

STEEL-BELTED
RADIUS[®]
LDAP Scripting Guide

Release 5.4
March 2006

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089
USA
(408) 745-2000
www.juniper.net

© Copyright 2004–2006 Juniper Networks, Inc. All rights reserved. Printed in the USA.

Steel-Belted Radius, Juniper Networks, and the Juniper Networks logo, are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners. All specifications are subject to change without notice.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

M06223

Contents

About This Guide

Who Should Read This Guide.....	v
Before You Begin	v
What's In This Manual.....	v
Typographical Conventions	vi
Related Documentation.....	vii
Contacting Technical Support.....	ix

Chapter 1

LDAP Authentication and JavaScript

LDAP Request Life Cycle	1
Unscripted LDAP Searches	2
LDAP Scripting.....	3
About JavaScript.....	4
Error Handling.....	4
[Failure] Section.....	4
Last Resort Server	4
Return Codes.....	5
Plugin Return Codes and Script Return Codes	5
Password Processing.....	5

Chapter 2

LDAP Script Configuration

LDAP Authentication (ldapauth.aut) File.....	7
[Settings] Section	7
[ScriptTrace] Section.....	8
[Script] Section.....	9

Chapter 3

Creating LDAP Scripts

LDAP Script Basics.....	11
Working with the Variable Table.....	11
Invoking LDAP Queries.....	12
Writing to the Steel-Belted Radius Log	12

Other Recommendations	12
Choosing the Return Code	13
Plugin Return Codes	13
Script Return Codes	14
Debugging LDAP Scripts	15
Script Tracing	15
Scriptcheck Utility.....	16

Chapter 4 LDAP Scripting Reference

Function Reference.....	19
Ldap Object.....	19
LdapVariables Object.....	20
Logging and Diagnostic Functions	21
JavaScript Return Codes	22

Chapter 5 LDAP Scripting Examples

Example 1: Simple Authentication.....	23
Example 2: Profile Assignment.....	24
Example 3: Received Attribute Normalization	25
Example 4: Conditional Profile Assignment from User Attribute.....	26

Index

About This Guide

The *LDAP Scripting Guide* describes how to use scripts written in the JavaScript programming language to enhance the search capabilities of the Steel-Belted Radius LDAP Authentication module.

Who Should Read This Guide

This guide is intended for experienced system and network specialists working in an Internet access environment. You should be familiar with Lightweight Directory Access Protocol (LDAP) directories and the JavaScript scripting language. You should also be familiar with your network environment and conventions.

Before You Begin

Before you use this manual, you should review the *Steel-Belted Radius Administration Guide* help you understand how the components of Steel-Belted Radius work together. In particular, you should be familiar with LDAP authentication and the use of RADIUS attributes.

This manual assumes that you have installed the Steel-Belted Radius server software and the SBR Administrator. For information on how to install the Steel-Belted Radius software, refer to the *Steel-Belted Radius Getting Started* manual.

What's In This Manual

This manual contains the following chapters:

- ▶ [Chapter 1, “LDAP Authentication and JavaScript,”](#) describes how the LDAP plugin handles authentication requests and describes how JavaScript integrates into that process.
- ▶ [Chapter 2, “LDAP Script Configuration,”](#) describes how to configure the `ldapauth.aut` file to support LDAP scripting.

- ▶ Chapter 3, “Creating LDAP Scripts,” describes how to write LDAP scripts and how to interpret return codes.
- ▶ Chapter 4, “LDAP Scripting Reference,” describes the functions, objects, and methods used in LDAP scripting.
- ▶ Chapter 5, “LDAP Scripting Examples,” provides examples of how LDAP scripting can be used to search and modify the LDAP variable tables in Steel-Belted Radius.

Typographical Conventions

This manual uses the following conventions to present special types of text.

Computer Text

Filenames, directory names, IP addresses, URLs, commands, and file listings appear in a plain fixed-width font:

```
For more information, go to http://www.funk.com...  
[EventDilutions]  
SQLConnectFailure=8
```

In examples, text that you type literally is shown in a bold font.

```
C:\>cd \radius\service
```

Variable Text

Variable text that you must replace with your own information appears in *italics*. For example, you would enter your name and password in place of **YourName** and **YourPassword** in the following interaction.

```
Enter your name: YourName  
Password: YourPassword
```

File names and computer text can also be displayed in italics to indicate that you should replace the values shown with values appropriate for your enterprise. For example, you would enter your own information in place of the italicized text in the following example:

```
[EventDilutions]  
EventName=DilutionCount
```

Key Names

Names of keyboard keys appear in SMALL CAPS. When you need to press two or more keys simultaneously, the key names are joined by a + sign:

Press RETURN.

Press CTRL+ALT+DEL.

Syntax

- ▶ *radiusdir* represents the directory into which Steel-Belted Radius has been installed. By default, this is `C:\Radius\Service` for Windows systems and `/opt/funk/radius` on Linux and Solaris systems.
- ▶ Brackets [] enclose optional items in format and syntax descriptions. In the following example, the first *Attribute* argument is required; you can include an optional second *Attribute* argument by entering a comma and the second argument (but not the square brackets) on the same line.

```
[AttributeName]  
<add | replace> = Attribute [,Attribute]
```

In configuration files, braces identify section headers:

...the [Processing] section of `proxy.ini`...

In screen prompts, braces indicate the default value. For example, if you press **ENTER** without entering anything at the following prompt, the system uses the indicated default value (`/usr/lib`).

- ▶ Angle brackets < > enclose a list from which you must choose an item in format and syntax descriptions.
- ▶ A vertical bar (|) separates items in a list of choices. In the following example, you must specify `add` or `replace` (but not both):

```
[AttributeName]  
<add | replace> = Attribute [,Attribute]
```

Related Documentation

The following documents supplement the information in this manual.

Steel-Belted Radius Documentation

Please review the `releasenotes.txt` file that accompanies your Steel-Belted Radius software for late-breaking information not available in this manual.

In addition to this manual, the Steel-Belted Radius documentation includes the following manuals:

- ▶ The *Steel-Belted Radius Getting Started* manual describes how to install, configure, and administer the Steel-Belted Radius software on a server running the Solaris, Linux, or Windows operating system.
- ▶ The *Steel-Belted Radius Administration Guide* describes how to configure and administer the Steel-Belted Radius server software.
- ▶ The *Steel-Belted Radius Reference Guide* describes the configuration files and settings used by Steel-Belted Radius.

Requests for Comments (RFCs)

The Internet Engineering Task Force (IETF) maintains an online repository of Request for Comments (RFC)s online at <http://www.ietf.org/rfc.html>.

