Cisco Systems
Empowering the Internet Generation

Cisco Customer Interaction Suite

Cisco ICM Web Option: Trailhead Configuration and Administration Guide

Cisco Trailhead, Version 4.0

Cisco Trailhead, Version 4.0

Table of Contents

Introduction

This document describes the configuration of Cisco Trailhead, a Web server application designed to process Web-initiated requests for live contact. Cisco Trailhead collects Web requests and submits them to the Intelligent Contact Management (ICM) Central Controller, which routes the requests to appropriate destinations.

Intended Audience and Scope

This document is written for system administrators of the Trailhead software. It assumes proper configuration of other products within the configuration.

Additional Information

Several software components comprise the Cisco solution for incorporating Web contact functionality into ICM routing. The following table lists each of these components and where to find more information about them.

Product	**Documentation**
Cisco ICM Web Option	*Cisco ICM Web Option Overview* *Cisco ICM Web Option Implementation Map*
ICM configuration and script set up	*Cisco ICR System Manager Guide* (and the switch supplements for that guide)
Media Blender	*Media Blender Configuration Handbook* *Media Blender Reference guide* Media Blender online Help
Cisco Collaboration Server	*Collaboration Administration and Reporting* Collaboration Agent and Caller online Help Collaboration Administration online Help

The Trailhead Menu

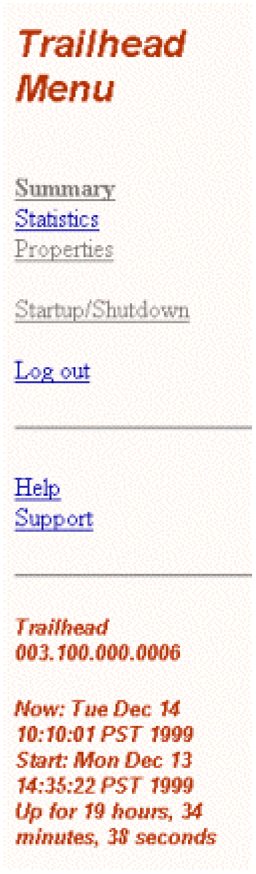
Trailhead provides Web-based administration, allowing the administrator to gather information and troubleshoot problems from a simple Web browser.

1. Open a Web browser and go to this URL:

   `http://<servername>/TrailheadAdmin`.

   The Trailhead Administration login page appears.

2. At the Trailhead login page, enter your username and password and click Log In. The Trailhead Menu appears, as shown below:



Trailhead
003.100.000.0006

Now: Tue Dec 14 10:10:01 PST 1999
Start: Mon Dec 13 14:35:22 PST 1999
Up for 19 hours, 34 minutes, 38 seconds

The Trailhead Menu displays all of the commands you can execute when administering Trailhead. In addition, it displays some constant information in the footer:

- Product name and version
- Current time
- Start time
- Run duration

The Help link provides detailed information about all of the features available on the Control Panel. Refer to the online Help for more information.

System Overview

The Cisco ICM Web Option incorporates the routing of Web-initiated requests with the routing capability of the Cisco Intelligent Contact Management (ICM) software. The ICM product allows enterprises to distribute toll-free telephone calls among centers in different geographic locations. With this integration, the ICM Central Controller is able to route Web requests for contact as well.

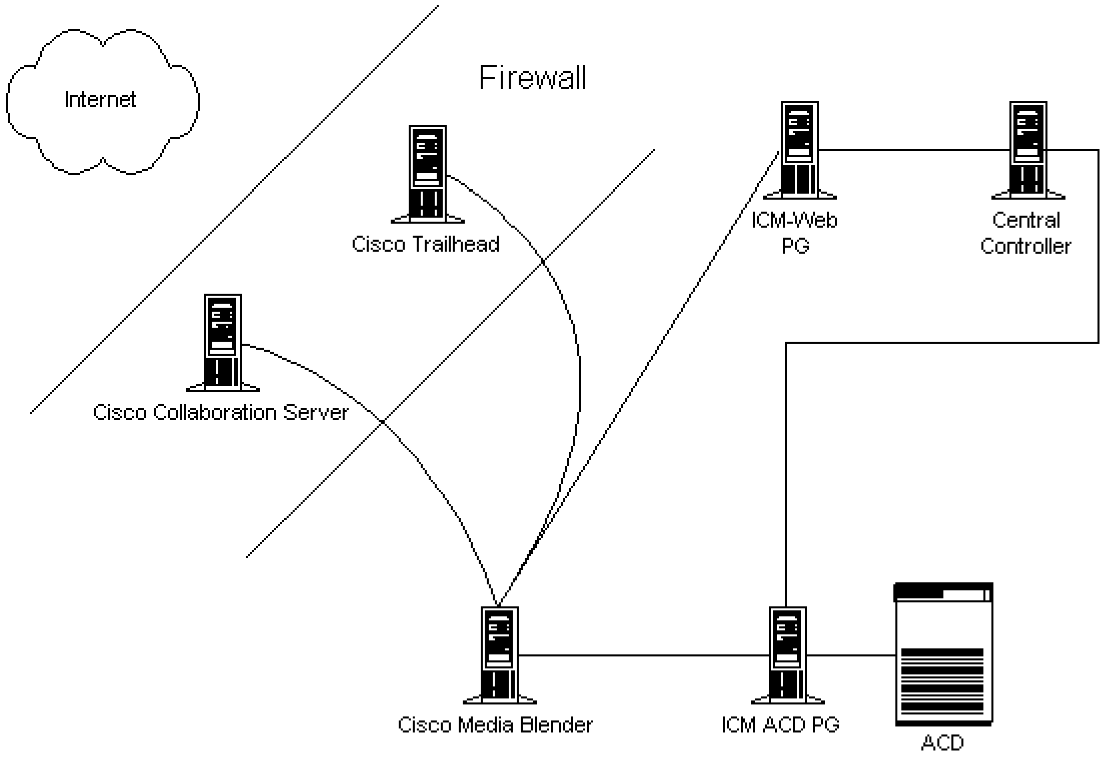
Integral to the solution is Cisco Trailhead, an application that accepts Web requests and submits them to the Web Peripheral Gateway (Web PG), which in turn submits them to the ICM Central Controller for routing. The Web PG effectively translates incoming data from the Web into information that can be interpreted by the ICM Central Controller, which can then route Web requests to appropriate sites, just as it would a telephone call.

Trailhead also accepts output from the ICM script, ensuring that each location handles the Web request appropriately. Locations that handle Web requests can provide these types of responses:

- **Basic Callback--**With Basic Callback, the caller who placed a Web request receives a phone call back from an agent.
- **Callback with Web Collaboration--**Callback with Web Collaboration allows callers to interact and share information with agents over the Web. In addition to receiving a phone call back from an agent, the caller and agent can share Web pages, forms, or applications using a Web browser.

ICM Web Option Components

The following figure illustrates the components of a simple ICM Web Option configuration.



For help in deploying the ICM Web Option product, see the *Cisco ICM Web Option Implementation Map* included with this release.

Cisco Collaboration Server (CCS)

CCS, Version 3.01, enables the World Wide Web as a point of contact between an enterprise and its customers. It allows a customer to interact with live call center agents using the Internet.

Cisco Trailhead (CTH)

CTH, Version 4.0, translates incoming Web data into information that can be interpreted by the ICM software. Once the ICM script has routed the call, Trailhead also interprets ICM data and uses it to redirect the caller to the selected appropriate call center.

Cisco Media Blender (CMB)

CMB, Version 4.0, enables the Automatic Call Distributor (ACD) to accept and handle Web-based requests, providing for blended Web Collaboration. Media Blender allows you to synchronize your Web-based and ACD-based call center systems by sharing Computer Telephony Integration (CTI) events among participating media. A typical blended configuration includes a Collaboration medium, an ACD medium, and a Trailhead medium.

Cisco Intelligent Contact Management (ICM)

ICM, Version 4.1.4, routes incoming Web-requests to the appropriate site and agent skill group. Routing logic is established in an ICM script, which returns a label that determines the final destination of the Web request.

Cisco ICM Web Peripheral Gateway (Web PG)

The Web PG allows the Trailhead medium to communicate with the ICM system. The Web PG in the ICM Web Option configuration must be set up as the Routing Client. The Routing client requests a route from the ICM, receives a response, and delivers the call to the specified destination.

Cisco Enterprise Computer Telephony Integration (ECTI) Server

ECTI provides the computer telephony integration (CTI) interface that Media Blender can use to communicate with the ACD. Versions 3.1 and 4.0 of the Media Blender support only the ECTI driver.

Automatic Call Distributor (ACD)

The ACD provides queueing and agent selection for the call center. The following ACDs are supported for this release:

- Aspect CallCenter, Versions 6.2 and 7.2
- Lucenty Definity G3, Version 6.3
- Nortel Meridian1, Version 24, SCCS 1.5
- Nortel Symposium, Versions 1.5 and 3.0
- NEC NEAX 2400, Version 4.12

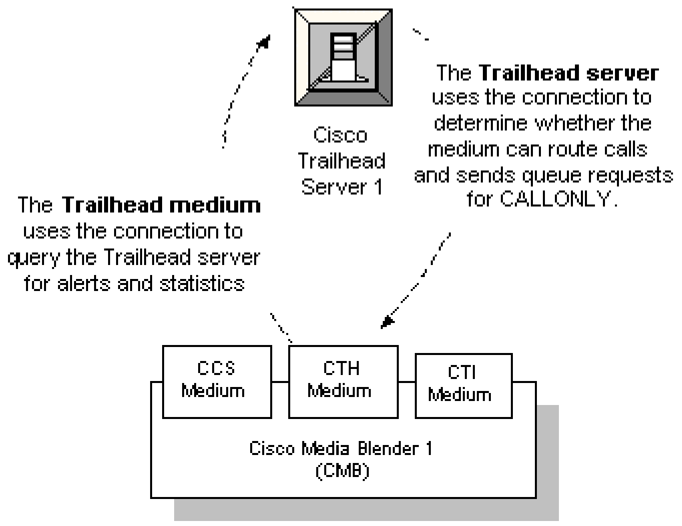
Note that the Meridian, Symposium, and NEC switches do not support the predictive CTI strategy.

For more information about the Cisco ICM Web Option, see the *ICM Web Option Overvie*w, included with this release.

Trailhead Components and Connections

Trailhead is made up of two software components: the Trailhead server, which resides outside the corporate firewall, and the Trailhead medium, which resides inside the firewall on the Media Blender machine. Each component is configured using a `Trailhead.properties` file.

The Trailhead server and the Trailhead medium communicate with each other using *connections*. Outside the firewall, Trailhead's connections query whether the Trailhead media inside the firewall can route calls. Inside the firewall, the Trailhead medium's connections query Trailhead for any alerts and statistics.


Cisco Trailhead Server 1
The **Trailhead server** uses the connection to determine whether the medium can route calls and sends queue requests for CALLONLY.
The **Trailhead medium** uses the connection to query the Trailhead server for alerts and statistics
CCS Medium
CTH Medium
CTI Medium
Cisco Media Blender 1 (CMB)

The Trailhead medium can be set up to perform two functions:

- Routing of requests from the Trailhead server to the Web PG
- Initiating Basic Callback requests

Note that you can set up a Trailhead medium to perform either or both of these tasks.

About Trailhead connections

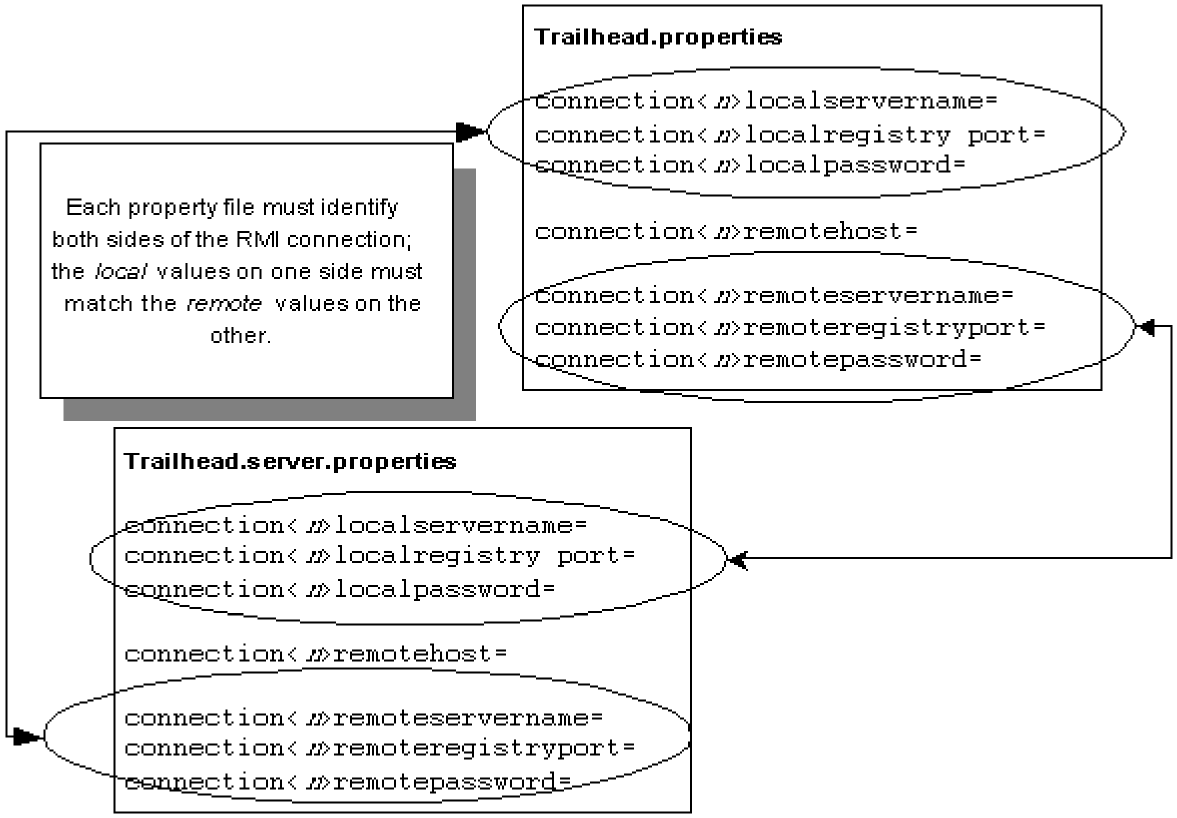
The Trailhead server and the Trailhead medium communicate with each other through the corporate firewall. To accommodate this, the Trailhead connection uses the Sun Microsystems Remote Method Invocation (RMI). With such a connection, an RMI registry exists on both sides of the firewall. Each registry identifies its machine to the machine on the other side of the firewall.

