

Cisco Access Registrar Customization Techniques

Cisco Access Registrar is a feature-rich, RADIUS-compliant, access policy server. Cisco Access Registrar is also a highly customizable application that allows service providers to meet those access needs not already built into the product. This white paper discusses these customizable capabilities, including:

- A flow-through configuration interface that enables service providers to automate configurations and integrate with the operations support system (OSS)
- Extension point scripting (EPS) that allows Cisco Access Registrar users to interact with request processing and communicate with Cisco Access Registrar
- Support for writing custom authentication, authorization, and accounting (AAA) services when the built-in services such as Lightweight Directory Access Protocol (LDAP), Open Database Connectivity (ODBC), local, and proxy AAA services do not meet service provider needs

This paper discusses each of these customization techniques in detail, including a description of key concepts and examples showing how to use the feature.

Although this paper constitutes a very high-level discussion of Cisco Access Registrar, if you need more detailed information, refer to the documentation links at:

<http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/index.htm>.



Introduction

A major challenge facing all classes of service providers is to develop and maintain a AAA infrastructure that can efficiently serve an increasingly diverse mix of access services, users, and partners. AAA systems have to be able to keep up with changes in services being sold, users subscribing to those services, and access servers managing the connections for new technologies.

Adding to this complexity is the changing nature of who is providing services because of ongoing consolidations and mergers, emerging broker services, and shifting wholesale/retail relationships that result in multivendor, heterogeneous AAA environments with increasingly complex business models.

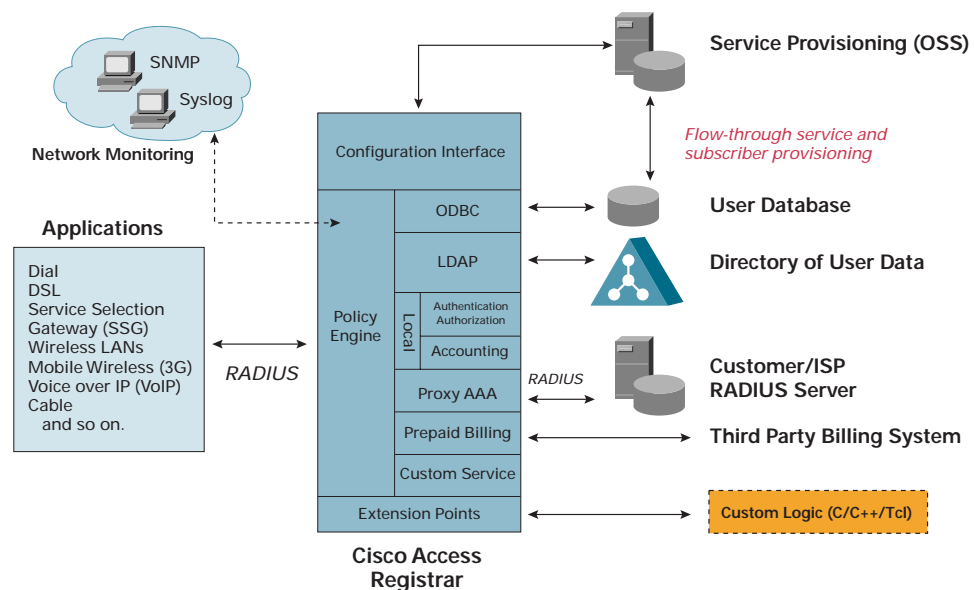
Service providers also have to keep up with the need to integrate the OSS, centralize data stores, and adapt billing systems to keep pace with all the changes.

How Cisco Access Registrar Can Help

Given the rapid rate and scope of change in the service provider AAA environment, successful service providers have had to balance the benefits of developing and maintaining their own proprietary AAA servers with the reality that the AAA world is too complex to be effectively and efficiently managed with in-house resources. To be successful, service providers need to keep their resources focused on their core business rather than AAA server development.

Cisco Systems, a leader in providing networking solutions for service providers, fully understands these issues. Cisco Access Registrar is Cisco's AAA solution for service providers. Cisco Access Registrar is designed as a comprehensive, out-of-the-box, standards-based, AAA system. It also can be customized to meet special AAA needs. Cisco Access Registrar users get the best of both worlds—support for most Remote Authentication Dial-In User Service (RADIUS) server needs, and customization capability for the rest. Figure 1 shows the standard features of Cisco Access Registrar as well as its customization support.

Figure 1 Cisco Access Registrar





The figure shows the configuration interface that service providers can use to customize OSS functions by using automated scripts and OSS integration. Service providers can launch extension point scripts during request/response cycles to change the results of the program flow. Finally, service providers can create their own custom services to replace or add to the built-in services such as LDAP and ODBC.

These three customization features provide the scalability (by allowing support for frequent and numerous configuration changes), power (by supporting the ability to manipulate each request), and functionality (by allowing support for new technologies) that service providers require without sacrificing flexibility and simplicity.