- ▶ RFC 2548, *Microsoft Vendor-specific RADIUS Attributes*. G. Zorn. March 1999.
- ▶ RFC 2618, *RADIUS Authentication Client MIB*. B. Aboba, G. Zorn. June 1999.
- ▶ RFC 2619, *RADIUS Authentication Server MIB*. G. Zorn, B. Aboba. June 1999.
- ▶ RFC 2620, *RADIUS Accounting Client MIB*. B. Aboba, G. Zorn. June 1999.
- ▶ RFC 2621, *RADIUS Accounting Server MIB*. G. Zorn, B. Aboba. June 1999.
- ▶ RFC 2809, *Implementation of L2TP Compulsory Tunneling via RADIUS*. B. Aboba, G. Zorn. April 2000.
- ▶ RFC 2865, *Remote Authentication Dial In User Service (RADIUS)*. C. Rigney, S. Willens, A. Rubens, W. Simpson. June 2000.
- ▶ RFC 2866, *RADIUS Accounting*. C. Rigney. June 2000.
- ▶ RFC 2867, *RADIUS Accounting Modifications for Tunnel Protocol Support*. G. Zorn, B. Aboba, D. Mitton. June 2000.
- ▶ RFC 2868, *RADIUS Attributes for Tunnel Protocol Support*. G. Zorn, D. Leifer, A. Rubens, J. Shriver, M. Holdrege, I. Goyret. June 2000.
- ▶ RFC 2869, *RADIUS Extensions*. C. Rigney, W. Willats, P. Calhoun. June 2000.
- ▶ RFC 2882, *Network Access Servers Requirements: Extended RADIUS Practices*. D. Mitton. July 2000.
- ▶ RFC 3162, *RADIUS and IPv6*. B. Aboba, G. Zorn, D. Mitton. August 2001.

Third-Party Products

For more information about configuring your access servers and firewalls, consult the manufacturer's documentation provided with each device.

Contacting Technical Support

If you're located in the U.S. and Canada, you can contact Funk Software Technical Support in the following ways:

Telephone:	1-617-491-6503
Email:	support@funk.com
Web:	Go to http://www.funk.com , click Tech Support , and then click Steel-Belted Radius .
From within SBR Administrator:	Choose Web > Steel-Belted Radius User Page .

If you're located outside the U.S. and Canada, please contact the authorized Funk Software partner in your area to obtain support.

- ▶ Our Technical Support department is open weekdays between 9:00 AM and 5:30 PM (Eastern) to customers who are on warranty support, who are evaluating the product, or who are not covered by a support contract.
- ▶ Our Technical Support department is open weekdays between 9:00 AM and 8:00 PM (Eastern) to customers who hold a current annual maintenance and support contract.

When you call, please have the following at hand:

- ▶ The Steel-Belted Radius product edition and release number.
- ▶ Information about the server configuration and operating system, including any OS patches that have been applied.
- ▶ For licensed products under a current maintenance agreement, your license or support contract number.
- ▶ Question or description of the problem, with as much detail as possible.
- ▶ Any documentation that may help in resolving the problem, such as error messages, memory dumps, compiler listings, and error logs.

You can use the Funk Software website (<http://www.funk.com>) to register your software, display answers to frequently asked questions, search the Steel-Belted Radius technical support database, and download product documentation in Adobe Reader (.pdf) format.

Chapter 1

LDAP Authentication and JavaScript

This chapter describes how the LDAP plugin handles authentication requests and describes how JavaScript integrates into that process.

LDAP Request Life Cycle

Many companies use Lightweight Directory Access Protocol (LDAP) directory servers to store user authentication and authorization information. Steel-Belted Radius can process authentication requests against records stored in one or more external LDAP databases. Steel-Belted Radius performs the following steps in response to an LDAP authentication request for both scripted and non-scripted configurations.

- 1 At the beginning of each LDAP authentication request, Steel-Belted Radius creates a variable table to map RADIUS access-request attributes to LDAP attributes for use in LDAP Bind, Base, and Search strings. The [Request] section of the LDAP plugin configuration file is used to select which attributes are extracted from the incoming request and placed in the variable table.
- 2 Steel-Belted Radius performs one or more LDAP searches. Parameters for each search are given in the [Search/name] sections of the configuration file. After a search is performed, selected attributes are copied from the LDAP response and placed in the variable table.
- 3 Steel-Belted Radius uses the [Response] section to select information from the variable table to be returned to the RADIUS client in the RADIUS response packet.

[Figure 1](#) illustrates how the LDAP variable table is populated with information coming from a RADIUS access-request message, default values, and the results of LDAP Bind, Base, and Search requests. The information in the variable table is then used to format the access-response packet returned to the RADIUS client.

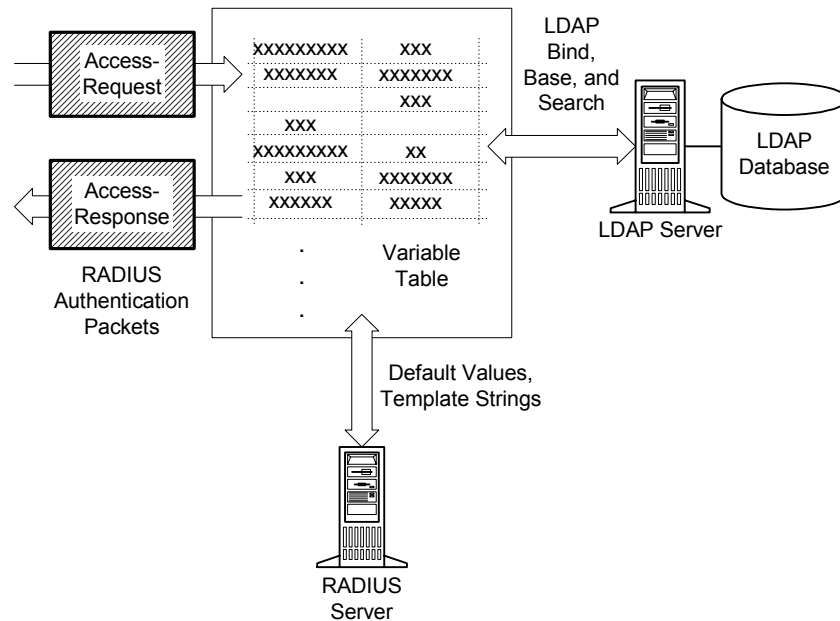


Figure 1 Role of the Variable Table in LDAP Authentication

Unscripted LDAP Searches

Scripting is not required for basic applications of LDAP authentication. In unscripted configurations, search parameters such as base Distinguished Names (DNs), filter strings, and attribute maps are configured in the `ldapauth.aut` file. Using the `OnFound` and `OnNotFound` settings of the `[Search/name]` sections, you can configure a decision tree in which the result of one LDAP query (“Found” or “Not Found”) determines whether another query is executed or the final authentication decision is returned to Steel-Belted Radius. The basic query tree provides sufficient control to meet the needs of many LDAP authentication applications. A sample query tree using unscripted branching is shown in [Figure 2](#).