When defining Trailhead connections, you need to define your connections in property files on both the Trailhead medium and the Trailhead server. The properties files are:

- `\CiscoTH\servlet\properties\`**`Trailhead.server.properties`** (on the Trailhead server)
- `\CiscoMB\servlet\properties\blender\`**`Trailhead.properties`** (the Trailhead medium on the Media Blender machine)

Each connection registers at its *registry port* using its *local server name*. The connection accepts incoming requests that are made to its registry port, as long the request provides the password (local password) to the server.

For one side of the RMI connection to connect to the other side, it needs to know the hostname and/or IP address, the port, and server name to request and the password to provide. You can set these on each side of the firewall using `Trailhead.properties` (for the Trailhead medium) and `Trailhead.server.properties` (for the Trailhead server.) The following image illustrates the properties used on both sides of the firewall to identify the RMI connection:


Trailhead.properties
connection<n>localservername=
connection<n>localregistry port=
connection<n>localpassword=
connection<n>remotehost=
connection<n>remoteservername=
connection<n>remoteregistryport=
connection<n>remotepassword=
Each property file must identify both sides of the RMI connection; the *local* values on one side must match the *remote* values on the other.
Trailhead.server.properties
connection<n>localservername=
connection<n>localregistry port=
connection<n>localpassword=
connection<n>remotehost=
connection<n>remoteservername=
connection<n>remoteregistryport=
connection<n>remotepassword=

For complete information on each of these properties, see Configuration file reference.

Connection Types

Connections on the Trailhead server can be configured as *`CanRoute`* and *`CanQueue`* connections.

`CanRoute` connections are connections that can be used to route a session to a Trailhead Web Collaboration destination. If the Media Blender at the other end of the connection is connected to a Web PG for routing Web callback requests, the connection can route.

`CanQueue` connections are connections that can be used to queue calls using Basic Callback destinations. `CanQueue` is used for Basic Callback only. If the Media Blender at the other end of the connection is used for Basic Callback, then the connection should be set so it can queue. Note that you can set up a Trailhead medium to perform either or both of these tasks.

To allow route requests to get from the Trailhead server to the Web PG

To perform routing, the Trailhead medium MUST have a `CanRoute` connection to the Trailhead server. Its destination chooser must be DcICM. (See About Destination Choosers for more information.) All routing destinations are configured on the Trailhead medium.

To initiate Basic Callback requests

In this case, the Trailhead medium must have a `CanQueue` connection on the Trailhead server. The Media Blender must have an ACD medium configured as well.

If a Trailhead medium does Basic Callback but not routing, its destination chooser should be DcNull. (See About Destination Choosers for more information.)
.

Understanding Trailhead Destinations

The Trailhead software package gathers information about a caller placing a Web-based request and submits it to the ICM. A script set up in the ICM receives the caller data and performs all routing logic, ensuring the call can be routed to an appropriate location. At this point, Trailhead again takes over, receiving ICM script output and providing appropriate response to the Web request.

ICM scripts route calls to call centers at different locations. The script returns labels, which are simply strings that identify each call center. With Trailhead, you set up *destinations* that correspond to each label. For each destination, you can specify the type of response that should be provided to the Web request (Basic Callback or Callback and Web Collaboration, as explained below).

Destinations are defined by the `Trailhead.properties` file on the Trailhead medium, which resides on the Media Blender server. You can set up destinations that do the following:

- Provide successful response to web callback requests
- Handle situations when callback is unavailable
- Handle ring, no answer, error, and default ICM terminations

Destinations that Provide Response to Web Callback Requests

Destinations can be set up to provide these types of response:

Basic Callback--With Basic Callback, the caller who placed a Web request receives a call back from an agent. When a destination is set up to provide Basic Callback only:

1. Trailhead sends a URL that tells the caller to expect a call back.
2. Trailhead sends a message to Media Blender, requesting that it place an outbound call to the caller.
3. The ACD routes the call to the agent

When you create destinations that provide Basic Callback, you assign them a CALLONLY destination type. This means that requests submitted to this destination are queued internally to the switch.

Basic Callback and Web Collaboration--In configurations that include the Cisco Collaboration Server (CCS), Trailhead can ensure that callers participate in a Collaboration session. Web collaboration allows callers to interact and share information with agents over the Web. The caller and agent can share Web pages, forms, or applications using a Web browser (See the CCS documentation for supported Web browsers).

When responding to a request for collaboration:

1. Trailhead serves the caller with a URL that launches a Collaboration session.
2. CCS sends a message to Media Blender, requesting that it place an outbound call to the caller.
3. The ACD routes the call to the agent.
4. The Collaboration Server connects the caller and the agent via the Web

When you create destinations that provide Web Collaboration, you assign them a *COLLAB* destination type. This means that requests submitted to this destination are queued externally, to the Collaboration Server. See Setting Destination Type, in the Configuring Destinations section of this guide for more information.

How Destinations Reflect Your Call Centers

Trailhead Destinations do not necessarily reflect individual call centers within your enterprise. Instead, they reflect a type of response provided to incoming Web requests. For example, consider a configuration that includes these two call centers:

- Boston--All agents at this call center will provide Web collaboration.
- San Jose--Some agents at this call center will provide Web collaboration. Some agents will NOT have access to Collaboration and must always provide Basic callback only.

To accommodate this configuration, set up Trailhead destinations as follows:

This call center...	**...requires this many Trailhead destinations**
Boston	One destination, that provides Web Collaboration
San Jose	Two destinations. The first destination provides Web Collaboration; the other destination should provide Basic Callback for those agents who do not have access to Collaboration.

Destinations that Handle Situations When Response is Not Available

You also set up destinations that correspond to ICM labels that indicate web response is not available. For example, you can set up a noagents destination to inform the caller of the situation. You could also set up a holiday destination, to handle situations when a call center is closed due to a holiday.These destinations serve URLs to callers, informing them of the reason response is unavailable. These are referred to as *noncalling* destinations.

Destinations that Handle ring, busy, error, and default ICM Conditions

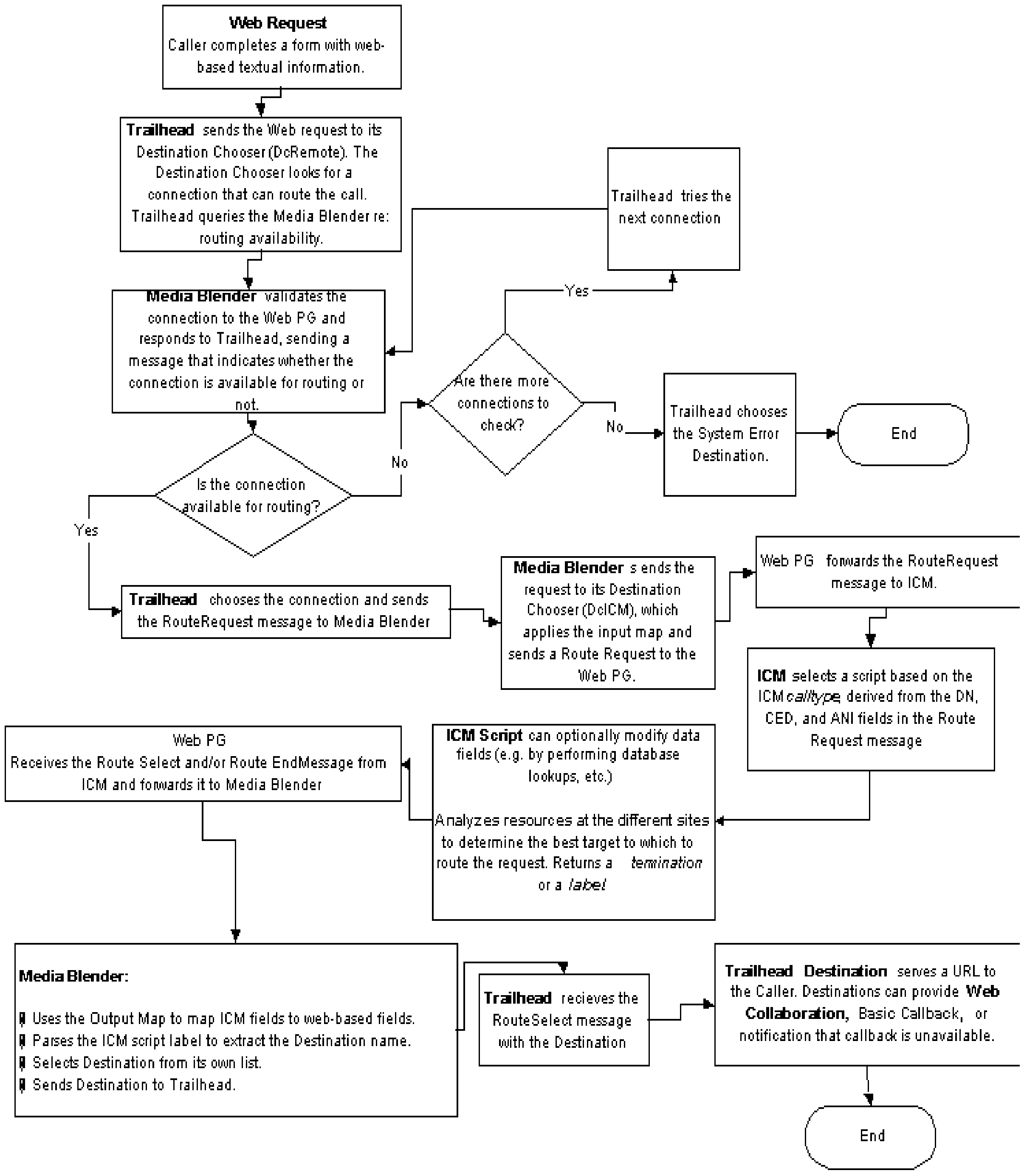
The ICM script can return special terminations rather than labels. These terminations are:

- ring
- busy
- error
- default

Although these terminations can be used to indicate that response is not available, you can also use them to handle crank or otherwise troublesome telephone calls.

Call Flow Through the Cisco ICM Web Option

The following diagram illustrates how a Web call might be routed through a configuration with Trailhead and ICM:



Web Request Caller completes a form with web-based textual information.
Trailhead sends the Web request to its Destination Chooser (DcRemote). The Destination Chooser looks for a connection that can route the call. Trailhead queries the Media Blender re: routing availability.
Media Blender validates the connection to the Web PG and responds to Trailhead, sending a message that indicates whether the connection is available for routing or not.
Is the connection available for routing?
Yes
No
Are there more connections to check?
Yes
Trailhead tries the next connection
No
Trailhead chooses the System Error Destination.
End
Trailhead chooses the connection and sends the RouteRequest message to Media Blender
Media Blender sends the request to its Destination Chooser (DcICM), which applies the input map and sends a Route Request to the Web PG.
Web PG forwards the RouteRequest message to ICM.
ICM selects a script based on the ICM *calltype*, derived from the DN, CED, and ANI fields in the Route Request message
ICM Script can optionally modify data fields (e.g. by performing database lookups, etc.)
Analyzes resources at the different sites to determine the best target to which to route the request. Returns a *termination* or a *label*
Web PG Receives the Route Select and/or Route EndMessage from ICM and forwards it to Media Blender
Media Blender:
Uses the Output Map to map ICM fields to web-based fields.
Parses the ICM script label to extract the Destination name.
Selects Destination from its own list.
Sends Destination to Trailhead.
Trailhead recieves the RouteSelect message with the Destination
Trailhead Destination serves a URL to the Caller. Destinations can provide **Web Collaboration**, Basic Callback, or notification that callback is unavailable.
End

About Destination Choosers

Destination choosers are dynamically loaded with the Trailhead software and determine how Trailhead should choose a destination. Both the Trailhead server and the Trailhead medium contain destination choosers. In ICM Web configurations, the default destination choosers rely on the ICM script to actually determine the correct destination for the call. The default destination choosers in the ICM Web Option environment are:

Trailhead Component	**Destination Chooser**	**Description**
Trailhead server	DcRemote	Sends a message to one of the configured `CanRoute` connections, asking the server on the other side of this connection to supply a destination, using its DestChooser.
Trailhead medium (on the CMB server)	DcICM	Sends a RouteRequest message to the Web PG which runs an ICM script and returns a label. The label matches a destination, which is returned.
Trailhead medium	DcNull	Allows a medium to accomplish ONLY Basic Callback. Use DcNull if the Trailhead medium should not perform destination routing.

