



Using Extension Points

This chapter describes how to use Cisco Access Registrar scripting to customize your RADIUS server. You can write scripts to affect the way Cisco Access Registrar handles and responds to requests, and to change the behavior of Cisco Access Registrar after a script is run.

All scripts reference the three dictionaries listed below, which are data structures that contain key/value pairs.

- Request dictionary—contains all of the attributes from the access-request or other incoming packets, such as the username, password, and service hints
- Response dictionary—contains all of the attributes in the access-accept or other response packets. As these are the attributes the server sends back to the NAS, you can use this dictionary to add or remove attributes.
- Environment dictionary—contains well-known keys whose values enable scripts to communicate with Cisco Access Registrar or to communicate with other scripts.

The process for creating and implementing a script involves:

- Determining the goal of the script
- Writing the script
- Adding the new script definition to Cisco Access Registrar
- Choosing a scripting point from within Cisco Access Registrar
- Testing the script using the **radclient** command.

Determining the Goal of the Script

The goal of the script and its scripting point are tied together. For example, when you want to create a script that performs some special processing of a username before it is processed by the Cisco Access Registrar server, you would reference this script as an *incoming* script.

When on the other hand, you would like to affect the response, such as setting a specific timeout when there is not one, you would reference the script as an *outgoing* script.

In order to be able to create a script, you need to understand the way Cisco Access Registrar processes client requests. Cisco Access Registrar processes requests and responses in a hierarchical fashion; incoming requests are processed from the most general to the most specific levels, whereas, outgoing responses are processed from the most specific to the most general levels. Extension points are available at each level.

An incoming script can be referenced at each of the following extension points:

- RADIUS server
- Vendor (of the immediate client)
- Client (individual NAS)
- NAS-Vendor-Behind-the-Proxy
- Client-Behind-the-Proxy
- Remote Server (of type RADIUS)
- Service

An authentication or authorization script can be referenced at each of the following extension points:

- Group Authentication
- User Authentication
- Group Authorization
- User Authorization

The outgoing script can be referenced at each of the following extension points:

- Service
- Client-Behind-the-Proxy
- NAS-Vendor-Behind-the-Proxy
- Client (individual NAS)
- NAS Vendor
- RADIUS server

Writing the Script

You can write scripts in either Tcl or as shared libraries using C or C++. In this section, the scripts are shown in Tcl.

To write a script, do the following:

-
- Step 1** Using an editor, create the Tcl source file.
 - Step 2** Give it a name.
 - Step 3** Define one or more procedures, using the following syntax:


```
proc name {request response environment}
{Body}
```
 - Step 4** Create the body of the script.
 - Step 5** Save the file and copy it to the `/opt/CSCOAr/scripts/radius/tcl` directory, or to the location you chose when you installed Cisco Access Registrar.

Choosing the Type of Script

When you create a script you can use any one or all of the three dictionaries: Request, Response, or Environment.

- When you use the Request dictionary, you can modify the contents of a NAS request. Scripts that use the Request dictionary are usually employed as incoming scripts.
- When you use the Response dictionary, you can modify what the server sends back to the NAS. These scripts are consequently employed as outgoing scripts.
- When you use the Environment dictionary, you can do the following:
 - Affect the behavior of the server after the script is run. For example, you can use the Environment dictionary to decide which of the multiple services to use for authorization, authentication, and accounting.
 - Communicate among scripts, as the scripts all share these three dictionaries. For example, when a script changes a value in the Environment dictionary, the updated value can be used in a subsequent script.

The following examples show scripts using all three dictionaries.

Request Dictionary Script

The Request Dictionary script is referenced from the server's IncomingScript scripting point. It checks to see whether the request contains a **NAS-Identifier** or a **NAS-IP-Address**. When it does not, it sets the **NAS-IP-Address** from the request's source IP address.

```
proc MapSourceIPAddress {request response environment}
{
    if { ! ( [ $request containsKey NAS-Identifier ] ||
            [ $request containsKey NAS-IP-Address ] ) } {
        $request put NAS-IP-Address [ $environment get Source-IP-Address ]
    }
}
```

Tcl scripts interpret **\$request** arguments as active commands that can interpret strings from the Request dictionary, which contains keys and values.

The **containsKey** method has the syntax: **<\$dict> containsKey <attribute>**. In this example, **<\$dict>** refers to the Request dictionary and the attributes **NAS-identifier** and **NAS-IP-Address**. The **containsKey** method returns **1** when the dictionary contains the attribute, and **0** when it does not. Using the **containsKey** method prevents you from overwriting an existing value.