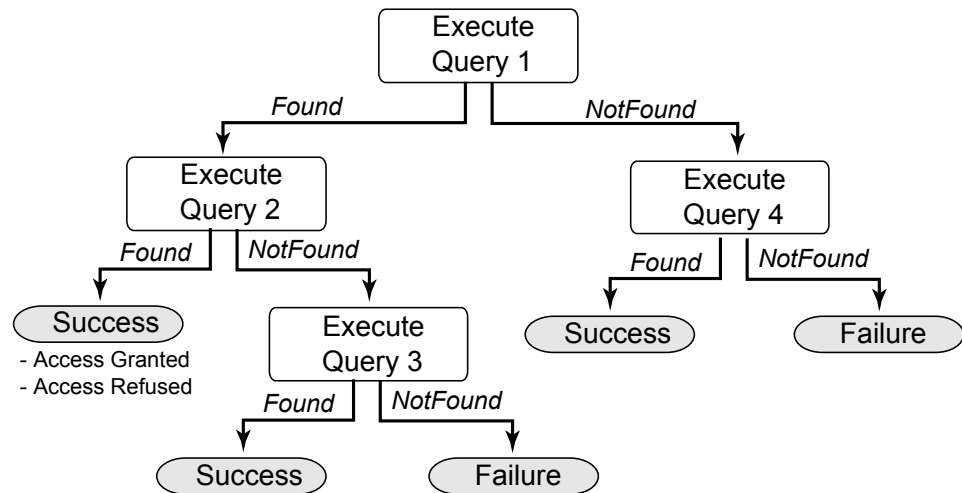


Figure 2 Query Tree with Unscripted Branching

LDAP Scripting

LDAP scripting is used when more sophisticated decision logic or attribute manipulation is required than can be implemented using unscripted searches. Incorporating JavaScript into the Steel-Belted Radius `ldapauth.aut` file gives you much greater flexibility in the processing of LDAP authentication queries. Scripted authentication allows a level of control comparable to SQL stored procedures.

For example, LDAP scripts can combine data from several LDAP queries and analyze the results to determine which query to invoke next. LDAP scripts can evaluate loops and complicated if-then-else logic, build up RADIUS attribute value strings from scratch, and write status messages to the Steel-Belted Radius log.

[Figure 3](#) illustrates the data flow involved in a scripted query. Instead of following a rigid branch structure, the request is processed according to the logic of the LDAP script, which may be arbitrarily complex. The script executes one or more LDAP queries, computes intermediate results from the return values, updates the LDAP variable table, and possibly executes additional queries against the LDAP server. Once the script has completed processing the request and made an authentication decision, it returns a result code to the plugin.

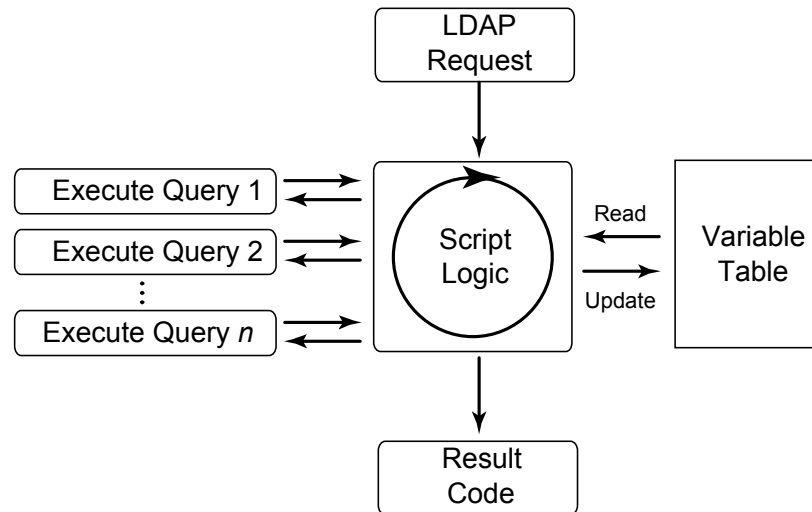


Figure 3 Scripted Query Data Flow

About JavaScript

Scripts for the Steel-Belted Radius LDAP authentication plugin are written in ECMAScript (ECMA-262), which is the international standard corresponding roughly to JavaScript 1.5. JavaScript is a compact, cross-platform, object-based scripting language that combines a powerful syntax with simplicity and ease of use.

NOTE: The JavaScript compiler that runs LDAP scripts is a component of Steel-Belted Radius, and does not depend on the browser or JVM you have installed on your computer.

Error Handling

Processing of an LDAP authentication request may not always be successful. Steel-Belted Radius may be unable to contact any of the LDAP servers listed in the configuration file, or the requested user may not be found. Settings in the `ldapauth.aut` file control how Steel-Belted Radius handles error conditions.

[Failure] Section

The [Failure] section of the `ldapauth.aut` file is used to determine the result when Steel-Belted Radius is unable to connect to any of the configured LDAP servers. Either an accept or reject response may be sent. If configured to accept on failure, the user's full name and a Steel-Belted Radius profile entry may be specified for the return result.

Last Resort Server

If an LDAP query against a server results in a “no record found” response, the request may be retried against the “last resort” server. A typical use for this feature is to query a

master accounts server to check for a user entry that has not yet propagated out to the rest of the LDAP server cluster.

Return Codes

Once the LDAP plugin has finished processing a request, it must return a result to the Steel-Belted Radius core. The plugin may signal that the user should be accepted or rejected, or (if it is unable to process the request) it may ask Steel-Belted Radius to defer the decision to another authentication method. Finally, the plugin may indicate to Steel-Belted Radius that a software failure occurred during processing. It is up to Steel-Belted Radius to interpret the plugin return code and determine what action to take next.

The return code from the plugin to Steel-Belted Radius is a numerical value that it sent via the Steel-Belted Radius authentication plugin application programming interface (API). For unscripted requests, the return code is generated internally by hard-coded logic within the LDAP plugin. When LDAP scripting is used, the plugin return code is generated by the LDAP plugin depending on the value returned by the script when it terminates.

NOTE: *The authentication plugin API is not visible to LDAP script programmers.*

Plugin Return Codes and Script Return Codes

It's important to understand the difference between plugin return codes and script return codes. After the LDAP plugin sends a return code to Steel-Belted Radius, the authentication request is finished and the plugin does not execute again until it receives a new request.

On the other hand, when an LDAP script returns, the LDAP plugin remains in control and may not send a result code to Steel-Belted Radius at that time. For example, the script may return the code indicating a communication failure with the LDAP server occurred. The plugin may retry the script with a different LDAP server and, if the connection succeeds, ultimately return a valid accept or reject code to Steel-Belted Radius.