Firewall Configuration and Networking

This section provides information about the following:

- Firewall Configurations
- Networking Considerations

Firewall Configurations

Media Blender communicates with the Trailhead Server through the corporate firewall using the Sun Microsystems Remote Method Invocation (RMI). This section describes the firewall configurations supported in this release, as well as special setup required for optimal performance.

The firewall between Media Blender and the Trailhead server can be set up in any of the following four modes:

- Polling over outbound socket connections from Media Blender to Trailhead
- Two-way socket connection
- Classic HTTP Proxy mode
- Transparent HTTP mode

Polling Over Outbound Socket Connections Mode

The polling mode allows outbound socket connections from Media Blender to the Trailhead server. Media Blender connects to the Trailhead server using TCP/IP socket connections. If you use this mode, ensure that your firewall permits outbound socket connections from Media Blender to Trailhead on all ports. This mode provides high throughput and is the recommended configuration.

Note: In this mode, you must disable attempts by the Trailhead server to automatically connect to the Trailhead medium. The `Trailhead.server.properties` file contains a property, `connection<`*`n`*`>.disableautoconnect`, which you must set to ***true*** to ensure the Trailhead server does not try to automatically connect to the Trailhead medium. This property setting ensures that the Trailhead medium will instead poll the Trailhead server from inside the firewall.

Two-way Socket Connection Mode

In the two-way socket connection mode, the firewall allows both of the following:

- Outbound socket connections from Media Blender to Trailhead
- Inbound socket connections from Trailhead to Media Blender

If you use this mode, ensure that your firewall permits inbound and outbound connections between Media Blender and Trailhead on all ports. This mode provides the highest throughput and the lowest delays of the four modes.

Note: If the firewall will allow both inbound and outbound communication, you must set the `connection<n>disableauotoconnect` property in `Trailhead.server.properties` to **false**. This property setting ensures that the Trailhead server will connect to the Trailhead medium automatically, initiating two-way communication through the connections.

Classic HTTP Proxy Mode

In the Classic HTTP proxy mode, the firewall permits outbound HTTP communication only to all ports. Media Blender connects to the Trailhead server using HTTP to the RMI registry port (1099 default). This mode provides the lowest throughput of the four modes. Make sure the checkurls property in the `Trailhead.properties` file on the Media Blender server is always set to false.

If you set up your firewall using this mode, you must make sure you set up the proxy parameters on the Media Blender using ServletExec Administration and the LoadWLRoot servlet. (See the *Media Blender Installation Guide* for more information.)

Transparent HTTP Mode

The firewall permits HTTP communication over all ports. There is no special set up required for this mode.

Networking Considerations

The following are some networking considerations:

- Both the Trailhead server and the Media Blender server require static IP addresses.
- Regardless of your firewall setup, you must maintain a hosts file rather than use Domain Name Service (DNS) to identify the IP address of your machines. Using DNS may result in RMI connectivity problems. The hosts file resides at these locations:
  - Windows NT: `winnt\system32\drivers\etc\hosts`
  - Solaris: `/etc/hosts`

The hosts file on the Blender machine must have an entry for the Trailhead server; the hosts file on the Trailhead server must have an entry for the Blender machine.

- If using Classic HTTP proxy mode, do one of the following:
  - Ensure the firewall machine contains a hosts file entry **or**
  - Specify the IP address in the Trailhead medium's property file on the Blender server (`Trailhead.properties`.)
- If using Classic HTTP proxy mode, the Trailhead server IP address must not be accessible by any means other than what is allowed through the firewall. To verify this, try to ping the Trailhead machine; you should receive the message "Destination host unreachable." (Receiving a timeout message does not ensure the machine is unreachable.)

Trailhead Failover Using LocalDirector

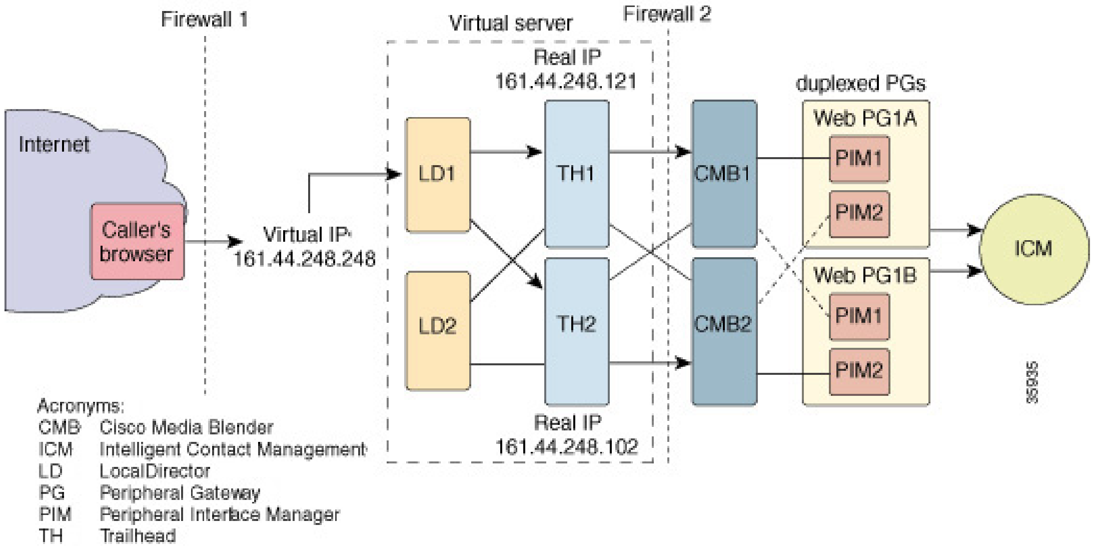
The ICM Web Option system is comprised of two paths--a routing path and a CTI path. This section describes only the routing path. When Trailhead routes requests to the ICM Central Controller using a routing configuration that includes the Cisco LocalDirector, the goal of the configuration is to eliminate a single point of failure, namely Trailhead.

How LocalDirector Helps

If any of the following events occur, the system cannot route Web session requests:

- The Trailhead server goes down
- The Trailhead/Media Blender connection fails.
- The Routing Media Blender fails.
- The connection between the Media Blender and the Web PG fails.

With the proposed solution using redundant LocalDirectors, Web session requests coming from the Web site, are routed to the redundant routing path, which is shown in the following figure:



Trailhead failover is accomplished using the Cisco LocalDirector. The LocalDirector offers a high-availability, Internet scalability solution that provides load balancing and a failover mechanism to eliminate points of failure for a server farm.

You can use a Cisco LocalDirector and the LocalDirector User Interface (LUI) software to *load balance* and *probe* two Trailhead servers. Load balancing, performed by the LocalDirector, is the distribution of traffic between the two servers. Probing, performed by the LUI, is the checking of the servers at specified intervals to be sure that the Web link is active. If the LUI probe detects that a Web link is inactive for one Trailhead server, it will direct the LD to designate that server as "out of service" (OOS) and direct traffic to the other Trailhead server.

Any ongoing sessions on the Trailhead server might be lost, but new session requests will be routed. The components or links that have gone down can be restored by hot swapping.

Limitations

Note that this configuration does not eliminate or overcome all modes of failure. Some of the limitations of the system include:

- Failover detection is limited to detecting if the Trailhead server is reachable over the network.
- If a Cisco Collaboration Server or a Cisco Media Blender in the CTI path of the system (not shown above) goes down, the system might still direct requests to that server. Such requests will, however, not be served.

Configuration for Trailhead Failover

For Trailhead failover using the LocalDirector with the ICM Web Option solution, you need a pair of Trailhead servers. For each additional Trailhead server, you need a complementary Cisco Media Blender and a Web PG. The figure shown previously illustrates the configuration. You must configure the following:

- Trailhead and Media Blender connections
- Trailhead destinations
- Duplexed Web PG
- Redundant LocalDirectors

Trailhead and Media Blender Connections

Each Trailhead server is configured to connect to both CMB1 and CMB2. When a request comes in from the LocalDirector, it goes to Trailhead (TH1). Trailhead sends the data to one CMB, not to both. TH1 chooses the CMB that is up. If one connection is down, Trailhead uses the other.

Configure the connections for the Trailhead medium on the CMB in the `trailhead.properties` file. Configure the connections for the Trailhead server in the `trailhead.server.properties` file. See the section Trailhead Components and Connections for additional details.

In the `Trailhead.server.properties` file on the Trailhead server, set the value for the `FormURL` property to the virtual IP address for the callback form (`callform.html`). The callback form itself should be on a server other than the Trailhead server as explained in the following section. For information about the virtual IP address, see Creating a Virtual Server.

Trailhead Destinations

To ensure that the destination URLs are accessible by a Trailhead server in the event that one Trailhead server goes down, you must move the HTML forms in the URLs to a new directory on the Cisco Collaboration Server (CCS) or some other Web server. You must also configure two property files. This is a two-step process:

1. Create a new directory and move the destination forms from the Trailhead server to this directory.
2. Configure two property files to point to the new directory.

Step 1: Create Directory and Move Forms

From the following directory location on the Cisco Collaboration Server:

```
   C:\<CCS dir>\pub\html\Forms\
```

create a `\trailhead` subdirectory:

```
   C:\<CCS dir>\pub\html\Forms\trailhead
```

and move all the destination forms to this new subdirectory from the following Trailhead server directory:

```
   C:\CiscoTH\pub\html\Forms\trailhead
```

Step 2: Configure the Property Files

In both the `Trailhead.properties` file on the Media Blender server and the `Trailhead.server.properties` file on the Trailhead server you fill find a `destination<n>.URL` property. Change the value of this property in **both** of the files so that they point to the new Trailhead directory you have just created:

```
destination<n>.URL=http://<CCS-HOST-NAME>/<CCS
dir>/html/Forms/trailhead/<filename>.html
```

Note: Although the above example uses the Cisco Collaboration Server, you can move the forms to a directory on any Web server and then have the `destination<n>.URL` property point to that location.

Duplexed Web PG

The ICM Web Peripheral Gateway (PG) connects to the Media Blender. To help keep things running, set up a duplexed Web PG using two Web PGs with Voice Response Unit Peripheral Interface Managers (VRU PIMs), as shown in the previous figure.

The duplexed Web PG operates as one PG with two sides--Side A and Side B. If one Web PG goes down, the other side becomes active. The shadow PIM picks it up. For information on how to install and configure duplexed PGs, see Chapters 2 and 7 of the *ICR Installation Guide*. For more information on fault tolerance using the ICM software, see Chapter 2 of the *ICR Administrator Guide*.

Redundant LocalDirectors

For full redundancy, you need two Cisco LocalDirectors. The two LocalDirectors can be configured to back each other up, so that when one unit fails, the other takes over for it. However, they are only one component of a fault tolerant Web application system. See the White Paper: Failover Configuration for LocalDirector for details on how to set up a primary and a secondary LocalDirector.
When you have both a primary and a secondary LocalDirector, there is still only one virtual server and one virtual IP address. If one LocalDirector goes down, the other one assumes all of its configurations.

Setting up the LocalDirector

You can configure the LocalDirector directly from the command-line interface or by using the LocalDirector User Interface application. The LUI is necessary for configuring probes. After you have configured the LocalDirector, the fact that there is more than one Trailhead server is invisible to the caller, because the LocalDirector presents a "virtual" server to the caller, and the requests to the virtual server are directed to the "real" Trailhead servers. Each Trailhead server and the virtual server have IP addresses that are added into the LocalDirector configuration file (a command script). See the Sample Configuration File.

This section provides information about the following:

- Installing LocalDirector
- Creating a Virtual Server
- Creating a Probe
- Sample Configuration File

Installing LocalDirector

Before you can configure the LocalDirector using the LUI application, you must do the following:

1. Install the LocalDirector User Interface application on your Windows NT or Solaris workstation.

2. Connect the enclosed null modem serial cable to the console port of the LocalDirector and the other end of the cable into your ASCII terminal or the serial port of your PC (running a terminal emulation program). Directly connecting to the LocalDirector is necessary so you can initially configure the IP address of the LocalDirector.

For details on how to install and use the LUI and the LocalDirector, see the *LocalDirector User Interface Install and User Guide* and the *LocalDirector Installation and Configuration Guide*.

Creating a Virtual Server

A virtual server presents a single IP address that represents two or more real servers. The virtual IP address is published to the user community, but the real Trailhead server IP addresses can remain unpublished, allowing you to hide actual site implementation details and provide a single point of contact for users. The virtual server address can be accessed only from the client side of LocalDirector. Also clients and the real servers bound to the LocalDirector virtual server cannot be located on the same side of LocalDirector.

In the `Trailhead.server.properties` file on the Trailhead server, set the value for the `FormURL` property to the virtual IP address for the callback form (`callform.html`).
Using the LocalDirector User Interface application, you must first create a virtual server and then create the real servers. Basically, you add the Name, IP address, Port, and bind ID using the Create Virtual Server window and the Create Real Server window.

See the Adding Virtual Servers and Adding Real Servers to Virtual Servers sections of the *LocalDirector User Interface Install and User Guide*.

Creating the Probe

In the Tree View of the LocalDirector User Interface Window, you will see a probes directory under the virtual server directory. Select Probes and click the Create New Web Probe icon. On the Create New Web Probe window, enter the following information into the appropriate fields:

- A statement that describes the purpose of the probe, such as "Test TH link."
- The URL for the link you want to test; for example, `http://161.44.248.121/`
- The interval of time between probes; for example, 30 minutes if you want LocalDirector to check the link every 30 minutes.