Automating Configurations

The *aregcmd* utility is a configuration tool that allows users to set Cisco Access Registrar configurable options in either interactive or noninteractive modes. This section focuses on the noninteractive mode. From the noninteractive mode, administrators and external programs (as well as an OSS) can run single commands, or multiple commands from a file.

By using individual *aregcmd* commands or by creating batches of *aregcmd* commands, service providers can customize their RADIUS server environment. Using individual *aregcmd* commands allows service provider support personnel to make configuration changes quickly. The benefit of using batch files of *aregcmd* commands is that service providers can simplify the repetitive configuration tasks by creating a file to do these tasks, and then reuse the file as often as necessary. The time-consuming tasks of creating new users, creating new RADIUS clients, or resetting passwords, for example, can be managed by creating configuration files and storing them for future use.

Batch files are also useful for creating AAA scenarios and testing them prior to deployment. Finally, single commands and batch files simplify the central configuration tasks of integrating OSS with multiple Cisco Access Registrar servers.

For more information on using *aregcmd* commands, see the Cisco Access Registrar documentation at:

<http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/index.htm>.

Example

This section illustrates using individual *aregcmd* commands, as well as batches of configuration commands in a file that is launched using the *aregcmd -f <filename>* command.

The following example uses the *aregcmd -f* command to add the user Bob to the cisco.com user list with the password *car30ispower* by placing the following into a file named *newuser*:

```
cd /radius/userlists/cisco.com
add bob
cd bob
set password car30ispower
```

Entering at the command prompt:

```
aregcmd -f newuser
```



results in producing the Cisco Access Registrar user known as Bob:

```
[ //localhost/Radius/UserLists/cisco.com/bob ]
  Name = bob
  Description =
  Password = <encrypted>
  AllowNullPassword = FALSE
  Enabled = TRUE
  Group~ =
  BaseProfile~ =
  AuthenticationScript~ =
  AuthorizationScript~ =
  UserDefined1 =
```

Alternately, you can add the user Bob by entering into a file:

```
add /Radius/Userlists/cisco.com/bob "" car30ispower
```

Notice that the double quotes ("") represent an empty Description.

An example of OSS executing the configuration change directly by running aregcmd follows:

```
aregcmd add /Radius/Userlists/cisco.com/bob "" car30ispower
```

Other examples of using aregcmd directly or in batch files to update configurations follow:

- To reset a user password using the aregcmd command directly:

```
aregcmd set /Radius/Userlists/cisco.com/bob/password mynewpassword
```

- To reset numerous user passwords, place the usernames and new passwords in a file and process the file using the aregcmd -f <filename> command. For example, to reset the user passwords for Bob, Sergio, and Pam, at Cisco, enter the following into a file named *newpass*:

```
set /Radius/Userlists/cisco.com/bob/password mynewpassword
set /Radius/Userlists/cisco.com/sergio/password mynewpassword
set /Radius/Userlists/cisco.com/pam/password mynewpassword
```

To execute the batch file, enter:

```
aregcmd -f newpass
```

- To add a RADIUS client using the aregcmd command directly:

```
aregcmd add /Radius/Clients/TokyoNAS01 "main Tokyo NAS" 10.0.0.1 mysharedsecret
```

- To add numerous new RADIUS clients, place the client name information in a file and process the file using the aregcmd -f <filename> command. For example, to add new RADIUS clients for Tokyo, London, and Paris, enter the following into a file named *newradius*:

```
add /Radius/Clients/TokyoNAS01 "main Tokyo NAS" 10.0.0.1 mysharedsecret
add /Radius/Clients/LondonNAS01 "main London NAS" 172.0.0.10 mysharedsecret
add /Radius/Clients/ParisNAS01 "main Paris NAS" 192.1.0.1 mysharedsecret
```

To execute the batch file, enter:

```
aregcmd -f newradius
```



For more information about batchable configuration files, refer to the samples included with your Cisco Access Registrar installation. These files are located at:

```
<CAR DIR>/examples/cli
```

Extension Point Scripting

To process RADIUS requests, service providers have to apply complex business rules that go beyond simply checking the user Domain Name System (DNS) domain, dialed number, or calling number. Occasionally, numerous values must be checked or a special parsing of the username is required. Service providers have a problem meeting this need: they have to be able to access the request, make the necessary changes, and allow the request-response cycle to complete—in real time.

Cisco Access Registrar solves this problem with EPS. EPS allows service providers to examine, change, or delete attributes in the request. For example, service providers can identify and modify username suffixes or prefixes as necessary. Usernames can be analyzed for illegal characters and reformatted.

In addition to being able to access attributes in the request and response, service providers can use EPS to communicate with Cisco Access Registrar at predefined points during packet processing by accessing the Cisco Access Registrar environment variables.