LDAP script return codes are strings, whereas plugin return codes are integers. For historical reasons, both sets of codes have the same names, and their meanings are similar, but distinct. LDAP script programmers should study the rules for the translation of script return codes to plugin return codes. These are described in detail in [“Choosing the Return Code”](#) on page 13.

Password Processing

For BindName authentication, the %Password attribute is used to return the value of the user password obtained by the LDAP search to the LDAP plugin. The LDAP plugin compares the %Password attribute with the password supplied in the incoming Access-Request and accepts the user if the passwords match. If the passwords do not match, the user is rejected, even if the script returned SBR_RET_SUCCESS.

For more information on password process, see [“Script Return Codes”](#) on page 14.

Chapter 2

LDAP Script Configuration

This chapter describes how to configure the `ldapauth.aut` file to support LDAP scripting.

LDAP Authentication (`ldapauth.aut`) File

The `ldapauth.aut` file tells the Steel-Belted Radius server how to communicate with the external LDAP database, how to query the external database for information, and how to structure the response from a query result. This file is also used to configure and define LDAP scripts. Many of the settings that you use for non-scripted LDAP authentication perform the same functions when LDAP scripting is implemented.

The following sections describe the settings that are new or changed for use with LDAP scripting. This information is a supplement to the "LDAP Authentication Files" chapter in the *Steel-Belted Radius Reference Guide*.

[Settings] Section

Two scripting parameters have been added to the `ldapauth.aut` file.

- ▶ The `MaxScriptSteps` parameter lets you specify the maximum number of statements a script can execute before terminating. You can use the `MaxScriptSteps` parameter to make sure a script does not get caught in an infinite loop.
- ▶ The `ScriptTraceLevel` parameter controls the amount of line-by-line debugging information that is produced automatically or under program control by the script. At the lowest level, no tracing is performed, and at the highest level a trace is written to the log for every JavaScript statement that is executed. See [“Debugging LDAP Scripts” on page 15](#) for more information on script debugging.

Table 1. [Settings] Section in ldapauth.aut

Parameter	Function
MaxScriptSteps	Specifies the number of JavaScript statements that can be executed during a query. If the limit is reached, the script terminates and the LDAP plugin returns an error code to Steel-Belted Radius. Default value is 10000.
ScriptTraceLevel	Controls the generation of line-by-line script trace information in the log. <ul style="list-style-type: none"> At Level 0, no traces are logged. At Level 1 traces are only logged when the SbrTrace() function is executed by the script. At Level 2, a trace is generated for every line executed by the script. Default value is 0.

[ScriptTrace] Section

The [ScriptTrace] section specifies which RADIUS attributes and script variables appear in script trace logs. You can use the [ScriptTrace] section to identify the trace information you want to record in the log file.

Table 2. [ScriptTrace] Section in ldapauth.aut

Parameter	Function
attr	Specifies the name of a RADIUS attribute that will appear in script trace logs.
var	Specifies the name of a JavaScript program variable that will appear in script trace logs.

Each line in the [ScriptTrace] section defines a single attribute or variable. In the following example, the identifiers `User-Name` and `Service-Type` refer to RADIUS attributes in the variable table. The identifiers `count` and `userid` refer to program variables in the JavaScript execution context.

```
[ScriptTrace]
attr = User-Name
attr = Service-Type
var = count
var = userid
```

[Script] Section

The [Script] section contains the body of your LDAP script. Unlike other configuration file sections, where parameters appear on individual lines, the LDAP script is entered as multi-line block of text. The script is processed until a line is encountered that begins with a left bracket ([) or the end of the file is reached.

The following example is a simple script that writes a message to the log and accepts the user regardless of the supplied credentials.

```
[Script]
// Define a function that writes its arguments to the log.
function logArgs(ival, sval) {
    SbrWriteToLog("logArgs: ival=" + ival + ", sval=" + sval);
}

// Declare two variables and pass them to the logging function.
var i = 1;
var s = "Hello";
logArgs(i, s);

// Accept the user unconditionally.
return SBR_RET_SUCCESS;
```


Chapter 3

Creating LDAP Scripts

This chapter describes how to write LDAP scripts and how to interpret return codes.

LDAP Script Basics

To configure LDAP scripting, you add JavaScript instructions to the [Script] section of the `ldapauth.aut` file. You can perform the following operations in your LDAP scripts:

- ▶ Get, set, and reset values of variables stored in the LDAP variable table
- ▶ Invoke LDAP queries defined in the [Search/*name*] sections of the `ldapauth.aut` file
- ▶ Write diagnostic messages and script traces to the Steel-Belted Radius log
- ▶ Evaluate arbitrary program logic coded in your script
- ▶ Exit the script and return a result code string to the LDAP plugin

When Steel-Belted Radius starts, it reads the text of the [Script] section from `ldapauth.aut` and passes it as a block to the JavaScript interpreter, which compiles it into bytecodes. The bytecodes are stored for execution during subsequent LDAP authentication requests. If syntax errors are detected in the JavaScript text, the script does not compile and the LDAP authentication plugin is disabled. Any error messages generated during script compilation appear in the Steel-Belted Radius log file.

If you are using the Global Enterprise Edition or Service Provider Edition of Steel-Belted Radius, you can use the `scriptcheck` utility to check your LDAP scripts for syntax errors without having to start Steel-Belted Radius. See [“Scriptcheck Utility” on page 16](#) for more information about `scriptcheck`.

Working with the Variable Table

You configure the variable table for scripting the same way you do for unscripted configurations. Input RADIUS attributes that the script manipulates must be identified in the [Request] section of the `ldapauth.aut` file. Output RADIUS attributes that the script manipulates must be identified in the [Response] section of the `ldapauth.aut` file.

The `LdapVariables` object is available to your script for manipulating attributes in the variable table. The `LdapVariables` object exposes three methods that scripts can call:

- ▶ `LdapVariables.Get()` retrieves the current value or values for a variable stored in the LDAP variable table.
- ▶ `LdapVariables.Add()` creates a new variable or adds a value to an existing variable.
- ▶ `LdapVariables.Reset()` deletes all of the values of the specified variable.

Invoking LDAP Queries

Any query defined in a `[Search/name]` section of `ldapauth.aut` can be invoked programmatically by an LDAP script. Use the `Ldap.Search()` method to invoke the query, giving the name of the query as the argument to the method.

As with unscripted searches, you can identify a set of LDAP attributes to be extracted from the LDAP response and placed in the variable table. You do this by creating an `[Attributes/name]` section in the `ldapauth.aut` file and specifying this section with the `Attributes` parameter in the query definition.

For more information about LDAP attributes, refer to the "LDAP Authentication Files" chapter of the *Steel-Belted Radius Reference Guide*.