The **put** method has the syntax: **<\$dict> put <attribute value>[<index>]**. In this example, **<\$request>** refers to the Request dictionary and the attribute is **NAS-IP-Address**. The **put** method sets the NAS's IP address attribute.

The **get** method has the syntax: **<\$dict> get <attribute>**. In this example, **<\$dict>** refers to the Environment dictionary and **<attribute>** is the **Source-IP-Address**. The **get** method returns the value of the attribute from the environment dictionary.

For a list of the methods you can use with scripts, see [Appendix A, "Cisco Access Registrar Tcl and REX Dictionaries."](#) They include **get**, **put**, and others.

Response Dictionary Script

This script is referenced from either the user record for users whose sessions are always PPP, or from the script, **AuthorizeService**, which checks the request to determine which service is desired. The script merges the Profile named **default-PPP-users** into the Response dictionary.

```
proc AuthorizePPP {request response environment}
{
    $response addProfile default-PPP-users
}
```

The **addProfile** method has the syntax: `<$dict> addProfile <profile>[<mode>]`. In this example, `<$dict>` refers to the Response dictionary and the profile is **default-PPP-users**. The script copies all of the attributes of the Profile `<profile>` into the dictionary.

Environment Dictionary Script

This script is referenced from the NAS Incoming Script scripting point. It looks for a realm name on the username attribute to determine which AAA services should be used for the request. When it finds `@radius`, it selects a set of AAA services that will proxy the request to a remote RADIUS server. When it finds `@tacacs`, it selects the Authentication Service that will proxy the request to a TACACS server for authentication. For all of the remaining usernames, it uses the default Service (as specified in the configuration by the administrator).

Note the function, **regsub**, is a Tcl function.

```
proc ParseProxyHints {request response environment} {
    set userName [ $request get User-Name ]
    if { [ regsub "@radius" $userName "" newUser ] } {
        $request put User-Name $newUser
        $radius put Authentication-Service "radius-proxy"
        $radius put Authorization-Service "radius-proxy"
        $radius put Accounting-Service "radius-proxy"
    } else {
        if { [ regsub "@tacacs" $userName "" newUser ] } {
            $request put User-Name
            $radius put Authentication-Service "tacacs-client"
        }
    }
}
```

Adding the Script Definition

After you have written the script, you must add the script definition to the Cisco Access Registrar's script Configuration directory so it can be referenced. Adding the script definition involves:

- Specifying the script definition; it must include the following:
 - **Name**—used in other places in the configuration to refer to the script. It must be unique among all other scripts.
 - **Language**—can be either Tcl or REX (shared libraries)
 - **Filename**—the name you used when you created the file
 - **EntryPoint**—the function name.

The **Name** and the **EntryPoint** can be the same name, however, they do not have to be.

- Choosing a scripting point; nine exist for incoming and outgoing scripts. These include:
 - the server
 - the vendor of the immediate client
 - the immediate client
 - the vendor of the specific NAS
 - the specific NAS
 - the service (only type rex)
 - the group (only AA scripts)
 - the user record (only AA scripts)
 - remote server (only type RADIUS)

The rule of thumb to use in determining where to add the script is the most general scripts should be on the outermost points, whereas the most specific scripts should be on the innermost points.

**Note**

The client and the NAS are the same entity, unless the immediate client is acting as a proxy for the actual NAS.

Adding the Example Script Definition

In the server configuration a **Scripts** directory exists. You must add the script you created to this directory. To add the **ParseProxyHints** script to the Cisco Access Registrar server, type the following command and supply the following information:

```
Name=ParseProxyHints
Description=ParseProxyHints
Language=tcl
Filename=ParseProxyHints
Entrypoint=ParseProxyHints
```

```
aregcmd add /Radius/Scripts/ParseProxyHints ParseProxyHints tcl ParseProxyHints
ParseProxyHints
```

Choosing the Scripting Point

As the example script, **ParseProxyHints**, applies to a specific NAS (NAS1), the entry point should be that NAS. To specify the script at this scripting point, type:

```
aregcmd set /Radius/Clients/NAS1/IncomingScript ParseProxyHints
```

Testing the Script

To test the script, you can use the **radclient** command, which lets you create and send packets. For more information about the **radclient** command, see [Chapter 2, “Using the aregcmd Commands.”](#)

About the Tcl/Tk 8.3 Engine

Cisco Access Registrar 1.6 and above uses Tcl engine is version Tcl/Tk8.3. Since the Tcl/Tk8.3 engine supports a multi-threading application environment, it can achieve much better performance than Tcl/Tk7.6.

**Note**

In this release, scripts that use Tcl global variables will not work across AR extension points. A future release will address script compatibility issues.

Tcl/Tk8.3 also performs *byte-compile*, so no run-time interpretation is required.