Program Flow

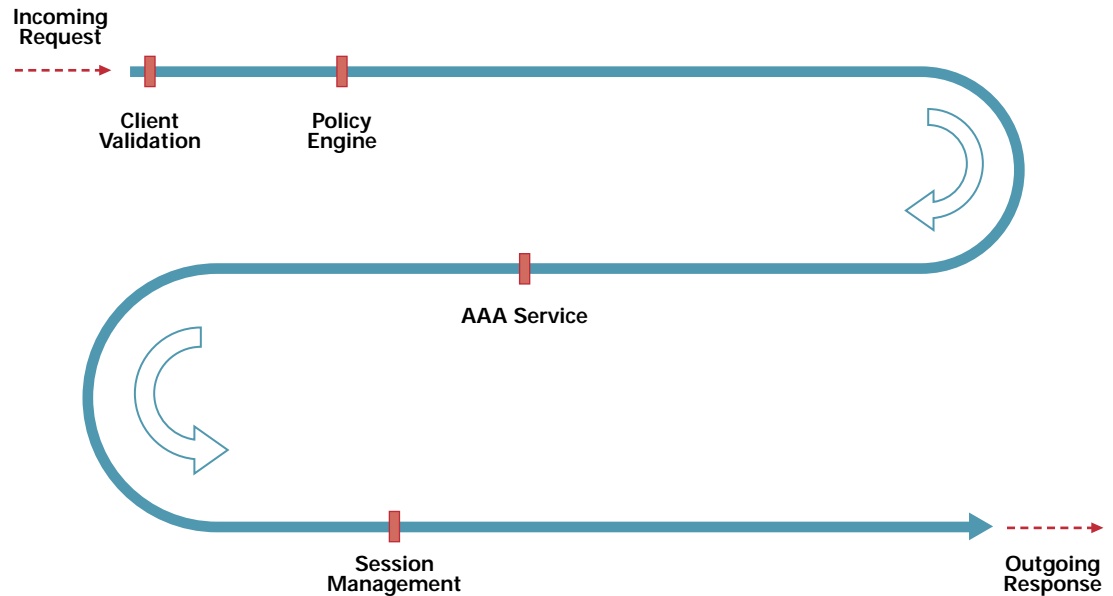
When Cisco Access Registrar receives a request from a network access server (NAS), for example, it processes the request in the following manner:

1. Client validation—Cisco Access Registrar server checks /Radius/Clients to see if the client IP address is listed.
2. Policy engine—Cisco Access Registrar invokes the policy engine if it is configured.
3. AAA—Cisco Access Registrar performs the configured AAA service (which is either an authentication/authorization service or an accounting service).
4. Session management—Cisco Access Registrar performs session management if it is configured.
5. Response—Cisco Access Registrar creates the response (Access-Accept, for example), and sends it back to the client.

Figure 2 shows the program flow.



Figure 2 Cisco Access Registrar Program Flow



For more detailed information on how Cisco Access Registrar processes, refer to the section of the user documentation titled “Program Flow” at:

<http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/index.htm>.

Extension Points

At specific points during a Cisco Access Registrar Request-Response program flow, service providers can initiate scripts to customize or modify the program flow. When the script finishes, the program flow continues with the next step. These extension points include the following:

- Server incoming/outgoing—Scripts run for every request packet.
- Vendor incoming/outgoing—Scripts run only for requests from the specified client vendor.
- Client incoming/outgoing—Scripts run only for requests from the specified client (router, NAS, and so on)
- Service incoming/outgoing—Scripts run before/after a service.
- Remote server incoming/outgoing—Scripts run for requests using a Cisco Access Registrar remote server.
- Group authentication/authorization—Scripts run for access requests for a user belonging to a group.
- User authentication/authorization—Scripts run for access requests for a particular user.

Notice that EPS allows request targeting. For example, a script applied at the server incoming extension point is invoked for all requests arriving at Cisco Access Registrar, whereas a script applied at the client incoming extension point is invoked only for requests from the specified client. The outgoing scripting points work in the same fashion.

Requests are either access requests or accounting requests. The program flow is the same for both types of requests, but the extension points invoked depend on the request type and service used. For example, an access request using the local Cisco Access Registrar authentication/authorization service is illustrated in Figure 3.