Writing to the Steel-Belted Radius Log

Use the `SbrWriteToLog()` function (described in “[SbrWriteToLog\(\)](#)” on page 21) to insert diagnostic or informational text strings into the Steel-Belted Radius log file. You can use the optional `level` argument to control the log level visibility of your message.

Use the `SbrTrace()` function (described in “[SbrTrace\(\)](#)” on page 21) to display trace information about your script in the Steel-Belted Radius log.

Other Recommendations

- ▶ ECMAScript comments begin with `//` or `/*`. Lines in the `ldapauth.aut` file that begin with a semicolon (`;`) are ignored. For clarity, you should use `//` or `/*` to identify script comments.
- ▶ Do not execute input/output (I/O) operations to the display, to a disk, or to a network from a script. I/O from scripts has not been tested, and may impair performances. To add messages to the radius log file, use the `SbrWriteToLog()` function.
- ▶ Thoroughly test all scripts for speed, and monitor the performance of Steel-Belted Radius after you deploy a new script. In general, the complexity of an LDAP script has a direct impact on server performance.
- ▶ If a `[Script]` section is present in the `ldapauth.aut` file, then the `Search=` line in the `[Settings]` section of the file is ignored. To prevent a script from running, change the `[Script]` heading to `[NoScript]` in the `ldapauth.aut` file.
- ▶ After you modify LDAP scripts, you must restart Steel-Belted Radius. You cannot reload the LDAP plugin with the `radhup` command.

Choosing the Return Code

When a script finishes running, it sends a return code string back to the LDAP plugin. Depending on the value of the return code and the state of the request, the plugin can do one of several things:

- ▶ It can convert the script return code to a plugin return code and send that code directly to Steel-Belted Radius, ending the processing of that request by the plugin.
- ▶ It can perform failure processing, and generate a plugin return code from the [Failure] section in `ldapauth.aut`.
- ▶ It can re-execute the script against a different LDAP server and process the new result when the script is finished.

The script programmer must understand exactly how the LDAP plugin processes script return codes. This information is given in detail in the following sections.

Plugin Return Codes

An LDAP script may execute several times in the course of handling a single authentication request. In the end, the LDAP plugin must make an authentication decision and send a plugin result code back to the Steel-Belted Radius server. The following are the four most common plugin return codes and the resulting actions taken by Steel-Belted Radius.

NOTE: The return codes described below are numerical values sent by the LDAP plugin to the Steel-Belted Radius core via the authentication plugin API. These codes are distinct from the LDAP script return codes that are described in “[Script Return Codes](#)” on page 14.

SBR_RET_SUCCESS

The `SBR_RET_SUCCESS` code indicates to the Steel-Belted Radius core that the authentication request was processed successfully and the user should be accepted.

SBR_RET_DO_NOT_AUTHENTICATE

The `SBR_RET_DO_NOT_AUTHENTICATE` code indicates that the authentication failed and a hard reject should be performed immediately. No other authentication methods should be called.

SBR_RET_NOT_AUTHENTICATED

The `SBR_RET_NOT_AUTHENTICATED` code indicates that the plugin was unable to authenticate the user, but the user should not be rejected yet. The next authentication method in the server configuration should be tried.

SBR_RET_FAILURE

The `SBR_RET_FAILURE` code indicates that an unspecified software failure occurred in the plugin during processing of the request. Steel-Belted Radius should reject the user and write a failure message to the log.

Script Return Codes

Script return codes are strings that are handed from the script to the LDAP plugin when the LDAP script finishes running. You specify the return code as an argument to the JavaScript return statement. When the plugin receives the script return code, it determines whether to send a plugin return code to Steel-Belted Radius or re-execute the script. (See “Return Codes” on page 5). The action taken by the plugin depends on both the value of the script return code and the current LDAP plugin state. The script return codes and their effect on the LDAP plugin are described below.

NOTE: *The return codes described below are strings sent by the script to the LDAP plugin. These codes are distinct from the plugin return codes sent by the plugin to Steel-Belted Radius. For descriptions of the plugin return codes, see “Plugin Return Codes” on page 13*

SBR_RET_SUCCESS

The SBR_RET_SUCCESS code indicates to the LDAP plugin that the script was processed successfully.

- ▶ If the script returns SBR_RET_SUCCESS and a value for the %Password return list attribute is not configured in the [Response] section of the applicable .aut file, the user is accepted.
- ▶ If the script returns SBR_RET_SUCCESS and a value has been set for the %Password return list attribute, the plugin validates the user-supplied password against the value in the return list attribute. If the passwords match, the user is accepted and normal attribute processing continues.

NOTE: *Because the value of the %Password return list attribute is encrypted, you cannot pass it into a script to compare values. You can configure a script variable to pass out a value for %Password.*

SBR_RET_DO_NOT_AUTHENTICATE

The SBR_RET_DO_NOT_AUTHENTICATE code indicates to the LDAP plugin that a hard reject should be performed by the server. The plugin finishes processing the request and sends the plugin return code SBR_RET_DO_NOT_AUTHENTICATE to the Steel-Belted Radius core.

SBR_RET_TRY_NEXT_AUTH_METHOD

The SBR_RET_TRY_NEXT_AUTH_METHOD code indicates that the LDAP plugin should stop processing the request and ask Steel-Belted Radius to try the next authentication method without immediately rejecting the user. The plugin sends the plugin return code SBR_RET_TRY_NEXT_AUTH_METHOD to the Steel-Belted Radius core. Last resort processing is not performed.

SBR_RET_NOT_AUTHENTICATED

The SBR_RET_NOT_AUTHENTICATED code indicates to the LDAP plugin that the script could not authenticate the user. If a last resort server is defined, the LDAP plugin

should re-execute the script against that server. If there is no last resort server, this return code has the same effect as `SBR_RET_NOT_TRY_NEXT_AUTH_METHOD`.

SBR_RET_FAILURE

The `SBR_RET_FAILURE` code indicates to the LDAP plugin that a communication failure with the LDAP server occurred. The plugin should re-execute the script against the next LDAP server in the configuration, if defined. If only one server is defined, or the last server has already been tried, the LDAP plugin should process the [Failure] section to determine the plugin return code. If there is no [Failure] section, this return code has the same effect as `SBR_RET_NOT_TRY_NEXT_AUTH_METHOD`.

Debugging LDAP Scripts

In addition to calling `SbrWriteToLog()` to write diagnostic messages to the Steel-Belted Radius log file, you can take advantage of the diagnostic features of the LDAP authentication plugin when debugging your scripts.

Script Tracing

A script trace is a block of program status information written to the Steel-Belted Radius log file prior to the execution of a JavaScript statement. Information in the script trace includes:

- ▶ The name of the `.aut` file in which the script is defined
- ▶ The line number and text of the script about to be executed
- ▶ The names and values of specified program variables at the time the trace is written
- ▶ The names and values of specific RADIUS attributes (from the variable table) at the time the trace is written

You define the names of program variables and RADIUS attributes to be displayed in script traces by entering them in the [ScriptTrace] section of the `ldapauth.aut` file. See “[ScriptTrace] Section” on page 8 for information on configuring script tracing in the LDAP plugin.