See Setting Up Probes in the *LocalDirector User Interface Install and User Guide* for more details.

For information on setting up the LocalDirector to do load balancing, see the *LocalDirector Installation and Configuration Guide.*

Sample LocalDirector Configuration File

By entering information into the LocalDirector User Interface application using directed mode, the following configuration file was created. The bold text near the end of the file shows the virtual, real, and bind IP address information. Each bind command binds a real Trailhead server to the virtual server. The caller's request goes first to the **virtual** server and then to the Trailhead server listed first in the file (first **real** IP address). Subsequent connections to the virtual server are load balanced based on a user-selectable predictor algorithm. The default predictor algorithm is "leastconns" (least number of connections). Note that the first IP address listed in the following file is that of the LocalDirector.

```
: LocalDirector 430 Version 3.3.2
syslog output 20.3
no syslog console
enable password 000000000000000000000000000000 encrypted
hostname localdir
no shutdown ethernet 0
no shutdown ethernet 1
shutdown ethernet 2
no shutdown ethernet 3
interface ethernet 0 auto
interface ethernet 1 auto
interface ethernet 2 auto
interface ethernet 3 auto
mtu 0 1500
mtu 1 1500
mtu 2 1500
mtu 3 1500
multiring all
no secure  0
no secure  1
no secure  2
no secure  3
ping-allow 0
ping-allow 1
ping-allow 2
ping-allow 3
ip address 161.44.248.250 255.255.252.0
route 0.0.0.0 0.0.0.0 161.44.248.1 1
no rip passive
rip version 1
failover ip address 0.0.0.0
no failover
failover hellotime 30
password cisco
telnet 161.44.251.121 255.255.252.0
telnet 161.44.240.120 255.255.255.0
snmp-server enable traps
snmp-server community public
no snmp-server contact
no snmp-server location
virtual 161.44.248.248:0:0:tcp is
real 161.44.248.121:0:0:tcp is
real 161.44.248.102:0:0:tcp is
bind 161.44.248.248:0:0:tcp 161.44.248.121:0:0:tcp
bind 161.44.248.248:0:0:tcp 161.44.248.102:0:0:tcp
localdir(config)#
```

Understanding ICM input and output

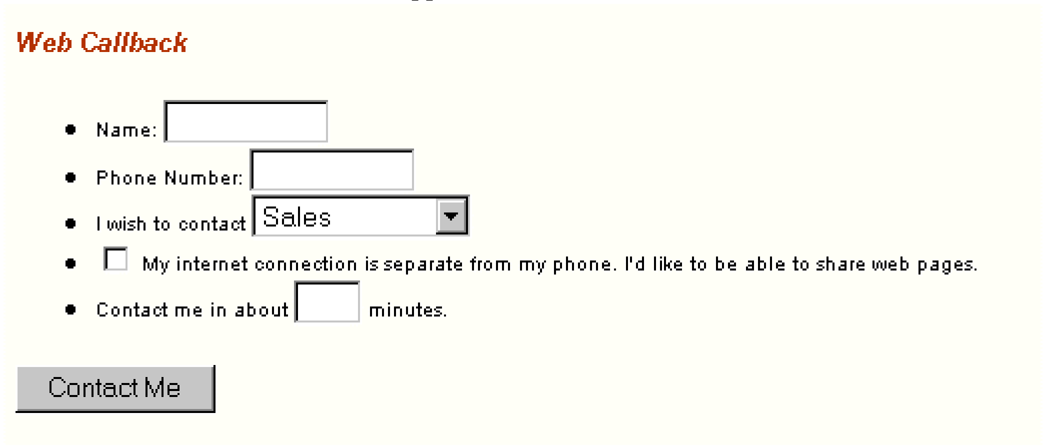
Trailhead provides information to the Intelligent Contact Management (ICM) Central Controller, which then handles routing of the request to the appropriate call center. ICM reads information about each incoming call, determines the best destination for the call, and sends the call to an appropriate destination. Trailhead then receives information from the ICM script after the request has been routed.

This section discusses these topics:

- ICM Input: The Web request
- The Route Request message to the Web PG

ICM Input: The Web request

The data Trailhead submits to ICM comes from the Web request, submitted by the caller. Callers place requests using the Web by filling out a callback form served by Trailhead. The callback form is an HTML form used to gather information about the caller. We provide a sample callback form, `callform.html`, which appears below:



Web Callback

- Name:
- Phone Number:
- I wish to contact Sales
- ☐ My internet connection is separate from my phone. I'd like to be able to share web pages.
- Contact me in about minutes.

Contact Me

This form gathers information about the caller. Your enterprise can alter the HTML that comprises this page to gather information pertinent to your business.
Some of the data passed in the Web request is entered by the caller; the Web request also captures less visible data, such as information about the Web browsers used in each session. The variables used to store all Web request values appear in the table below:

Visible fields

The fields on this form are described as follows:

Field name	Field name in HTML source	Description
Name	`Name`	The caller's name. Trailhead retrieves the name entered here and inserts it into the forms served back to the caller
Phone number	`PhoneNumber`	The caller's telephone number. Trailhead retrieves the number entered here and inserts it into the HTML forms served back to the caller.
I wish to contact...	`Route`	The routing code to which this call should be routed. You can format this as a drop-down box, allowing the caller to choose from a list of skills or geographic locations. Or, you can create several callback forms, one for every agent group, with the appropriate routing code hard-coded into a Route field.
My internet connection is separate from my phone. I'd like to be able to share Web pages...	`WantsCollab`	This flag determines whether the request is routed to a CALLONLY or a COLLAB destination. If the Collaboration flag is checked, the request is routed to a Collaboration destination. **Note**: The ICM script MUST evaluate the value in this field to route the request correctly. See Understanding the ICM script in this guide for more information.
Contact me in about ...minutes	`CallBackDelay`	This field lets the caller specify a delayed callback. The caller can enter how long the call center should wait to call back, beginning when the request was submitted. The caller enters this time in minutes (numeric values only). The time entered should not exceed 120 minutes.

Hidden fields

The following table describes the hidden HTML fields included in `callform.html` that you can pass to the ICM software.

Field	Description
`Trailhead`	If set to 1, this field identifies this as a Trailhead form.
`RequestFromURL`	This field identifies the page from which the Web request was made.
`User1-User4`	You can define these fields to provide information about the caller to ICM, Collaboration Server, or Media Blender. See the *Media Blender Configuration Handbook* for more information on defining these fields.

Customizing your form

The tables above show the fields that are included in the sample callform provided with Trailhead. You can customize the HTML source to add any other fields your site may require or to remove fields that are unimportant to your organization.

The Route Request Message to the Web PG

Trailhead sends a Route Request message to the Web PG, which, in turn sends the Route Request message to the ICM Central Controller. Using the Route Request message, the Web PG (the routing client) asks the ICM for a destination for the call.

The Route Request message sent to the Web PG can contain the following ICM variables. (Each of the 10 Call Variables listed below can have up to 40 characters.)

Variable	Description
`DN`	Dialed Number
`ANI`	Automatic Number Identification
`CED`	Caller Entered Digits
`CALLVAR1`	User-defined string
`CALLVAR2`	User-defined string
`CALLVAR3`	User-defined string
`CALLVAR4`	User-defined string
`CALLVAR5`	User-defined string
`CALLVAR6`	User-defined string
`CALLVAR7`	User-defined string
`CALLVAR8`	User-defined string
`CALLVAR9`	User-defined string
`CALLVAR10`	User-defined string

Mapping ICM and Web Data

You can ensure that information from the Web is passed to the ICM and to the ECTI server. You can also ensure that ICM information is passed back to the Web database. This section includes these topics:

- Mapping Web data to ICM variables
- Mapping ICM variables to Web variables
- Passing data to the Cisco ECTI server

Mapping Web Data to Route Request Variables

You populate the Route Request message with information gathered in the Web request using a text file called the input map. Trailhead's Input map (`in.map.properties`) is a text file that maps fields found on the Trailhead callback form with ICM fields. The format of each entry is as follows:

```
<ICM variable> = <Web variable>
```

For instance, the file might contain this entry:

```
CED=Route
```

This entry specifies that the value in the Route field on the web request should be passed in the CED (Caller Entered Digits) variable in the Route Request message. In this way, the Input Map translates the information from the incoming web request and populates a route request message submitted to ICM.

Note that you can set the ICM variable to a literal string rather than a Web variable. For instance, consider the following example entry:

```
DN="1000"
```

This line ensures that 1000 is always passed in the DN field.

Important: Mapping DN to a literal string ensures that the same script is run for all web calls. See How the ICM script is selected for alternative methods for selecting scripts.
A sample input map appears in the *Configuration File Reference* section of this guide.

Using ICM Call Variable 10 (Lucent switches only)

When using the predictive call strategy with a Lucent switch, Media Blender makes internal use of Call variable 10. If you need to be able to use call variable 10 in your application, you can free up variable 10 by setting the calltag property in `ACD.ciscocti.properties` to another ICM call variable. See the *Media Blender Reference Guide* for a description of this and all ECTI properties.

Mapping ICM Variables to Web Variables

ICM also passes caller data through each label to the Web PG. The Web PG then sends a Route Select message to Trailhead. You use a text file called an output map (`out.map.properties`) to map ICM values received in the Route Select message to web-based values that can repopulate the web request. The output map is optional.

The format of each entry in the file is as follows:

```
<ICM variable> = <Web variable>
```

Using the Output Map to Enhance ICM/Collaboration Reporting

The output map is particularly important if you want to be able to identify Web calls in your Collaboration reports. You can write an ICM script to associate a unique identifier, such as the ICM Router Call Key, with each Web call. You can then use the output map to place the identifier in one of the Collaboration call variables.

For instance, if you placed the ICM call identifier in ICM Call variable 6, an entry in your output map might be:

```
CallVar6=User1
```

This entry specifies that the value in the call variable 6 field on the ICM system should be placed in to User1 field in the Collaboration Server data base. The value in the User 1 field is displayed in the Collaboration Adminsitration Request report (see Reporting on Web calls and Understanding the ICM Script in this guide for more information.)

See The Web Request in this guide for a complete list of Web callform variables.
A sample output map appears in the Configuration File Reference section of this guide.

Note: You can also pass any of the Web fields to the Cisco ECTI server. See Passing data to the Cisco ECTI server for more information.

Passing Data to the ECTI Server and Back to the Web

You have the option of loading callback form data, as well as data obtained by the ICM script, into Cisco ECTI server call variables. The ECTI server can then use these values in a variety of CTI applications, such as agent screen pops.

You can map any information from any of the variables on the callback form, as well information stored in the 10 ICM call variables to any of the 10 variables on the CTI server. The 10 ICM call variables can contain up to 40 characters each.

Important: This release of Trailhead does not support the mapping of ICM expanded call variables to Web variables.

(For more information, see these sections of this guide: ICM input--the Web request and The Route Request message to the Web PG.)

To use this feature, you must set up a text file called an input map between the CTI server and the Web callback form. This file is called `cti.in.map.properties` and resides in the `CiscoMB/servlet/properties` directory. The format of each entry is as follows:

```
<ICM variable> = <Web variable>
```

For instance, the file might contain this entry:

```
CALLVAR2=user1
```

This entry would ensure that the value in the User1 field on the callback form is passed into Call Variable 2 on the CTI server.

Note that you can set the CTI server variable to a literal string rather than a Web variable. For instance, consider the following example entry:

```
CALLVAR1="Web Call"
```

This line ensures that the words "Web Call" are passed to call variable 1.

A sample ECTI peripheral input map appears in the Configuration File Reference section of this guide.

Note: You must identify the peripheral input map in your `ACD.ciscocti.properties` file, using the peripheral.inmap property. See the *Media Blender Reference Guide* for more information.

You can use the output map to then map any of the values stored in the ICM variables back to the Web database.

Sample files

We provide several sample files you can use to ensure Web and ICM data is passed to the ECTI server. These three samples reside in the `CiscoMB/servlet/properties` directory and are designed to be used together. Specifically:

`callvars-cti.in.map.properties`

This sample file maps Web call variables 1-10 ito the CTI call variables 1-10 (in the phantom callback to reserve the agent and Web callback, if there is one.)

`callvar-out.map.properties`

This sample file maps the CTI call variables 1 - 10 into the Web variables (from the Web request call).

`callvars-sanjose1.html`

This sample file is an HTML form that launches a Web Collaboration session. It substitutes in the CTI Call Variables 1-10 (from the Web request into the Web session in Collaboration in Blender.)

ICM Script Considerations

This section contains information about ICM script considerations.

How the ICM Script is Selected

ICM scripts are selected by the incoming ICM *call type*. ICM call types refer to different categories of calls; for instance a call type might be set up for Sales; another might be set up for Service. When a call comes in for Service, ICM selects and runs the appropriate script.

The call type for Web calls is determined by the Dialed Number (DN) field either by itself or with any combination of the Automatic Number Identification (ANI), and the Caller Entered Digits (CED) fields in the incoming Route Request Message.

In your input map, ensure the following Web variables are passed to these ICM variables:

ICM Variable	**Web Variable**
`DN`	"<script number>" *or* Route (see section below).
`ANI`	PHONENUMBER
`CED`	NAME

Using a Variable or Literal in the Dialed Number Field

You can set the DN field to a literal string by enclosing the script number within quotation marks. For instance, you can ensure that 1000 is always passed in the DN field by including the entry DN="1000" in your input map.

If you choose to map DN to a literal string, and if it is the only Route Request variable used to select the script, be aware that you will be able to run only one script for your Web requests--the script mapped to the literal passed in the DN field. If you also use the ANI and CED variables when setting up the Web call type, however, you can run multiple scripts.