Figure 3 Cisco Access Registrar Authentication/Authorization Service

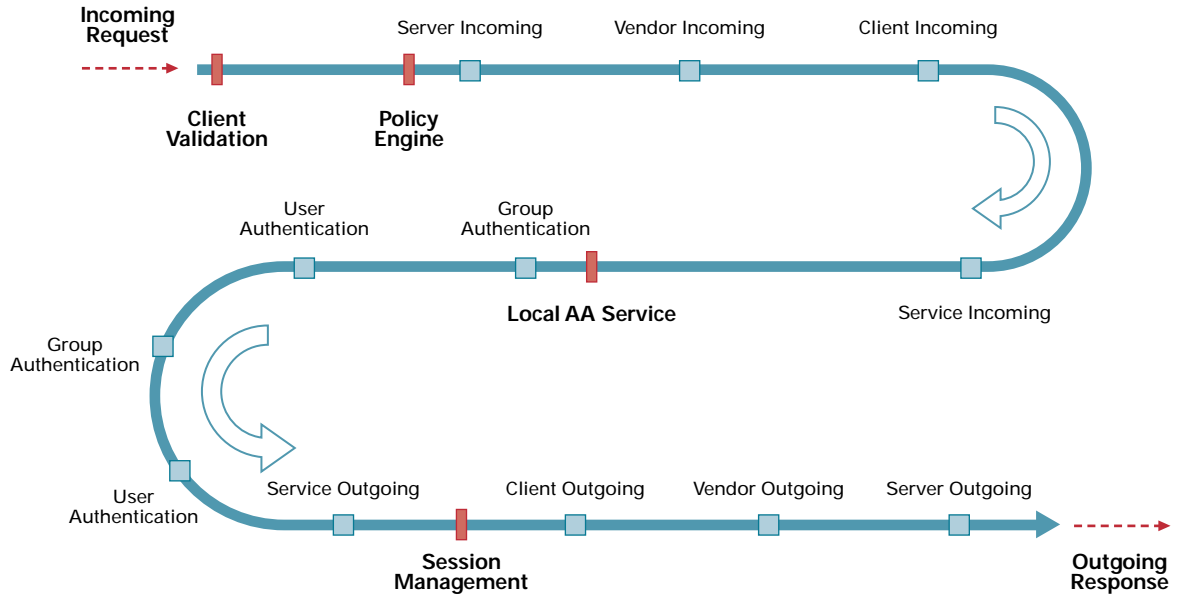


Figure 4 shows an accounting request being processed locally to a flat file.

Figure 4 Local Processing of Accounting Request

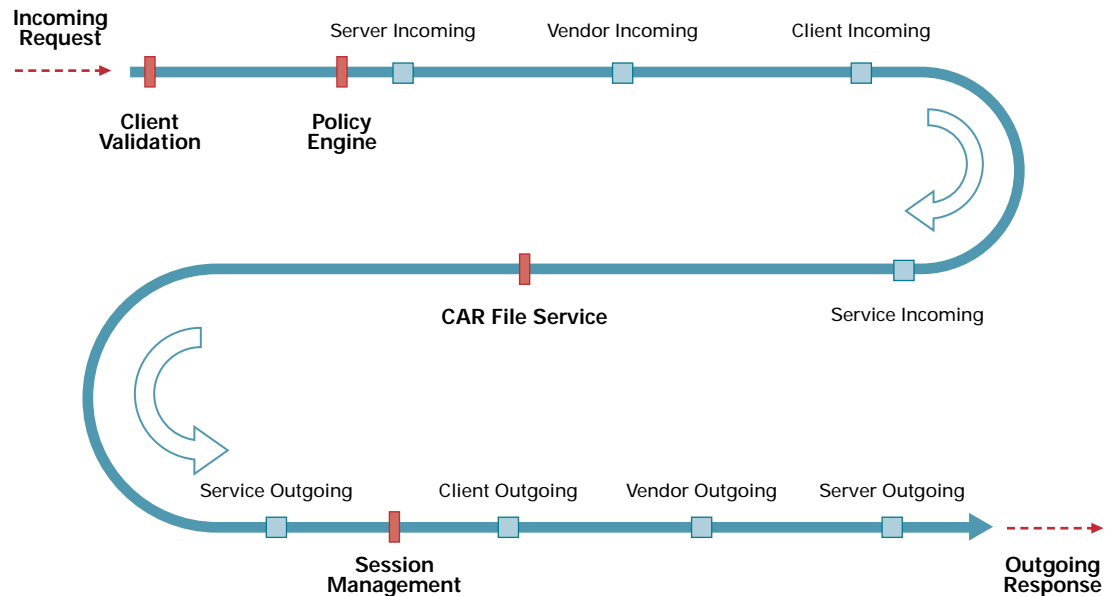
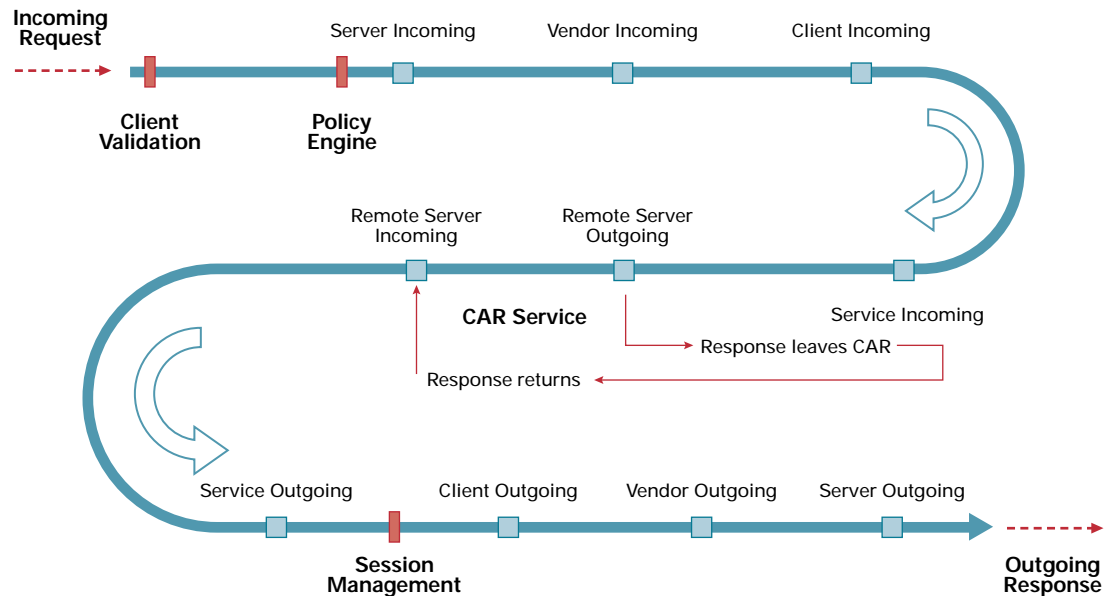