You have two options for enabling tracing of your LDAP scripts.

- ▶ Manual tracing – You can set the `ScriptTraceLevel` parameter in the [Settings] section of `ldapauth.aut` to 1 and call the `SbrTrace()` function from within your script. This causes a single script trace record to appear in the log from the point in your script where the `SbrTrace()` function was called.
- ▶ Automatic tracing – You can set the `ScriptTraceLevel` parameter in the [Settings] section of `ldapauth.aut` to 2 to enable automatic tracing. In this mode, a script trace is performed every time a JavaScript statement is executed in your script.

NOTE: *Because of the large volume of information produced and the resulting performance impact on Steel-Belted Radius, the use of automatic script tracing is not recommended for production environments.*

The following example lists a small script and a portion of the automatic script trace generated from it.

```
[Script]
var a = 1;
var s = "Hello";
return SBR_RET_SUCCESS;

[ScriptTrace]
attr = User-Name
var = a
var = s
. . .

*** Script Trace (c:\radius\service\ldapauth.aut)
(line 1) var a = 1;
User-Name = testuser
a = <not found>
s = <not found>
*** Script Trace (c:\radius\service\ldapauth.aut)
(line 2) var s = "Hello";
User-Name = testuser
a = 1
s = <not found>
*** Script Trace (c:\radius\service\ldapauth.aut)
(line 3) return SBR_RET_SUCCESS;
User-Name = testuser
a = 1
s = Hello
. . .
```

Note that traces are produced just prior to execution of the JavaScript statement referenced in the trace. For instance, the value of variable "a" is not reflected in the trace on line 1, but appears in the trace on line 2, after the assignment statement has executed. If a variable or attribute has not yet been assigned, or if a variable is out of scope at the time of the trace, the value is displayed in the log as <not found>.

Scriptcheck Utility

The `scriptcheck` utility, which is included in the Global Enterprise Edition (GEE) and Service Provider Edition (SPE) of Steel-Belted Radius, lets you check your LDAP scripts for syntax errors.

NOTE: *The `scriptcheck` utility verifies that your script is syntactically correct. The `scriptcheck` utility does not guarantee your script is free of runtime errors or produces correct results. If your script does not appear to be working properly, review the Steel-Belted Radius log for error messages and enable script tracing to diagnose the problem.*

Running the scriptcheck Utility

To run the `scriptcheck` utility, you specify the name of the `.aut` file containing the script you want to verify. For example, the following command validates the script contained in the `myldapauth.aut` file:

```
/opt/funk/radius% scriptcheck myldapauth.aut
```

When the `scriptcheck` utility runs, it loads the `[Script]` section in the specified `.aut` file and uses the JavaScript interpreter to compile the script text. Any error messages produced during script compilation are printed on the console. You can then correct the errors and rerun `scriptcheck` to verify that the script compiles correctly.

NOTE: *Syntax error line numbers reported by the `scriptcheck` utility are counted relative to the first line of the `ldapauth.aut` file. This is different from the error messages produced by Steel-Belted Radius at server start time. When Steel-Belted Radius prints script syntax error messages, the line numbers are counted relative to the first line of the `[Script]` section in the `ldapauth.aut` file.*

Installation Location

By default, the `scriptcheck` utility (which is named `scriptcheck` on Solaris/Linux and `scriptcheck.exe` on Windows) is installed in two locations:

- ▶ The `radiusdir` home directory. By default, this is `C:\Radius\Service` for Windows systems and `/opt/funk/radius` on Linux and Solaris systems.
- ▶ The `Support_Files\scriptcheck\[Windows | Solaris | Linux]` directory on the installation CD-ROM

The “loose” copies of the program are included for the case where Steel-Belted Radius is installed on one platform, but script development is done on a different platform. This saves users the bother of doing a dummy SBR install just to get a platform-specific copy of the `scriptcheck` utility.

You can copy the appropriate `scriptcheck` executable version to any convenient location and run it there, provided you also copy the `radius.lic` file (described in the next section) to the same location.

JavaScript Upgrade License

Before you can run the `scriptcheck` utility, you must have the `radius.lic` file, which contains your JavaScript upgrade license, in the directory where you run the program. If the `radius.lic` file is not present in the working directory, the program prints `scriptcheck: can't open license file (radius.lic)`. If the file is present but the license isn't correct, the program prints `scriptcheck: not licensed for JavaScript`.

- ▶ If you install the JavaScript upgrade license using SBR Administrator, the `radius.lic` file is updated automatically in the `radiusdir` home directory. After that, you can run `scriptcheck` in that directory with no additional configuration. To run the `scriptcheck` utility in another location, place a copy of your `radius.lic` file to the same directory.
- ▶ Alternatively, you can use a text editor to create a `radius.lic` file and enter the JavaScript upgrade license string into the file by hand.

Chapter 4

LDAP Scripting Reference

This chapter describes the functions, objects, and methods used in LDAP scripting.

Function Reference

Ldap Object

The `Ldap` object exposes methods for invoking LDAP queries from scripts.

Ldap.Search()

Purpose

The `Ldap.Search()` method is used to invoke an LDAP query defined in a `[Search/name]` section of the `ldapauth.aut` file.

Syntax

`Ldap.Search(SearchSection)`

Parameters

<i>SearchSection</i>	Specifies the name of a <code>[Search/<i>name</i>]</code> section of the <code>ldapauth.aut</code> file.
----------------------	--

Returns

<code>Ldap.FOUND</code>	The search found a matching LDAP entry.
<code>Ldap.NOTFOUND</code>	The search did not find a matching LDAP entry.
<code>Ldap.FAILURE</code>	The search encountered an unexpected failure.
<code>Ldap.NOSUCHSEARCH</code>	The specified section was not found in <code>ldapauth.aut</code> .

Example

Given the following query definition,

```
[Search/vpn]
Base = ou=vpn dc=jcn dc=com
Scope = 2
Filter = uid=<Vpn-User-Name>
Attributes = VpnAttrList
```

```
Timeout = 20
%DN = dn
```

you would use the following JavaScript command to invoke the query:

```
Ldap.Search("vpn");
```

LdapVariables Object

The `LdapVariables` object exposes methods for manipulating attributes in the LDAP plugin variable table.

LdapVariables.Get()

Purpose

The `LdapVariables.Get()` method retrieves the current value or values for a variable stored in the LDAP variable table. `LdapVariables.Get()` can be used to retrieve binary (raw) data or text strings.