If, however, you choose not to use a literal string, you can set the DN to equal the value in the Route field on the Web callback form. You can then maintain several scripts to route Web requests, based on whatever value is entered in the Route field on the callback form. Keep in mind, however, that setting the DN to a Web variable means that whatever is entered in the Route field will be mapped to the DN on the ICM Web PG. If a Web user tampers with call form data, in particular with the Route value, an unexpected value may be returned and mapped to the DN field. It is therefore possible that a Web-routing script may be selected that is different from the script you intended.

Understanding the ICM Script

The ICM script performs all tasks associated with ensuring the web request is routed to the correct peripheral target. The script can perform real-time analysis of available resources at each location in your configuration and make decisions on routing the requests based on that analysis.

ICM Script Tasks

When routing Web requests, the ICM script must perform these tasks:

- Evaluate available resources at each call center in your configuration.
- Determine whether the request should receive Web Collaboration or Basic Callback.
- Associate the call with a unique ICM identifier (for Collaboration reporting purposes--optional).
- Return labels that correspond to Trailhead destinations.

Evaluate Available Resources

ICM selects a resource based on a number of factors. For instance, a script might query all resources to determine the location of the agent who has been available the longest. Another factor ICM can use to route calls is the Minimum Expected Delay (MED). With MED, ICM queries selects the resource that expects the shortest wait for the call.

Choose between Collaboration and Basic Callback

The script can also evaluate values passed to the script from the Web request. For instance, the script might evaluate the value passed in the ANI field to determine the best skill group to which to route the call.

The route request also contains a flag that determines whether a Collaboration session or Basic Callback has been requested. This value is entered in the wantscollab field on the Web callback page. (A value of *on* indicates Collaboration; *off* indicates Basic Callback.) We suggest this value be mapped (using the input map) to one of the ICM call variables (CALLVAR2).

The script must evaluate the value in this field to determine proper routing of the Web request. Trailhead destinations are set up to provide either Web Collaboration OR Basic Callback. Therefore, the script must examine the value in the wantscollab field to ensure the request is routed to the correct destination.

Associate a Unique ICM Identifier with the Web request (optional)

This optional step ensures that Web calls routed through the ICM can be identified in Collaboration reports. Using a Set node, you can use the RouterCallDay and RouterCallKey variables to assign a unique identifier to one of the ICM call variables. You can then use your output map to map the identifier to Collaboration call variable 1 (User1). You can then identify Web calls that were routed through the ICM by locating the identifier in Collaboration reporting.

Return Labels that Correspond to Trailhead Destinations

ICM scripts return labels that indicate the peripheral target to which the call is routed. Labels are strings that are interpreted by the Web PG and point to particular locations and routing addresses within the configuration. (A routing address typically corresponds to a skill group.)

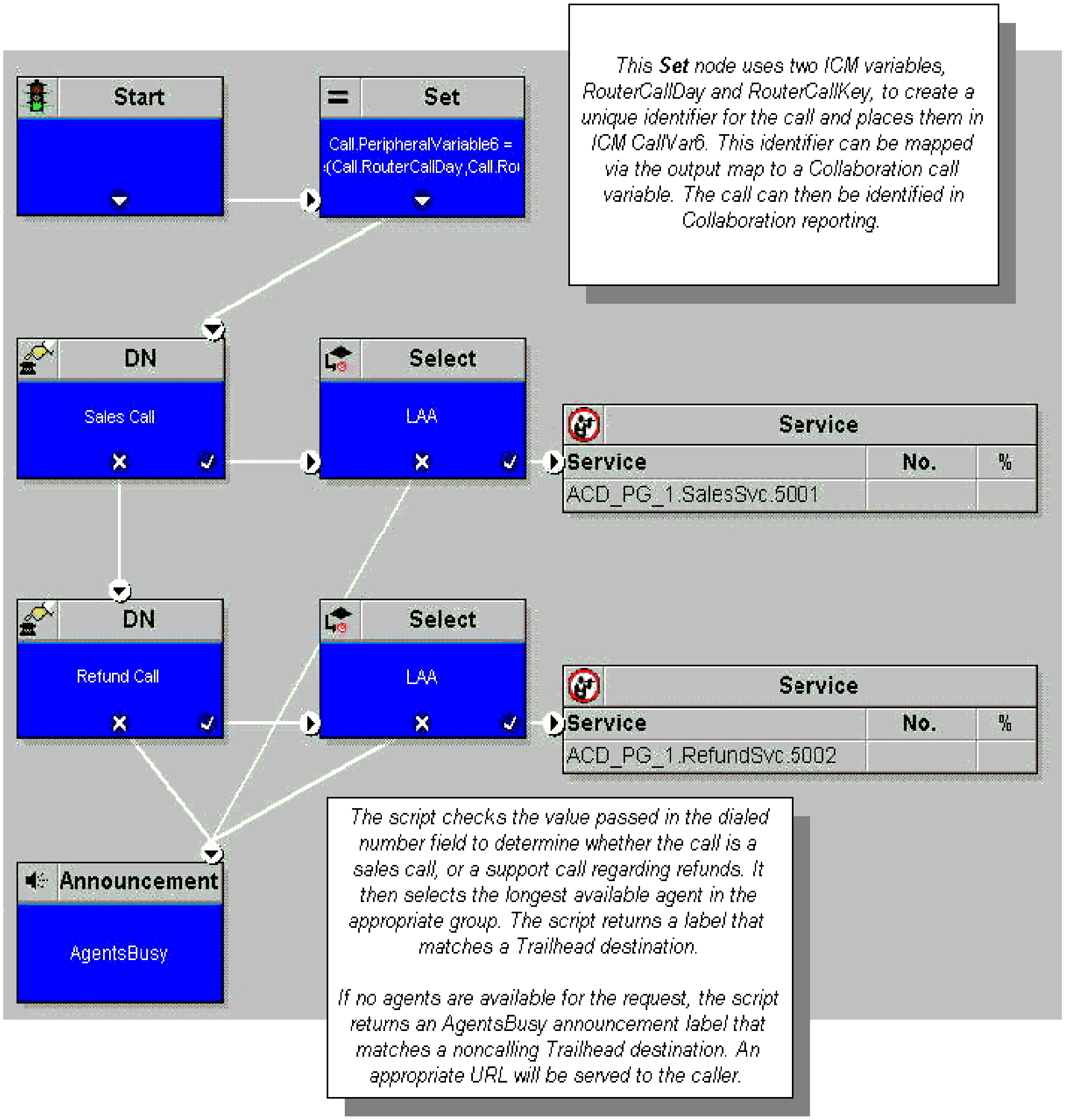
The Trailhead administrator sets up Trailhead destinations that correspond to each label output by an ICM script. Trailhead destinations simply indicate the method your site will use to respond the Web request. For instance, one Trailhead destination might provide Web Collaboration; another might provide Basic Callback only. Another destination might serve the caller a URL that indicates the call center is down. (See the Configuring destinations of this guide for more information on Trailhead destinations.)

Because Trailhead destinations indicate a type of response to a request, you will likely have more Trailhead destinations than actual call centers.

Your ICM script, therefore, must return labels that correspond to each Trailhead destination.

Example

Below is a sample script that might be used to route Web calls to two different types of agents.



*This **Set** node uses two ICM variables, RouterCallDay and RouterCallKey, to create a unique identifier for the call and places them in ICM CallVar6. This identifier can be mapped via the output map to a Collaboration call variable. The call can then be identified in Collaboration reporting.*

The script checks the value passed in the dialed number field to determine whether the call is a sales call, or a support call regarding refunds. It then selects the longest available agent in the appropriate group. The script returns a label that matches a Trailhead destination.

If no agents are available for the request, the script returns an AgentsBusy announcement label that matches a noncalling Trailhead destination. An appropriate URL will be served to the caller.

For more information on creating scripts, see the *ICR System Manager Guide.*

ICM Script Label Configuration

When Trailhead receives the label from the ICM, it extracts the destination name and the routing address that should be used for the call from the label name. In most cases, script labels should follow this format:

`<destination name><delimiter><routing address>`

where

- `<destination name>` indicates the location of the call center to which the request is routed.

Important: The destination name portion of each label must match the names of your Trailhead destinations. See the Defining and naming destinations section of this guide for more information.

- `<delimiter>` is any character used to separate the two portions of the label. You can define the delimiter to any character of your choice. (We suggest using a hypen.) See the Configuring destinations section of this guide for more information. (Note that you can choose not to use a delimiter, but rather a length-based parsing method for passing this information in the ICM label. See Reserving a number of characters for the destination name, below for more information.)
- `<routing address>` is the numeric identifier used by the ACD to determine how to queue the call. For example, in the Lucent Definity, this is the vector directory number (VDN).

For example, the label Boston1-12345 indicates that the call should be routed to the Boston1 location, and to VDN 12345.

For more information on ICM scripting logic, see the *ICR System Manager Guide.*

Reserving a number of characters for the destination name

You can choose not to use a delimiter in your label names, you can configure Trailhead to interpret the first *n* characters in a label name as the destination name. The ICMLabelLength property in the Trailhead properties file lets you specify how many characters reflect the destination name.

For instance, in the properties file for the Trailhead servlet, you might specify the ICMLabelLength property as 8. If the Label name is SanJose112345, Trailhead will interpret the first 8 characters as the destination name, or SanJose1. The routing address, therefore, is the remainder of the label name, or 12345.

If you choose this method, make sure your ICM Script labels follow the convention specified in the IcmLabelLength property in `Trailhead.CiscoVRU.properties`.

ICM Scripting Tip

Using the Trailhead medium on the Media Blender server, you can configure a CallOnly destination and a Collaborative destination that point to the same peripheral target. This requires that you:

1. Edit the in.map.properties file on the Media Blender Server
2. Configure the ICM
3. Set up the ICM routing script

Edit in.map.properties on the Media Blender Server

Edit the `in.map.properties` file on the Media Blender server and set the ICM Call Variable 2 to the Web WantsCollab field. For example, the file should contain this line:

```
CallVar2=WantsCollab
```

Configure the ICM

Create two labels with the same peripheral target; for example, site1_collab-6000 and site1_callonly-6000.

Set up the ICM routing script

This routing script example has two rules, one which maps to Service.Sales for a call-only session, and the other which maps to Service.Sales for a collaborative session. Using the following method, the same label is returned every time; that is, the first rule returns the site1_collab-6000 label, and the second rule returns the site1_callonly-6000 label.
Complete the following steps:

1. In the lower left corner of the Routes tab on the ICM Route Select Properties dialog box, select the "Allow connection for each target" checkbox.
2. Add the following formula in the "Consider If" section for the Collaborative destination:
   ```
   Service.Sales.AgentsLoggedOn > 0 && Call.PeripheralVar2 = =
     "on"
   ```
   This assumes that CallVar2 contains the contents of the Wants Collab form element mapped through the in.map.properties file.
3. Add the following formula in the "Consider If" section for the CallOnly destination:
   ```
   Service.Sales.AgentsLoggedOn > 0 && Call.PeripheralVar2 !=
     "on"
   ```
4. Create Label nodes for each Route Select connection and attach them. You can directly specify the returned label in each of these Label nodes.

Note: You can also use node connections after each route selection rule to set a call variable that should be returned in `out.map.properties` for that route selection. For example, a CMB call type could be set if the destined CMB was configured with multiple call strategies.
See the ICM documentation for additional information on scripting.

Reporting on Web Calls

Full reporting on Web requests routed through the ICM is not yet available. You can, however, track ICM-routed Web requests through Collaboration reporting. You do so by associating an ICM identifier with each Web call routed through the ICM.

To associate an ICM identifier with each Web call:

1. In the script that routes Web calls, include a Set node that uses the RouterCallDay and RouterCallKey variables to assign a unique ICM identifier to an ICM call variable associated with the request.

2. In the output map, map this ICM call variable to Collaboration call variable 1 (User1).

To generate a report that shows the call key:

1. In Collaboration Administration, select **Reports > Requests**.

   The **Reports: Requests** screen appears.

2. For Request type, choose **Extension**. Specify any other criteria as desired and click **run report**.

   The **Reports: Requests: results** screen appears. The ICM identifier will appear in the ApplStr1 column of the report.

Configuring Destinations

This section describes the different kinds of Trailhead destinations you may set up to handle different Web requests. It includes these sections:

- Defining and naming destinations
- Setting a destination's type
- About Callonly destinations
- Determine which URL should be served by each destination
- Example 1: Setting up a Basic Callback destination
- Example 2: Setting up a destination for Web Collaboration
- Example 3: Setting up a non-calling destination
- Destinations for other script terminations

Defining and Naming Destinations

You define parameters for all destinations that can route calls using one file, `Trailhead.properties`, which resides in the `\CiscoMB\servlet\properties\blender` directory on the Media Blender server.

Begin by numbering and naming each destination using the Destination<n>.Name property. For instance, these properties define two destinations:

```
Destination1.Name=
Destination2.Name=
```

(All properties that pertain to each destination are prefixed by the Destination<*n*>. prefix.) Keep in mind that, even though the destinations are numbered sequentially, you can delete a destination within the sequence without having to renumber your remaining destinations. For example, if you originally configure destinations 1-5, and then delete destination 3, you need not renumber destinations 4 and 5; Trailhead automatically accounts for the missing number.

Important: ICM labels that indicate destinations for Web requests are comprised of both the location name and the routing address that should be used for the call. Your destination names MUST match exactly the first portion of the ICM label associated with the destination. For example if the ICM script returns the label Boston1-12345, the Trailhead destination name should be Boston1. See Configuring Script Labels in this guide for more information.