Figure 5 shows a request (access or accounting) being forwarded to a remote server.

Figure 5 Request Forwarded to Remote Server



Note that the Cisco Access Registrar Service program flow point is displayed as a box (rather than as a single line) to show that the remote processes take place during the service.

Using the Application Program Interface

EPS allows you to manipulate the request and response attributes using the Cisco Access Registrar application programming interface (API) functions. This is useful for changing information contained in the request or the response. Manipulating environment variables also allows EPS to communicate with Cisco Access Registrar as well as with other scripts. These values are grouped into: request attributes (which deal with requests), response attributes (which deal with responses), and environment variables. Examples of environment variables are discussed in the next section.



Environment Variables

Cisco Access Registrar supports many variables that scripts use to communicate with—or change the behavior of—the Cisco Access Registrar server, or to communicate with other scripts. For example, if the “Accounting-Service” environment variable is set, the server directs the request to the specified accounting service for processing. Likewise, you can set the “Authentication-Service” and “Authorization-Service” environment variables to direct requests to the appropriate authentication/authorization service.

Other examples of environment variables follow:

- You can set the “Ignore-Accounting-Signature” environment variable to tell the server not to verify the accounting request signature. This was done to work around those RADIUS implementations that did not sign accounting requests.
- Cisco Access Registrar sets the “Request-Type” environment variable and the script reads it to determine the type of request arriving (for example, access request or accounting request).
- You can set the “Response-Type” environment variable to the type of response (for example, Access-Reject).
- An extension point script can use the “Trace-Level” environment variable to set the debugging trace level. For a complete listing of the environment variables and a full description of their usage, refer to the user documentation at: <http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/index.htm>.

Extension Point Scripting Examples

The EPS examples shown here are written in Toolkit Command Language (TCL) for the purpose of clarity. All examples can be written in C or C++, and they all use the same APIs.

Basic Examples

The four examples shown in this section illustrate such basic API commands such as put, get, and remove.

The following request attribute example adds the attribute Service-Type to a request with its value set to Outbound:

```
$request put Service-Type Outbound
```

This response attribute example removes the State attribute from the Cisco Access Registrar response (some noncompliant RADIUS clients behave unpredictably when they receive this attribute):

```
$response remove State
```

The following environment variable example illustrates communicating with Cisco Access Registrar to set AAA services:

```
$environ put Authentication-Service auth-service  
$environ put Authorization-Service auth-service  
$environ put Accounting-Service acc-service
```

The following environment variable example returns the type of request received by Cisco Access Registrar; for example, Access-Request:

```
$environ get Request-Type
```



Selecting the Service

The following example illustrates how an extension point script looks at the DNS domain in the username and sets the AAA services. It also overrides the username in the request by setting the User-Name environment variable. The script can be run from the server incoming extension point so that it runs for every request that Cisco Access Registrar processes.

```
proc RealmServiceSelection {request response environ} {
    set userName [ $request get User-Name ]
    if { [ regexp {([^@]+)@([^@]+)} $userName dummy newUserName realm ] } {
        $environ put User-Name $newUserName
        $environ put Authentication-Service $realm
        $environ put Authorization-Service $realm
        $environ put Accounting-Service $realm
    }
}
```

Handling Accounting Requests

The following example shows how an extension point script diverts accounting requests to a file.