Syntax

```
LdapVariables.Get(variableName[, nItem])
```

Parameters

<i>VariableName</i>	Specifies the name of the variable in the variable table.
<i>nItem</i>	Specifies the index of the value for multi-valued attributes. You can specify the value of <i>nItem</i> as part of the command, or you can use a separate variable to set the value of <i>nItem</i> (as shown in the following example).

Returns

The value of the specified attribute, or a null value if the attribute doesn't exist or if the index is out of bounds.

Examples

The following example illustrates a single-value attribute lookup:

```
var name = LdapVariables.Get("User-Name");
```

The following example illustrates a multi-value attribute lookup:

```
var index = 2;
var alias = LdapVariables.Get("User-Alias", index);
```

LdapVariables.Add()

Purpose

The `LdapVariables.Add()` method creates a new variable or adds a value to an existing variable.

Syntax

```
LdapVariables.Add(variableName, value[, raw])
```

Parameters

<i>variableName</i>	Specifies the name of the variable to be created or updated.
<i>value</i>	Specifies the value of the variable, which may be text or binary data.
<i>raw</i>	<ul style="list-style-type: none"> If set to <code>True</code>, specifies that the value is binary data, If set to <code>False</code>, specifies that the value is text. Default value is <code>False</code> .

Returns Nothing.

Example `LdapVariables.Add("Vpn-User-Name", "Fred");`

LdapVariables.Reset()

Purpose The `LdapVariables.Reset()` method deletes all of the values of the specified variable from the variable table. If the specified variable does not exist, the method call is ignored.

Syntax `LdapVariables.Reset(variableName)`

Parameters

<i>VariableName</i>	Specifies the name of the variable to be removed from the table.
---------------------	--

Returns Nothing.

Example `LdapVariables.Reset("Vpn-User-Name");`

Logging and Diagnostic Functions

You can use the `SbrWriteToLog()` and `SbrTrace()` functions to insert messages and trace information in the Steel-Belted Radius log file.

SbrWriteToLog()

Purpose The `SbrWriteToLog()` function writes text strings to the Steel-Belted Radius log file.

Syntax `SbrWriteToLog([logLevel,]msg)`

Parameters

<i>logLevel</i>	Specifies log level of the message. The log level configured in Steel-Belted Radius must be greater than or equal to this value for the message to appear in the log.
<i>msg</i>	Specifies the message text to be logged.

Returns Nothing.

Example `SbrWriteToLog(2, "Print this message in the log.");`

SbrTrace()

Purpose The `SbrTrace()` function writes a script trace to the log from the point in the script where the statement appears.

Syntax `SbrTrace([logLevel])`

Parameters

<i>logLevel</i>	Specifies the message log level. The log level must be greater than or equal to this value for the script trace to appear in the log.
-----------------	---

Returns Nothing.

Example `SbrTrace(1);`

JavaScript Return Codes

Table 3 summarizes the behavior of script return codes. The first column (“Script Return Code”) shows the return code string generated by the script. The second column (“Action”) shows the action taken by Steel-Belted Radius. The third column (“Plugin Return Code”) shows the plugin return code sent from the LDAP plugin to the Steel-Belted Radius core.

Table 3. Javascript Return Codes

Script Return Code	Action	Plugin Return Code
SBR_RET_SUCCESS	Accept the user, subject to password processing.	SBR_RET_SUCCESS if the %Password response attribute matches the password supplied in the Access-Request or if the %Password response attribute is not set. SBR_RET_DO_NOT_AUTHENTICATE if the %Password response attribute is set but does not match the password in the Access-Request.
SBR_RET_DO_NOT_AUTHENTICATE	Hard reject. Do not invoke another authentication method.	SBR_RET_DO_NOT_AUTHENTICATE
SBR_RET_TRY_NEXT_AUTH_METHOD	Return from the LDAP plugin and invoke the next authentication method. Do not process [Failure] section or try the last resort server.	SBR_RET_NOT_AUTHENTICATED
SBR_RET_FAILURE	A communication error occurred. Retry the script with the next server in the list, or go to [Failure] section if no server is available.	If another server is available, plugin return code depends on script return code when script is re-executed. If no server is available, process [Failure] section and return SBR_RET_SUCCESS or SBR_RET_NOT_AUTHENTICATED, depending on configuration
SBR_RET_NOT_AUTHENTICATED	Retry script with the last resort server, if defined. Otherwise, go to the next authentication method.	If LastResort is defined, the plugin return code depends on the script return code when the script is re-executed. If LastResort is not defined, returns SBR_RET_NOT_AUTHENTICATED.

Chapter 5

LDAP Scripting Examples

This chapter provides examples of how LDAP scripting can be used to search and modify the LDAP variable tables in Steel-Belted Radius.

Example 1: Simple Authentication

The following script executes the search criteria specified in the [Search/LdapSearch1] section of the ldapauth.aut file. If the search is unsuccessful, the script prepends myco. to the user name and executes the search criteria specified in the [Search/LdapSearch2] section.

```
[Script]
// Try the initial query.
Status = Ldap.Search('LdapSearch1');
if (status == Ldap.NOTFOUND) {
// Add "myco." to the user name and run new query.
userName = LdapVariables.Get('User-Name');
LdapVariables.Reset('User-Name');
LdapVariables.Add('User-Name', 'myco.' + userName);
status = Ldap.Search('LdapSearch2');
}

// Return value depends on final search status.
switch (status) {
case Ldap.FOUND:
    return SBR_RET_SUCCESS;
case LDAP.NOTFOUND:
    return SBR_RET_NOT_AUTHENTICATED;
default:
    return SBR_RET_FAILURE;
}
```

Example 2: Profile Assignment

Scripts can use authentication information to determine the profile that should be assigned to a user. In the following example, the script executes the query specified in the [Search/Radius] section. This query looks up an object named `ProfileData` containing a multiple instances of the `radiusattrs` attribute. The script iterates through the returned values of `radiusattrs`, looking for the first instance that begins with the prefix `sbr-`. If a matching attribute is found, the prefix is stripped from the attribute and returned as the name of the user profile.