The Trailhead medium on the Media Blender should have a full list of destinations, matching every label that can be returned by the ICM script.

The Trailhead server, since it is using a DcRemote destination chooser, gets its destinations from the Media Blender. Therefore, it needs only one destination, for SystemError.

Setting a Destination Type

In the Trailhead properties file you define the destination's type. The type of a destination determines what kind of response will be given to Web requests that are routed to the destination. Use the Destination<*n*>.Type property to establish the destination's type. Acceptable values are:

Destination Type	Description
`CALLONLY`	Basic Callback. This destination type queues the Web request internally, to the switch.
`COLLAB`	Web Collaboration. This destination type queues the web request to an external system (e.g. the Collaboration Server).
`NOCALL`	Non-calling. This destination type does not provide callback or collaboration to the caller, but instead serves a URL that informs the caller that response is unavailable. Use this type of destination when callback is unavailable or to handle crank or troublesome calls.

About Callonly Destinations

Callonly destinations are destinations that provide basic callback only to Web requests. Requests for basic callback are queued to an ACD using connections that have been configured as `CanQueue.` (See Connection Types in this guide for more information.

Destinations are defined for the Trailhead medium, in `Trailhead.properties` on the Media Blender server. For callonly destinations, you must establish the following four values:

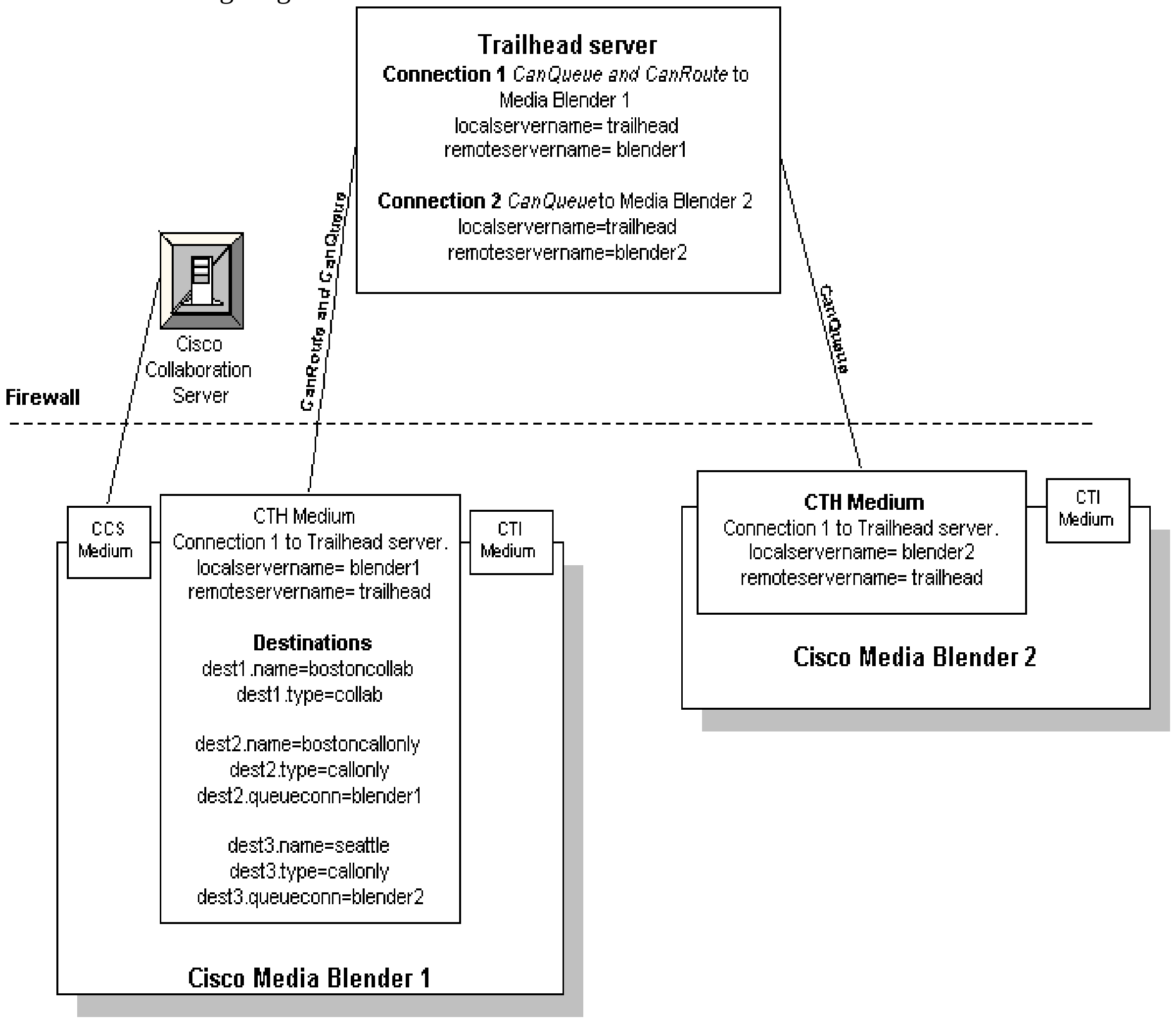
Property in Trailhead.properties	Description
`destination<n>.name=`	The destination's name. Remember that the ICM script must return a label that begins with this name to ensure requests are submitted to the correct destination.
`destination<n>.type=`	The destination's type, which should be callonly.
`destination<n>.URL=`	The URL that should be served to the caller by this destination.
`destination<n>.queueconn=`	The connection through which the request should be queued. This property should be set to the local server name of the Trailhead server that will queue the call. This property is explained further in the following section.

Understanding the `destination<n>.queueconn` Property

The `destination<n>.queueconn` property identifies the Trailhead connection this destination should used to queue basic callback requests. Each end of an RMI connection is defined using `connection<n>.localservername` and `connection<n>.remoteservername` properties. These properties identify both the "Trailhead end" and the "Media Blender end" of the connection.

In the `destination<n>.queueconn` property, enter the value in the `localservername` property of the "Trailhead end" of the connection. This value identifies the actual Media Blender that will queue the call to the ACD.

Consider the following diagram:


Trailhead server
Connection 1 CanQueue and CanRoute to Media Blender 1
localservername= trailhead
remoteservername= blender1
Connection 2 CanQueue to Media Blender 2
localservername=trailhead
remoteservername=blender2
Cisco Collaboration Server
Firewall
CanRoute and CanQueue
CanQueue
CCS Medium
CTH Medium
Connection 1 to Trailhead server.
localservername= blender1
remoteservername= trailhead
Destinations
dest1.name=bostoncollab
dest1.type=collab
dest2.name=bostoncallonly
dest2.type=callonly
dest2.queueconn=blender1
dest3.name=seattle
dest3.type=callonly
dest3.queueconn=blender2
CTI Medium
Cisco Media Blender 1
CTH Medium
Connection 1 to Trailhead server.
localservername= blender2
remoteservername= trailhead
CTI Medium
Cisco Media Blender 2

This diagram illustrates a Trailhead/Media Blender configuration set up to provide both Web collaboration and Basic callback. Media Blender 1 provides Web Collaboration and Basic Callback. Media Blender 2 provides Basic Callback only. Because Connection 1 on the Trailhead server is the only CanRoute connection, all destinations must be defined on Media Blender 1. Both Connection 1 and 2 are CanQueue connections; both connections can provide Basic Callback.

The configuration contains two *callonly* destinations: destination 2 (bostoncallonly) and destination 3 (seattle). When setting up the bostoncallonly destination, you must specify the "Trailhead-end" of the connection that will be used to queue callonly requests. Connection 1 on the Trailhead server will be used to queue callback requests for bostoncallonly. Therefore, the dest2.queueconn= property should be set to *blender1*.

When setting up the seattle destination, you must specify the "Media Blender end" of the connection that will queue calls routed to it. Connection 2 on the Trailhead server will be used to queue callback requests for seattle. Therefore, the `dest3.queueconn=` property should be set to *blender2*.

Determining the URL to be Served by Each Destination

In the `Trailhead.properties` file, you map each destination to a URL that should be served to callers when requests are received. Use the `Destination<*n*>.URL` property to specify appropriate URLs.

The following example sets the URL for a COLLAB (Collaboration) destination:

```
Destination1.URL=http://<servername>/success_v30.html
```

We provide sample URLs you can use and modify for your configurations. See Maintaining HTML Pages and Forms in this guide for more information.

Note for NOCALL destinations: NOCALL destinations can serve a URL, or, instead, serve only an HTTP error code and reason to the caller. If you choose to server an error code rather than a URL, used the **destination<*n*>errorcode** and **destination<*n*>errorreason** properties. See Configuration File Reference in this guide for information on these properties for both the Trailhead server and the Trailhead medium.

Example 1: Setting up a Basic Callback Destination

Use this procedure for all of your destinations that will provide Basic Callback only.
In the `Trailhead.properties` file do the following:

1. List each Destination using the `Destination<`*`n`*`>.Name` property:

   ```
   Destination1.Name=Boston1
   ```

2. Define the Destination type. Use CALLONLY for Basic Callback.

   ```
   Destination1.Type=CALLONLY
   ```

3. Enter the URL that should be served for this destination using the `Destination<`*`n`*`>.URL` property:

   ```
   Destination1.URL=http://<trailhead-host-name>
     /CiscoTH/pub/html/forms/blender/boston1.html
   ```

4. Define a connection to which this call should be queued:

   ```
   Destination1.QueueConn=<local servername of connection to
     use>
   ```

   Important: This connection name should be the name listed in the `localservername` property on the Trailhead server, the machine that ultimately processes the destination.

Example 2: Setting up a Destination for Web Collaboration

Use this procedure for all of your destinations that will provide Web Collaboration.
In the `Trailhead.properties` file:

1. List each destination using the Destination<*n*>.Name= property:

   ```
   Destination2.Name=San Jose
   ```

2. Define the destination type. Use COLLAB for Web Collaboration.

   ```
   Destination2.Type=COLLAB
   ```

3. Enter the URL that should be served for each destination using the Destination<n>URL property.

   ```
   Destination<2>.URL=http://<Trailhead server>
     /CiscoTH/pub/html/forms/blender/success_v30.html
   ```

   Note: If you are using the Cisco LocalDirector, the URL should point to the server where you have moved the forms.

Example 3: Setting up a Noncalling Destination

You can use noncalling destinations to respond when no agents are available or to handle crank or troublesome calls.

In the property file for the destination, do the following:

1. Define the destination using the `Destination<n>.Name=` property:

   ```
   Destination3.Name=noagents
   ```

2. Define the destination type using the `Destination<n>.Type` property. Use NOCALL for unavailable.

   ```
   Destination3.Type=NOCALL
   ```

3. Enter the URL that should be served for each destination using the Destination<*n*>.URL property.

   ```
   Destination3.URL=http://<Trailhead server>
    /CiscoTH/pub/html/Forms/blender/noagents.html
   ```

Note: Your site might use busy, ring, and error terminations to indicate down situations. See Destinations for other ICM script terminations in this guide for more information.

Destinations for Other Script Terminations

In most cases, the ICM script returns labels configured to indicate busy, after hours, and error situations. Such destinations correspond to labels set up in the ICM script.

However, your enterprise might configure a script to return ring or busy terminations in the ICM script instead of creating separate labels for these situations. Ring, busy, and error terminations differ from labels in that they do not direct the web request to a specific destination. In addition, ICM sometimes returns a default termination if it cannot find an appropriate label for the request. Should your ICM script be set up to return ring, busy, error, and default terminations, you must set up destinations that correspond to these terminations. For instance, you should set up a system busy destination and a system noagents destination to provide response to callers when the script returns these labels. Replace `<TRAILHEAD-HOST-NAME>` with the real name in the following properties in the `Trailhead.properties` file:

```
Destination9.Name=SysBusy
Destination9.Type=NOCALL
# URL of the page - replace <TRAILHEAD-HOST-NAME> with real name
Destination9.URL=http://<TRAILHEAD-HOST-NAME>
/CiscoTH/pub/html/Forms/blender/sysbusy.html
Destination9.Substitute=false
Destination9.SystemBusy=true

Destination10.Name=sysnoagents
Destination10.Type=NOCALL
# URL of the page - replace <TRAILHEAD-HOST-NAME> with real name
Destination10.URL=http://<TRAILHEAD-HOST-NAME>
/CiscoTH/pub/html/Forms/blender/sysnoagents.html
Destination10.Substitute=true
Destination10.SystemNoAgents=true

Destination11.Name=SysError
Destination11.Type=NOCALL
# URL of the page - replace <TRAILHEAD-HOST-NAME> with real name
Destination11.URL=http://<TRAILHEAD-HOST-NAME>
/CiscoTH/pub/html/Forms/blender/syserror.html
#Destination11.ErrorCode=500
#Destination11.ErrorReason=System Error - please try later
Destination11.SystemError=true
```

Additional properties are included in the file for holidays and after hours terminations.

Note: For DcICM, the destination names must match the ICM labels.

Configuration File Reference

This section describes all of the properties you can use in all of the configuration files used for Trailhead. It includes these files:

- On the Trailhead Server: trailhead.server.properties
- On the Media Blender server: the Trailhead medium (trailhead.properties)
- On the Media Blender server: the Input map
- On the Media Blender server: the Output map
- On the Media Blender server: the Peripheral Input map

Note: This section documents all properties you can use to configure behavior within these files. Be aware that not all properties shown here will appear in your property files when you install the software.