To find all the accounting requests, the script must process every request. Hence, the script runs from the server incoming extension point. The script identifies accounting requests by examining the Request-Type environment variable. All accounting requests with the attribute Acct-Status-Type set to Accounting-On or Accounting-Off are then written to the local file accounting service named SysAcc-file.

```
proc divert-AccOnOff { request response environ } {
    set request-type [$environ get Request-Type]
    if { [string equal ${request-type} "Accounting-Request" ] } {
        set AST [ $request get Acct-Status-Type ]
        if { ( [string equal $AST "Accounting-On" ] ||
            [string equal $AST "Accounting-Off" ] ) } {
            $environ put Accounting-Service SysAcc-file
        }
    }
}
```



Ignoring Accounting Signatures

This example script checks the Request-Type environment variable. If the packet is an accounting request packet, the script sets the Ignore-Accounting-Signature environment variable to TRUE. This script is useful in situations where a service provider has NASs that do not sign the accounting requests properly. This script allows the service provider to tell Cisco Access Registrar to ignore the accounting signature.

Because these NASs can be grouped under a vendor, this script can be run from the vendor incoming extension point.

```
proc ig-acc-sig { request response environ } {
    set request-type [ $environ get Request-Type ]
    if { [string equal ${request-type} "Accounting-Request" ] } {
        $environ put Ignore-Accounting-Signature TRUE
    }
}
```

Controlling Debugging

In service provider environments with large volumes of production traffic, it is not feasible to turn on debugging (tracing) for long periods of time because debugging is resource intensive. Specifically, in attempting to analyze data, debugging overhead slows request processing and uses large amounts of disk space. EPS offers an easy and elegant way to turn debugging on and off selectively.

In the following example, EPS turns debugging on only for requests with username bob@cisco.com. The environment variable Trace-Level sets the trace level, and Retrace-Packet makes Cisco Access Registrar display the full contents of the request received. This script could be set at the server incoming extension point.

```
proc debug { request response environ } {
    set user [ $request get User-Name ]
    if { [ string equal $user "bob@cisco.com" ] } {
        $environ put Trace-Level 5
        $environ put Retrace-Packet TRUE
    }
}
```



Filtering Attributes

The following example illustrates how to use EPS to validate a response from a RADIUS server. In this example, the remote RADIUS server is not in your administrative control, so you need to validate its responses. Here, the script looks in the response for the Framed-IP-Address attribute value of 255.255.255.255. If the script finds this value, it removes the attribute.

To target a specific RADIUS server, use the remote server incoming extension point to target the response from that server:

```
proc remove-bad-IP {request response environ} {
    if { [ $request containsKey Framed-IP-Address ] &&
        [ string equal [ $request get Framed-IP-Address ]
          "255.255.255.255" ] } {
        $request remove Framed-IP-Address
    }
}
```

To configure this example, create a file containing the TCL code. Name the file remove-bad-IP.tcl. Then, in aregcmd enter:

```
cd /Radius/Scripts
add remove-bad-IP
cd remove-bad-IP
set Language tcl
set Filename <yourscripts dir>/remove-bad-IP.tcl

cd /Radius/RemoteServers
cd myRemoteServer
set IncomingScript remove-bad-IP

save
reload
```

For more information on EPS scripting examples, refer to the samples that are part of your Cisco Access Registrar installation at:

For EPS examples in C:

```
<CAR DIR>/examples/rexscript/rexscript.c
```

For EPS examples in TCL:

```
<CAR DIR>/scripts/radius/tcl/tclscript.tcl
```

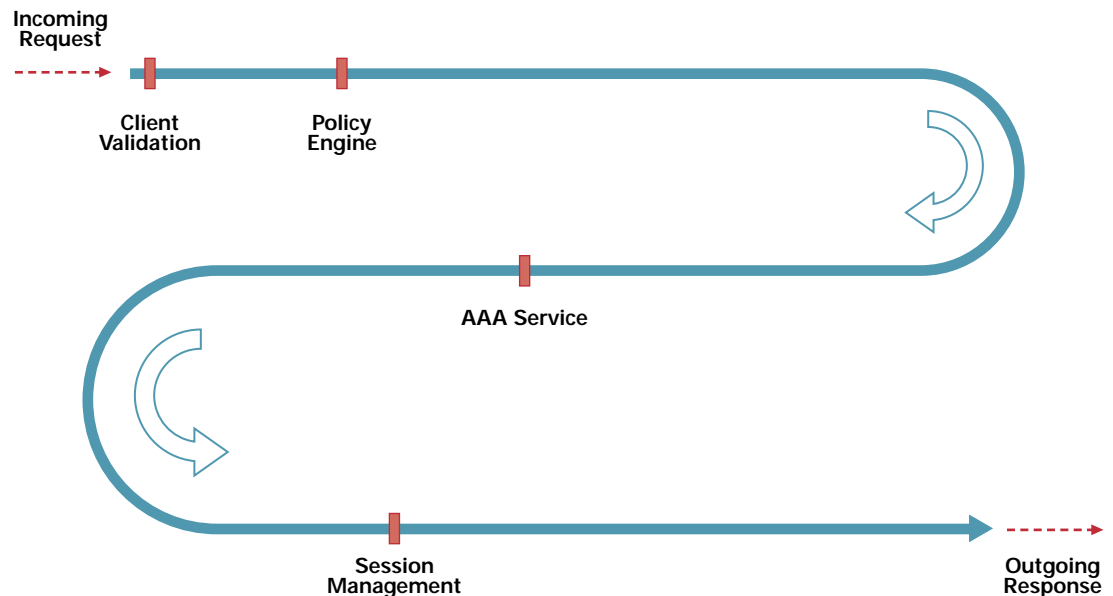


Creating Custom Services

In cases where you need a service not provided directly by Cisco Access Registrar (such as those shown in the block diagram in “How Cisco Access Registrar Can Help”), you can create your own C++ custom service using the Cisco Access Registrar API functions. (These are the same API functions used in EPS discussed in the previous section.)

