "fax-phone/local" and remove it from the original list.
47-52	If the list item fails both of the above tests, append it to the new list followed by the default AddressType and remove it from the original list.
54-55	Return the new list to the calling program.

Modifying the Script

The Destination Parsing script processes every inbound our outbound destination address handled by the IP LaunchPad components for which it is registered. This makes it a powerful instrument for controlling how the system handles addresses. Since the Destination Parsing script is written in Tcl, it can be easily modified as needed. It is possible to modify Destination parsing on a wide scale (by modifying the Main script) or for an individual Pool (by creating an override script).

To Create an Override Destination Parsing Script

1. Log in to any IP LaunchPad server as a System Administrator.
2. Make a copy of the file *$OPT_ROOT/lib/tcl/OptParseDestination.tcl*.
3. Open the copy using *vi* or some other text editor.
4. Customize the copy (see below).
5. Save the under a different name.
6. Register the copy as an override Destination Parsing script in the Hive.

Possible Areas of Customization

By editing the Destination Parsing script, the System Administrator can customize the process of Destination parsing in the following major ways:

Change the Broadcast List Identifier

Line 39 of the default script looks for the string `099` as the identifier for a BroadCast List ID. This line can be adapted to "look for" any identifier.

NOTE: if the messaging system uses Ascend Redialers, the Broadcast List identifier must NOT include an asterisk character (""). This character is reserved for use by the Ascend device.*

Parse for Other Address Types

The generic Destination Parsing script tests each address string for two special address types: "harmony/broadcast" and "fax-phone/local". These tests are performed between lines 38 and 46 of the script. More "testing" lines can be added in this section of the script; this can enable the script to test for other special address format, and assign a particular address type to each one.

Address Exemption

Each of the "testing" lines in the generic Destination Parsing script (for example, line 38) is followed by two lines of code that write the address type and address string to the variable *addrList*. (If the address string conforms to the format specified in line 38, line 39 writes the address type "harmony/broadcast" to *addrList*, and line 40 writes the address string itself. Line 41 removes this address string from *destinationStr*.)

By extension, it is possible to add a line of code that test for a particular format, followed by another line that simply removes the matching address string from *destinationStr*–without adding it to *addrList*. This enables the System administrator to block messages to or from certain addresses (to prevent the transmission of outbound messages to "911", for example, or to block messages to certain area codes or country codes).

The Destination Script Testing Tool

ToolName

OptParseDestinationTest

Usage

Usage:

```
OptParseDestinationTest [-s script-name]
     destination-string default-addr-type
```

Overview

This test harness is used for testing Destination Parsing scripts. For most cases, this is the parsing of dial strings passed into IPLP by redialers.

You will usually change the Destination Passing script when you want to support a new destination parse mechanism for a redialer. It is imperative to test all possible cases in order to ensure the best performance.

Also note that all addresses are run through this script, not just those from redialers. So make sure you code and test for email and voice mail addresses as well.

Arguments

By default, OptParseDestinationTest uses the script */lib/h5/tcl/OptParseDestination.tcl*. If you want to override this behavior, use the `-s` script-name options.

The first argument is the destination address string to parse. This is the destination string as presented from the redialer or other on-ramp client.

The second argument is the default address type. This is supplied by the Connector and is context specific.

Examples

```
/lib/h5/tools/OptParseDestinationTest 5551212
```

This passes the simple destination string `5551212 fax-phone/local` through the default Destination Parse script.

```
/lib/h5/tools/OptParseDestinationTest 0*1234*1*5551212 fax-phone/local
```

This passes the complex destination string `0*1234*1*5551212` through the default Destination Parse script.