On the Trailhead Server: Trailhead.server.properties

You define the destinations used within your enterprise using the `Trailhead.server.properties` file, which resides on the Trailhead machine in `\CiscoTH\servlet\properties\`.

Note: Some properties require the URL of a Web page that should be served to callers for different circumstances. We provide sample HTML you can use when setting up these pages. These files reside in the directory `\<CCS dir>\pub\html\Forms\blender` on the Cisco Collaboration Server. Some HTML files reside in the `\CiscoTH\pub\html\Forms\` directory on the Trailhead server. Individual filenames are included in the appropriate property description.

See Maintaining Trailhead HTML Pages and Forms in this guide for more information on HTML files.

General Trailhead Properties

Use the properties in the `Trailhead.server.properties` file to configure the behavior of Trailhead in a Cisco ICM environment.

`adminname=`

Property type: String
Default value: admin

This property identifies the administration username used to access the Trailhead control panel (`http://<server>/TrailheadAdmin`). Note that this user name is automatically encrypted; when you edit this file, the name will appear differently.

`adminpw=`

Property type: String
Default value: None

This property identifies the administration password used to access the Trailhead control panel (`http://<server>/TrailheadAdmin`). Note that this user name is automatically encrypted; when you edit this file, the name will appear differently.

`autostart=`

Property type: Boolean
Default value: False

This property indicates whether Trailhead should automatically start with the Web server. Be sure not to set this property to true until you are sure that all of your media are properly configured.

`destchooser=`

Property type: string
Default: none

Required. This property identifies the destination chooser used by the Trailhead server. In ICM Web configurations, this should be Com.WebLine.Trailhead.DcRemote.

Specifying Trailhead Connections

Use these properties to set up the connections from your Trailhead servers to your Trailhead media. Note that, even though connections are numbered sequentially, you can delete a connection within the sequence without having to renumber your remaining connections.
In the properties that follow, note that "local" properties refer to the Trailhead medium. Properties that refer to "remote" values indicate the Trailhead server.

For more information on the Trailhead RMI connections, see Trailhead Components and Connections.

`connection<n>.localservername=`

Property type: String
Default: trailhead

This property identifies the local name for this connection (the Trailhead server).

`connection<n>.localregistryport=`

Property type: Numeric
Default: 1099

This property identifies the local port for this connection (the Trailhead server).

`connection<n>.localpassword=`

Property type: String
Default: trailheadpw

This property identifies the password for this connection. The value here must match the value indicated in the connection<*n*>.remotepassword property on the Trailhead medium. Note that this password is automatically encrypted; when you edit this file, the value you enter here will appear differently.

`connection<n>.remoteservername=`

Property type: String
Default: blender

This property identifies the remote name for this connection (the Trailhead medium). The value here must match the value in the `connection<`*`n`*`>.localservername` property on the Trailhead medium.

`connection<n>.remotehost=`

Property type: String
Default: None

This property identifies the remote host for this connection (the Trailhead medium).

`connection<n>.remoteregistryport=`

Property type: Numeric
Default: 1099

This property identifies the remote port for this connection. (the Trailhead medium.) The value here must match the value entered in the `localregistryport` property on the Trailhead medium.

`connection<n>.remotepassword=`

Property type: String
Default: blenderpw

This property identifies the password for this connection. The value here must match the value indicated in the `connection<`*`n`*`>.localpassword` property on the Trailhead medium. Note that this password is automatically encrypted; when you edit this file, the value you enter here will appear differently.

`connection<n>.canroute=`

Property type: Boolean
Default: true

This property specifies whether this connection should appear on the list of connections that a DcRemote destination choose tries to use when routing a session.

`connection<n>.canqueue=`

Property type: Boolean
Default: true

This property specifies whether this connection is usable to queue a call. This connection appears in the `destination<`*`n`*`>queueconn` property on the Trailhead medium.

`connection<n>disableautoconnect=`

Property type: Boolean
Default: false

This property allows you to disable attempts by the Trailhead server to automatically connect to the Trailhead medium. Set this property to true if inbound communication is not possible through your firewall. This setting ensures that the Trailhead server does not try to automatically connect to the Trailhead medium. Instead, the Trailhead medium will instead poll the Trailhead server from behind the firewall.

If the firewall will allow both inbound and outbound communication, set this property to false. This setting ensures that the Trailhead server will connect to the Trailhead medium automatically, initiating two-way communication through the connections.

Specifying a System Error Destination

On the Trailhead server, you need set up only one destination to handle system error. Use these properties to establish a system error destintation on the Trailhead server.

`destination<n>.name=`

Property type: Valid file name
Default:

This property defines the name for a destination in your configuration. When specifying destinations, number each destination sequentially, beginning with 1.

Example:

```
destination1.name=SysError
```

`destination<n>.type=`

Property type: String
Default value:

This property lets you specify the type of response this destination will provide to web requests. Acceptable values are:

Value	**Description**
`CALLONLY`	Basic Callback
`COLLAB`	Web Collaboration
`NOCALL`	Noncalling

On the Trailhead server, you need only set up a system error destination, which should be type **NOCALL**.

`destination<`*`n`*`>.URL=`

Property type: Valid URL
Default value: none

This property maps a valid URL to a particular destination. When a request is routed to the destination, Trailhead serves the URL specified here.

We provide sample URLs for different destination types. See Maintaining Trailhead HTML Pages and Forms in this guide for more information for more information.

Note for NOCALL destinations: You can choose to server a URL here, or, instead, serve only an HTTP error code and reason to the caller. If you choose to server an error code instead, do not use this property; instead, use **destination<n>errorcode** and **destination<n>errorreason**, described below.

`destination<`*`n`*`>errorcode=`

Property type: Integer
Default: none

This property specifies an error number to send the user, rather than a URL.You can enter 404, 500, or any HTTP error code to send these errors.

`destination<n>errorreason`

Property type: String
Default: System Error. Please try later.

This property explains the error specified in destination<*n*>errorcode.

Identifying the Callback Form

The callback form is the HTML page used to gather caller information to be submitted to the ICM. Use this property to identify the callback form.

`formurl=`

Property type: String
Default value: None

Available in the general release of the product, this property identifies the callback form used to gather information about the caller. We provide a sample, `callform.html`. See ICM Input: The Web request and Maintaining HTML forms and pages in this guide for more information.

Setting up the Log

The following properties let you establish a rotation of log files generated by Trailhead. Trailhead creates new log files until it reaches the number of logs specified in the logcount property.

`logpath=`

Property type: String
Default value:none

This property identifies the pathname to which logs are written. Be sure to end the path with a forward slash.

`loglines=`

Property type: Numeric
Default value: 20000

This property sets the maximum number of lines per log file.

`logcount=`

Property type: Numeric
Default value: 2

This property indicates the number of log files in file rotation.

`verbose=`

Property type: Integer
Default value: 3

This property indicates the level of verbosity of logged messages. Acceptable values are:

1 -- Urgent
2 -- Critical
3 -- Important
4 -- Informational

`timestampfiles=`

Property type: boolean
Default: True

This property determines whether Trailhead appends log file names with the date and time of creation. Log-1999-07-31-7-12_1.log

`threadfloor=`

Property type: Integer
Default: 20

This property specifies the minimum number of threads.

`threadceiling=`

Property type: Integer
Default: 20
This property specifies the maximum number of threads.

On the Media Blender Server: The Trailhead medium (Trailhead.properties)

Setting up the Trailhead medium involves these steps:

1. Define the Trailhead medium on the Media Blender
2. Configure Trailhead properties

Define the Trailhead Medium on the Media Blender

You must identify Trailhead as a medium in the properties file for Media Blender (`blender.properties`). In the `Medium#` property, enter `Trailhead.properties`.

Note: You must have the proper Blender key code to install and use the Trailhead medium.

Configure Trailhead Properties

Properties for configuring Trailhead are in the `Trailhead.properties` file that resides in the `\CiscoMB\servlet\properties\blender` directory on the Blender server. (You identify this file to Media Blender using the Medium property in the Blender properties file.) Use this property file to determine which events the Trailhead medium can accept and share. See "Event Filters" in the *Media Blender Reference Guide* for complete information about event filter parameters.

Use each property only once in each properties file. If you include a property more than once in a file, Media Blender automatically recognizes the setting in the last occurrence of the property, overwriting the value in preceding occurrences.

In addition to CTI event filter parameters, Trailhead.properties contains these properties:

`name=`

Display only. This property identifies the medium. It should be set to Trailhead.

`package=`

This property displays the Java package name of this Blender medium. In this file, it must be set to Com.WebLine.Blender.Trailhead. This package must be accessible to the servlet engine (either JWS or servlet exec) through its CLASSPATH. See your Installation Guide for information on setting the Classpath.

`serverport=`

Property type: Numeric
Default: None
This property identifies the port Trailhead uses to connect to the Web PG.

destchooser=

Property type: string
Default: Com.WebLine.Trailhead.DcICM

Required. This property identifies the destination chooser used by the Trailhead medium on the Media Blender server. In ICM Web configurations, this should be Com.WebLine.Trailhead.DcICM

Media Blender Startup Properties

Use these properties to adjust the amount of time Media Blender should wait for the incoming Web PG connection at start up.

serverconntimeout=

Property type: numeric
Default: 120

This property specifies the amount of time Media Blender should wait for the incoming Web PG connection at start up.

serverstartwithoutconn=

Property type: Boolean
Default: true

This property specifies whether to allow Media Blender to start if the timeout specified in serverconntimeout is exceeded.

- If this property is set to true, the Media Blender will complete its own startup regardless of whether it detects the incoming Web PG connection. Since the Web PG connection will not exist, any route request messages sent to it will be routed to the System Error destination. The Trailhead medium will continue to look for a connection and will generate alerts until one is made.

- If this property is set to false, Media Blender will not start up if the Web PG connection is not made within the serverconntimeout limit.

Specifying Trailhead Connections

Use these properties to set up the connections from your Trailhead servers to your Trailhead media. Note that, even though connections are numbered sequentially, you can delete a connection within the sequence without having to renumber your remaining connections.
In the properties that follow, note that "local" properties refer to the Trailhead medium. Properties that refer to "remote" values indicate the Trailhead server.

connection<n>.localservername=

Property type: String
Default: blender

This property identifies the local name for this connection; that is, the name of the Trailhead server end of the connection. This is the name that a destination must specify in a **destination<*n*>.queueconn** property.

Note: In previous versions, the value for the Media Blender end of the connection was specified.

connection<n>.localregistryport=

Property type: Numeric
Default: 1099

This property identifies the local port for this connection (i.e. the Trailhead medium).

connection<n>.localpassword=

Property type: String
Default: blenderpw

This property identifies the password for this connection. The value here must match the value indicated in the **connection<n>.remotepassword** property on the Trailhead server. Note that this password is automatically encrypted; when you edit this file, the value you enter here will appear as an encrypted code.

connection<n>.remoteservername=

Property type: String
Default: trailhead

This property identifies the remote name for this connection; that is, the Trailhead server. The value here must match the value in the connection<n>localservername property on the Trailhead server.

connection<n>.remotehost=

Property type: String
Default: None

This property identifies the host name of the Trailhead server.

`connection<n>.remoteregistryport=`

Property type: Numeric
Default: 1099

This property identifies the remote port for this connection; that is, the Trailhead server. The value here must match the value entered in the `localregistryport` property on the Trailhead server.

`connection<n>.remotepassword=`

Property type: String
Default: trailheadpw

This property identifies the password for this connection. The value here must match the value indicated in the `connection<`*`n`*`>.localserverpassword` property on the Trailhead server. Note that this password is automatically encrypted; when you edit this file, the value you enter here will appear in an encrypted code.

Specifying Trailhead Destinations

Use these properties to set up your Trailhead destinations. Note that, even though connections are numbered sequentially, you can delete a connection within the sequence without having to renumber your remaining connections.

`destination<n>.name=`

Property type: String
Default: None

This property defines the name for a destination in your configuration. When specifying destinations, number each destination sequentially, beginning with 1.

Example:
```
destination1.name=Boston1
destination2.name=Boston2
```

`destination<n>.type=`

Property type: String
Default:

This property lets you specify the type of response this destination will provide to web requests. Acceptable values are:

Value	Description
`CALLONLY`	Basic Callback
`COLLAB`	Web Collaboration
`NOCALL`	Noncalling

`destination<`*`n`*`>.queueconn=`

Property type: string
Default: None

This property is required for CALLONLY destinations. It identifies the local server name of a connection, defined on a Trailhead server, to which queue requests are sent for this destination.

Note: This is a change from previous versions when the value of the Trailhead medium on the Media Blender end of the connection was used.

`destination<`*`n`*`>.URL=`

Property type: Valid URL
Default: None

This property maps a valid URL to a particular destination. When a request is routed to the destination, Trailhead serves the URL specified here.

Note for NOCALL destinations: You can choose to serve a URL here, or, instead, serve only an HTTP error code and reason to the caller. If you choose to serve an error code instead, do not use this property; instead, use **destination<n>errorcode** and **destination<n>errorreason**, described below.

We provide sample URLs for different destination types. See Maintaining Trailhead HTML Pages and Forms in this guide for more information.

`destination<n>.substitute=`

Property type: Boolean
Default: None

This property lets you specify whether data will be dynamically substituted into HTML fields on the URL specified for the destination. This property should be set to True for COLLAB destinations.

`destination<`*`n`*`>.errorcode=`

Property type: Integer
Default: None

This property specifies that a HTTP error code should be sent to a user when callback is unavailable or to handle crank or troublesome calls. Use this property for NOCALL destinations if you want serve users an error code rather than an HTML page explaining the problem. If you use this property, do not specify a URL in the `destination<n>.URL` property for this destination.