Custom services are run from the program flow point labeled “AAA Service” as shown in Figure 6.

Figure 6 Custom Services



Also, refer to the `rex.h` file found at `<Cisco Access Registrar DIR>/examples/rexservice/` for more information on service points.

Custom services are written as C/C++ programs, and as such, have the ability to return values to the Cisco Access Registrar program flow. For example, a custom service can return `REX_PENDING`. This return value indicates that the service is taking over responsibility for the request and later will call the `reschedule()` function to let the server know when the service is done with the request, and the server can resume processing. This allows the service to pass the request to a different thread for background processing, freeing the current thread. This feature is useful if the service is accessing another process or server that will take time to return an answer.



Creating a Custom Service

This section describes some examples of custom services. For more information on custom service examples, refer to the sample that is included as part of your Cisco Access Registrar installation. The sample is located at:

```
<CAR DIR>/examples/rexservice/rexservice.cpp
```

In these examples, REX refers to the Cisco RADIUS Extension.

A Basic Custom Service

The following example illustrates the framework for a minimal access request service named BlindAA:

```
int REXAPI BlindAA( int iServicePoint,
                   rex_AttributeDictionary_t* pRequest,
                   rex_AttributeDictionary_t* pResponse,
                   rex_EnvironmentDictionary_t* pEnviron )
{
    int iResult = REX_OK;
    switch (iServicePoint)
    {
        case REX_START_SERVICE:
        case REX_STOP_SERVICE:
            break;
        case REX_AUTHENTICATION_SERVICE:
        case REX_AUTHORIZATION_SERVICE:
        case REX_AUTHENTICATION_AND_AUTHORIZATION_SERVICE:
            pEnviron->put( pEnviron, "Response-Type",
                "Access-Accept" );
            break;
        default:
            pRequest->log( pRequest, REX_LOG_ERROR,
                "Unsupported Service Point" );
            iResult = REX_ERROR;
            break;
    }

    return iResult;
}
```

This service looks for authentication and authorization requests. It is not an accounting service. This service, BlindAA, checks the service point from which it is being called. If the service detects that Cisco Access Registrar is starting or stopping, it does nothing. If it detects that it is being run as an authentication/authorization service, it sets the Response-Type environment variable to Access-Accept, (that is, it accepts the request). If the service detects that it is being run as anything else, (for example, an accounting service), it creates the error message "Unsupported Service Point."



To use BlindAA:

Compile the completed program. (Use the compiler recommended for your version of Cisco Access Registrar. For example, use gcc-2.95.3 for Cisco Access Registrar 3.0.) For the header file and the make file, refer to <Cisco Access Registrar DIR>/examples/rexservice.

In aregcmd, enter:

```
cd /Radius/Services
add BlindAA
cd BlindAA
set Type rex
set Filename <location/filename of shared object file>
set EntryPoint BlindAA
```

Now you can use this service like one of the provided services in Cisco Access Registrar by entering:

```
cd /Radius
set DefaultAuthenticationService BlindAA
set DefaultAuthorizationService BlindAA
```

```
save
reload
```



An Accounting Service Example

The following example shows a custom accounting service named `RexAccountingService`.

```
int REXAPI RexAccountingService( int iServicePoint,
                                rex_AttributeDictionary_t* pRequest,
                                rex_AttributeDictionary_t* pResponse,
                                rex_EnvironmentDictionary_t* pEnviron )
{
    int iResult = REX_OK;

    if( iServicePoint == REX_START_SERVICE )
    {
        // open connection to datastore
        iResult = openDatastoreConnection() ;
    }
    else
    if( iServicePoint == REX_STOP_SERVICE )
    {
        // close connection to datastore
        iResult = closeDatastoreConnection() ;
    }
    else
    if( iServicePoint == REX_ACCOUNTING_SERVICE )
    {
        // send accounting attributes to datastore
        iResult = sendToDatastore( pRequest ) ;
    }
    else
    {
        pRequest->log( pRequest, REX_LOG_ERROR, "Unsupported Service Point" );

        iResult = REX_ERROR;
    }

    return iResult;
}
```

In this example of the accounting service `RexAccountingService`, the service checks the service point from which it is being launched. If it detects that Cisco Access Registrar is starting, it opens a connection to the datastore. If it detects that Cisco Access Registrar is stopping, it closes any open connections. This ensures that any actions that require writing data have a connection to the datastore when needed and connections are closed gracefully when Cisco Access Registrar is shutting down.