Acceptable values are 404, 500, or any numeric HTTP error code.

`destination<n>.errorreason`

Property type: string
DefaultNone

This property explains the error specified in `destination<`*`n`*`>errorcode`.

Setting System Defaults

The following properties let you specify default destinations that should be used when the ICM script returns any of these labels:

- ring
- busy
- default

In addition, you can set a default for when the ICM script returns an error.

`destination<`***`n`***`>.systemdefault=`

Property type: Boolean
Default: False

This property lets you specify whether the destination should be used whenever the ICM script returns a default label.

`destination<`***`n`***`>.systembusy=`

Property type: Boolean
Default: false

This property lets you specify whether the destination should be used whenever the ICM script returns a busy label.

`destination<`***`n`***`>.systemnoagents`

Property type: Boolean
Default: false

This property lets you specify whether the destination should be used whenever the ICM script returns a ring label.

`destination<`***`n`***`>.systemerror=`

Property type: Boolean
Default: false

This property lets you specify whether the destination should be used whenever the ICM script returns an error label or any error occurs.

Specifying the Trailhead Input and Output Maps

Use these properties to identify the input and output maps used in your system.

`inmap=`

Property type: Filename
Default: in.map.properties

This property identifies the file used to translate web-based data into data that can be used by the ICM scripts. See the Input Map in this guide for more information

`outmap=`

Property type: Filename
Default: out.map.properties

This property identifies the file used to translate information returned by the ICM script into textual information that can be used by the Collaboration Server. See Creating the Output Map for more information

ICM Label Information

`icmlabeldelimiter=`

Property type: Character
Default: **- (**hyphen)

This property identifies the delimiter used to separate the two distinct portions of the ICM label output by scripts.Use this property if you are not using the IcmLabelLength method of parsing ICM labels. For information on ICM script labels, see ICM Script Label Configuration in this guide.

Note: If you specify a delimiter here, do not specify a label length using the `icmlabellength` property, explained below.

`icmlabellength=`

Property type: Numeric
Default: None

This property lets you determine which portion of the ICM label should be used as a routing address based on length. For more information, see ICM Script Label Configuration in this guide.
Note: If you specify a length here, do not specify a label delimiter using the `icmlabeldelimiter` property, explained above.

Verifying URLs

`checkurls=`

Property type: Boolean
Default: False

This property determines whether Trailhead or the Trailhead Medium should check for the existence of the URLs referred to by each destination.

On the Media Blender Server: The Input Map (in.map.properties)

Trailhead's Input map (`in.map.properties`) is a text file that maps fields found on the Trailhead callback form with ICM VRU fields. The format of each entry is as follows:

```
<ICM value> = <Web value>
```

For instance, the file might contain this entry:

```
CED=Route
```

This entry specifies that the value in the RouteAddr field on the web request is equal to the DN (Dialed Number) value on the ICM system. In this way, the Input Map translates the information from the incoming web request and populates a route request message submitted to ICM.
Note that you can map literal strings to ICM variables by enclosing strings in quotes. For example, the following entry maps the literal 1000 to the DN field

```
DN="1000"
```

In this case, the string 1000 indicates a script number on the ICM. Passing a literal script number through the DN field ensures the same script is run for all web requests.

A sample input map appears below:

```
 DN="1000"
 ANI=PhoneNumber
 CED=Route
 CALLVAR1=WantsCollab
 CALLVAR2=User2
 CALLVAR3=User3
 CALLVAR4=User4
 CALLVAR5=Host
 CALLVAR6=Referer
 CALLVAR7=Accept-Language
 CALLVAR8=CallbackDelay
 CALLVAR9=User-Agent
 CALLVAR10=
```

On the Media Blender Server: The Output Map (out.map.properties)

Trailhead's Output Map (`out.map.properties`) is a text file that maps ICM data with textual data that can be recognized by the Collaboration Server. The output map repopulates the caller data with values that have been returned by the ICM script.

The format of each entry in the file is as follows:

```
<ICM value> = <Call form value>
```

For instance, the file might contain this entry:

```
DN=PhoneNum
```

This entry specifies that the value in the DN field on the ICM system is equal to the phone number field on the Collaboration Server. In this way, the Input Map translates the information from the ICM script into data that can be used by the CCS.

We provide two sample output maps:

out.map.properties:

```
DN=
ANI=Route
CED=Name
CALLVAR1=User1
CALLVAR2=User2
CALLVAR3=User3
CALLVAR4=User4
CALLVAR5=Host
CALLVAR6=Referer
CALLVAR7=Accept-Language
CALLVAR8=CallbackDelay
CALLVAR9=User-Agent
CALLVAR10=
```

callvars-out.map.properties

```
CALLVAR1=callvar1
CALLVAR2=callvar2
CALLVAR3=callvar3
CALLVAR4=callvar4
CALLVAR5=callvar5
CALLVAR6=callvar6
CALLVAR7=callvar7
CALLVAR8=callvar8
CALLVAR9=callvar9
CALLVAR10=callvar10
```

On the Media Blender Server: The Peripheral Input Map

Trailhead's peripheral input map lets you load callback form data into Cisco ECTI server call variables. To use this feature, you must set up a text file called an input map between the CTI server and the Web callback form. This file is called `cti.in.map.properties` and resides in the `\CiscoMB\servlet\properties` directory. The format of each entry is as follows:

```
<ICM variable> = <Web variable>
```

For instance, the file might contain this entry:

```
CALLVAR2=user1
```

This entry would ensure that the value in the User1 field on the callback form is passed into Call Variable 2 on the CTI server.

Note that you can set the CTI server variable to a literal string rather than a Web variable. For instance, consider the following example entry:

```
CALLVAR1="Web Call"
```

This line ensures that the words "Web Call" are passed to call variable 1.

We provide two sample CTI peripheral input maps:

cti.in.map.properties:

```
CALLVAR1="Web Call"
CALLVAR2=user1
CALLVAR3=user2
#CALLVAR4=user3
#CALLVAR5=name
#CALLVAR6=host
#CALLVAR7=referer
#CALLVAR8=callbackdelay
#CALLVAR9=
#CALLVAR10=
```

callvars-cti.in.map.properties

```
CALLVAR1=callvar1
CALLVAR2=callvar2
CALLVAR3=callvar3
CALLVAR4=callvar4
CALLVAR5=callvar5
CALLVAR6=callvar6
CALLVAR7=callvar7
CALLVAR8=callvar8
CALLVAR9=callvar9
CALLVAR10=callvar10
```

Maintaining HTML Pages and Forms

Our sample HTML pages provide for dynamic name substitution; Trailhead retrieves the caller name and number from the callback form and inserts them into the appropriate HTML page. The following HTML source code is taken from one of our sample pages, `boston1.html`.

```
<HTML>
<HEAD>
<TITLE>Expect our call! (Boston 1)</TITLE>
<SCRIPT language="JavaScript">

Function sendCall()
{
        // substitute in the callback data from form
        document.callback.invokingUrl.value="<subst data=requestfromurl/>";
        document.callback.customerFirstName.value="<subst data=name/>";
        document.callback.phoneNumber.value="<subst data=phonenumber/>";
        document.callback.RoutingAddr.value="<subst data=route/>";
        document.callback.Appl1.value="<subst data=user1/>";
        document.callback.Appl2.value="<subst data=user2/>";
        document.callback.Appl3.value="<subst data=user3/>";
        document.callback.Appl4.value="<subst data=user4/>";
        document.callback.submit();
}

</SCRIPT>

</HEAD>
```

Note: All Trailhead forms must be served by the Trailhead server; this ensures proper substitution of the invoking URL.

HTML File Location

The directories `\<CCS dir>\pub\html\Forms\` on the Cisco Collaboration Server and `\CiscoTH\pub\html\Forms\` on the Trailhead server contain samples of HTML pages and forms that you can modify to suit your needs. You should, however, be familiar with HTML if you intend to make changes. Changes you make here can affect the system functionality.
If you do make extensive changes to these forms, copy the modified forms to another location on the server, but not in the Media Blender tree. Then, if you decide to upgrade Media Blender, you will not lose your customized forms.

Note: Always make a backup copy before altering an HTML page.

This section describes the Trailhead callback form and the HTML pages used by Trailhead to respond to different scenarios. It includes these sections:

- The Callback form
- The Web Collaboration HTML page
- The Basic Callback HTML page
- Pages for noncalling destinations

The Trailhead Start Page

We provide a sample page that links to the Trailhead server: **launchTrailhead.html**. You can use this sample when designing how your Web site will provide access to Trailhead. The link to the Trailhead server is shown below:

```
http://<servername>/servlet/Trailhead
```

The Callback Form

The callback form is served to a caller who has requested callback from the Web. This form provides Trailhead with information about the caller, such as the caller's name and phone number, as well as the code to which the call should be routed.

The callback form should be accessed indrectly through the Trailhead servlet, using an HTML link containing the URL: `http://<servername>/servlet/Trailhead`. This way, the Web page containing the link (the callback button) will be accessible to the Collaboration server.

The callback HTML forms used in Trailhead configurations should be served by the Trailhead server. If a Web site just points to the form without getting the substituted form from Trailhead, then the site will not obtain the correct invoking URL; that is, the URL from which the caller requested callback.

The fields on the callback form are described in detail in ICM Input: The Web request.

The Web Collaboration Pages

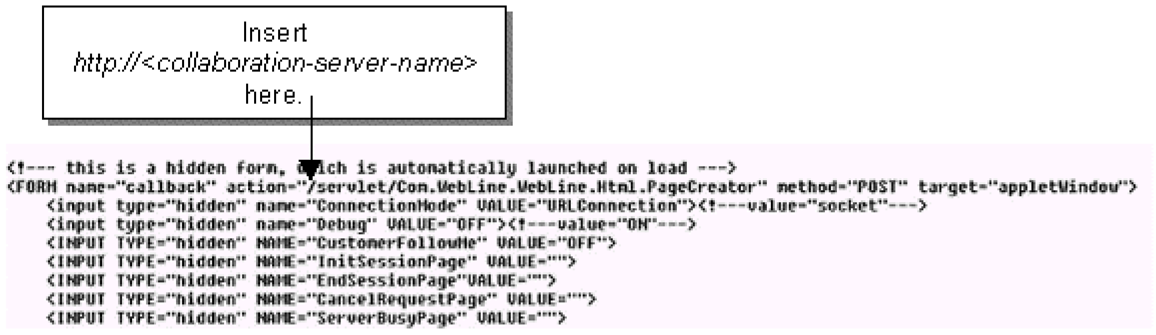
You must maintain an HTML page to be served to callers whose requests have been routed to a Web Collaboration destination. We provide a sample page: `success-v30.html,` explained below.

`success-v30.html`

This sample page is designed to be served to a caller who has successfully submitted a request for blended Collaboration. This page launches a Collaboration Server session.

Note that this form retrieves the caller's name and phone number from callform.html, placing it in the text.

If the Collaboration Server is running on a different computer than Trailhead, you must edit whichever success file you are using so that it points to the correct Collaboration server. A hidden form appears at the end of the HTML source code of both of these files. You specify the CCS server name at the beginning of this form, using the action field. Insert `http:<`*`collaboration-server-name`*`>` immediately before `/servlet/Com.WebLine.WebLine.Html.PageCreator`, as shown below:



```
<!--- this is a hidden form, which is automatically launched on load --->
<FORM name="callback" action="/servlet/Com.WebLine.WebLine.Html.PageCreator" method="POST" target="appletWindow">
    <input type="hidden" name="ConnectionMode" VALUE="URLConnection"><!---value="socket"--->
    <input type="hidden" name="Debug" VALUE="OFF"><!---value="ON"--->
    <INPUT TYPE="hidden" NAME="CustomerFollowMe" VALUE="OFF">
    <INPUT TYPE="hidden" NAME="InitSessionPage" VALUE="">
    <INPUT TYPE="hidden" NAME="EndSessionPage"VALUE="">
    <INPUT TYPE="hidden" NAME="CancelRequestPage" VALUE="">
    <INPUT TYPE="hidden" NAME="ServerBusyPage" VALUE="">
```

Other Collaboration Samples

We include several other examples of HTML pages you might set up to provide Web Collaboration. They are:

- `boston1.html`
- `boston2.html`
- `sanjose1.html`
- `sanjose2.html`
- `callvars-sanjose1.html` (See Passing data to the Cisco ECTI server in this guide for more information.)

The Basic Callback Page

You must maintain HTML pages that will be served to callers who will receive Basic callback in response to a Web request. We provide several samples of Basic Callback pages:

- `bostoncallonly.html`
- `sanjosecallonly.html`

Note that these forms retrieve the caller's name and phone number from `callform.html`, and place it in the text.

A Collaboration session is not initiated; instead, Trailhead stores each request and Media Blender then retrieves the requests and queues them to the switch.

Pages for Noncalling Destinations

Sometimes callers request callback during hours when the phone system is down. We provide sample HTML pages you can use to design your own pages to cover these instances.

For instance, you may alter these pages so that they request user information; that way, you can retrieve valuable information about potential customers even when callback is prohibited.

`afterhours.html`

This sample Web page is designed to be served to callers who place calls outside of your call center's normal hours of operation.

`sysnoagents.html`

This sample page is designed to be served to the caller when no agents are available for callback.

`syserror.html`

This sample page informs the caller that a technical problem prohibits response.

`sysbusy.html`

The sample page informs the caller that all agents are busy.

`holiday.html`

This sample page informs the caller that the contact center is closed due to a holiday.

Index