If the service detects an accounting request (that is, the service is being launched by the accounting service service point), it writes the accounting attributes to the datastore. If the service detects that it is being run as anything else, it creates the error message “Unsupported Service Point.”



An Accounting Service with Thread Control

The following example uses the same accounting service as in the previous example, but uses the REX_PENDING/reschedule facilities.

```
int REXAPI RexAccountingService( int iServicePoint,
                                rex_AttributeDictionary_t* pRequest,
                                rex_AttributeDictionary_t* pResponse,
                                rex_EnvironmentDictionary_t* pEnviron )
{
    int iResult = REX_OK;

    if( iServicePoint == REX_START_SERVICE )
    {
        // open connection to datastore
        if( iResult = openDatastoreConnection() )
        {
            // start thread that sends accounting attributes to datastore
            iResult = startDatastoreProcessingThread() ;
        }
    }
    else
    if( iServicePoint == REX_STOP_SERVICE )
    {
        // stop datastore thread stopDatastoreProcessingThread() ;
        // close connection to datastore closeDatastoreConnection() ;
    }
    else
    if( iServicePoint == REX_ACCOUNTING_SERVICE )
    {
        // send accounting attributes to datastore processing thread
        // Datastore processing thread will return REX_PENDING immediately once it
        // has the request. Once the request is processed to the datastore, it will
        // call reschedule() for each request it has processed
        iResult = sendToDatastoreProcessingThread( pRequest ) ;
    }
    else
    {
        pRequest->log( pRequest, REX_LOG_ERROR, "Unsupported Service Point" );

        iResult = REX_ERROR;
    }

    return iResult;
}
```

In this extension of the example of the accounting service `RexAccountingService`, the service checks the service point from which it is being launched to determine if the service is being launched by the accounting service service point. If it detects that Cisco Access Registrar is starting, it opens a connection and starts a thread to write data to the datastore. If the service detects that Cisco Access Registrar is stopping, it stops any threads and closes any open connections. This ensures that connections and threads are properly managed.

If the service detects an accounting request, it sends the accounting attributes to the processing thread. Upon receipt of the request, the datastore processing thread returns `REX_PENDING` to let the server know that the service has taken responsibility for the request. As soon as the request is processed to the datastore, the thread calls the `reschedule()` function to let the server know that the service is done with the request and that the server can resume processing the request.

If the service detects that it is being run as anything else, it creates the error message "Unsupported Service Point."

Conclusion

Service providers today compete in a complicated and fast-paced environment in which everything changes except the need to provide the services and technologies that their customers need. AAA is critical in this dynamic and demanding environment, and service providers need a AAA solution that is scalable, powerful, and flexible.

Cisco Access Registrar is designed to meet these needs by providing a feature-rich AAA server. Just as importantly, Cisco Access Registrar also includes special interfaces that allow service providers to automate and centralize AAA configuration, provide control and customization over the processing of each request, and allow for the customization of the AAA function itself.

These special interfaces provide the scalability (by allowing support for frequent and numerous configuration changes), the power (by supporting the ability to manipulate each request), and the flexibility (by allowing support for new technologies) that service providers require to be able to meet AAA demands.



Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

European Headquarters
Cisco Systems Europe
11 Rue Camille Desmoulins
92782 Issy-les-Moulineaux
Cedex 9
France
www-europe.cisco.com
Tel: 33 1 58 04 60 00
Fax: 33 1 58 04 61 00

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-7660
Fax: 408 527-0883

Asia Pacific Headquarters
Cisco Systems, Inc.
Capital Tower
168 Robinson Road
#22-01 to #29-01
Singapore 068912
www.cisco.com
Tel: +65 317 7777
Fax: +65 317 7799

Cisco Systems has more than 200 offices in the following countries and regions. Addresses, phone numbers, and fax numbers are listed on the
Cisco Web site at www.cisco.com/go/offices

Argentina • Australia • Austria • Belgium • Brazil • Bulgaria • Canada • Chile • China PRC • Colombia • Costa Rica • Croatia
Czech Republic • Denmark • Dubai, UAE • Finland • France • Germany • Greece • Hong Kong SAR • Hungary • India • Indonesia • Ireland
Israel • Italy • Japan • Korea • Luxembourg • Malaysia • Mexico • The Netherlands • New Zealand • Norway • Peru • Philippines • Poland
Portugal • Puerto Rico • Romania • Russia • Saudi Arabia • Scotland • Singapore • Slovakia • Slovenia • South Africa • Spain • Sweden
Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • United States • Venezuela • Vietnam • Zimbabwe

All contents are Copyright © 2002 Cisco Systems, Inc. All rights reserved. Cisco, Cisco Systems, Cisco IOS, and the Cisco Systems logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and certain other countries.

All other trademarks mentioned in this document or Web site are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0203R)