The following is the LDIF representation of the `ProfileData` object, showing the values of the `radiusattrs` attributes:

```
dn: name=ProfileData, ou=radius, dc=funk,dc=com
name: ProfileData
objectClass: top
objectClass: radiusobject
radiusattrs: attr1
radiusattrs: attr2
radiusattrs: sbr-defaultprofile
radiusattrs: attr3
```

The relevant sections of the `ldapauth.aut` file are shown below.

```
[Attributes/RadiusAttrs]
radiusattrs

[Response]
%Profile = Return-Profile

[Search/Radius]
Base = ou=radius,dc=funk,dc=com
Scope = 2
Filter = name=ProfileData
Attributes = RadiusAttrs
Timeout = 20
%DN = dn

[Script]
// Look up "ProfileData" object using the "Radius" query.
if (Ldap.Search("Radius") == Ldap.FOUND) {
    var attr;
    var profile = "default";

    // Loop through all "radiusattrs" attributes.
    for(i = 0; attr != "null"; i++) {
        attr = LdapVariables.Get("radiusattrs", i);
        // If prefix matches "sbr-" extract profile name.
        if (attr.substr(0, 4) == "sbr-") {
            profile = attr.substr(4);
            break;
        }
    }
}
```

```

        // Add profile name to the variable table and return.
        LdapVariables.Add("Return-Profile", profile);
        return SBR_RET_SUCCESS;
    }

    // Object wasn't found, so signal a failure.
    return SBR_RET_FAILURE;
}

```

Example 3: Received Attribute Normalization

A common requirement is to normalize incoming RADIUS attributes to a common format before performing an LDAP search. The following example checks the length of the telephone number string in the Calling-Station-ID attribute, preserving only the final seven digits, if necessary. The truncated telephone number is saved as a new entry (Stripped-CSID) in the variable table. The value of Stripped-CSID is specified as part of the Filter parameter in the [Search/Query1] query definition. This query is executed by the script, and the resulting status code determines the script return code.

```

[Request]
%UserName = User-Name
Calling-Station-Id = Received-CSID

[Search/Query1]
Base=ou=people,dc=funk,dc=com
Scope = 2
Filter =
    (&(uid=<User-Name>) (callingStationId=<Stripped-CSID>))
Timeout = 20
%DN = dn

[Script]
// Get the received Calling-Station-ID attribute.
var csid = LdapVariables.Get("Received-CSID");

// Check length and retain last seven digits of CSID.
var length = csid.length;
if (length > 7) {
    csid = csid.substr(length - 7);
    SbrWriteToLog("Shortened CSID to: " + csid);
}

// Save result to variable table so we can search on it.
LdapVariables.Add("Stripped-CSID", csid);

// Perform the search with normalized CSID.
var status = Ldap.Search("Query1");

// Generate return code based on search result.
if (status == Ldap.FOUND) {
    return SBR_RET_SUCCESS;
}
return SBR_RET_NOT_AUTHENTICATED;

```

Example 4: Conditional Profile Assignment from User Attribute

The following example illustrates how you can use LDAP scripts to implement multiple queries and complex decision logic. The script starts by invoking the FindUser query to look up the specified user in the LDAP repository. Depending on the employeeType attribute returned from the first query, a second query is selected and invoked to retrieve attributes specific to the user's employee type. Finally, the Radius-Profile attribute of the employee type record is returned as the profile name for the authentication response.

The LDIF data for a sample user is as follows:

```
dn: uid=SStudent, ou=People, dc=funk,dc=com
employeeType: Student
uid: SStudent
userPassword::
    e1NTSEF9cTZvdFFOYXArcFowaG5rOWJQU3dZYlExbkFILLdoMXBnMlR4
==
givenName: Sam
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
sn: Student
cn: Sam Student
```

The following LDIF entries present the data objects holding the "Radius-Profile" attributes associated with each employee type:

```
dn: ou=radius, dc=funk,dc=com
ou: radius
objectClass: top
objectClass: organizationalunit

dn: name=VendorType, ou=radius, dc=funk,dc=com
Radius-Profile: Vendor-Profile
name: VendorType
objectClass: top
objectClass: radius

dn: name=FacultyType, ou=radius, dc=funk,dc=com
Radius-Profile: Faculty-Profile
name: FacultyType
objectClass: top
objectClass: radius

dn: name=StudentType, ou=radius, dc=funk,dc=com
Radius-Profile: Student-Profile
name: StudentType
objectClass: top
objectClass: radius
```

Finally, here are the configuration settings and the LDAP search script:

```
[Request]
%UserName = User-Name

[Response]
%Profile = Radius-Profile
>Password = userpassword

[Attributes/UserAttributes]
employeeeetype
userpassword

[Attributes/TypeAttributes]
radius-profile

[Search/FindUser]
Base=ou=people,dc=funk,dc=com
Scope = 2
Filter = uid=<User-Name>
Attributes = UserAttributes
Timeout = 20
%DN = dn

[Search/Student]
Base=ou=radius,dc=funk,dc=com
Scope = 2
Filter = name=StudentType
Attributes = TypeAttributes
Timeout = 20

[Search/Faculty]
Base=ou=radius,dc=funk,dc=com
Scope = 2
Filter = name=FacultyType
Attributes = TypeAttributes
Timeout = 20

[Search/Vendor]
Base=ou=radius,dc=funk,dc=com
Scope = 2
Filter = name=VendorType
Attributes = TypeAttributes
Timeout = 20

[Script]
// Look up the specified user in the LDAP repository.
var status = Ldap.Search("FindUser");
if (status != Ldap.FOUND) {
    return SBR_RET_NOT_AUTHENTICATED;
}

// Get the employeeeetype attribute from the query result.
var type = LdapVariables.Get("employeeeetype");
```

Example 4: Conditional Profile Assignment from User Attribute

```
// Execute query to look up employee type object.
switch (type) {
case "Student":
    status = Ldap.Search("Student");
    break;
case "Faculty":
    status = Ldap.Search("Faculty");
    break;
case "Vendor":
    status = Ldap.Search("Vendor");
    break;
default:
    SbrWriteToLog("Invalid employee type: " + type);
    return SBR_RET_DO_NOT_AUTHENTICATE;
}

// This error should never happen.
if (status != Ldap.FOUND) {
    SbrWriteToLog("No record for employee type: " + type);
    return SBR_RET_DO_NOT_AUTHENTICATE;
}

// Get the profile name for this employee type.
var profile = LdapVariables.Get("profilename");
if (profile == "null") {
    profile = "Default-Profile";
}

// Save profile name to variable table and return.
LdapVariables.Add("Radius-Profile", profile);
return SBR_RET_SUCCESS;
```

A

- access-request attributes 1
- angle brackets, meaning of vii
- attr 8
- attribute maps 2
- attribute normalization 25
- Attributes/name section 12

B

- Base strings 1
- Bind strings 1
- brackets, meaning of vii
- bytecodes 11

C

- Calling-Station-ID attribute 25
- comments 12
- contacting technical support ix
- conventions vi

D

- Distinguished Names (DNs) 2

E

- ECMAScript 4
 - comments 12
- employeetype attribute 26

F

- Failure section 4
- filter strings 2

I

- input/output (I/O) operations 12
- Internet Engineering Task Force (IETF) viii

- interpreter 11

J

- JavaScript 4
 - compiler 4
 - interpreter 11
- JavaScript upgrade license 17

L

- last resort server 4, 22
- LDAP 1
- LDAP attributes 1
- Ldap object 19
- Ldap.FAILURE 19
- Ldap.FOUND 19
- Ldap.NOSUCHSEARCH 19
- Ldap.NOTFOUND 19
- Ldap.Search() method 12, 19
- ldapauth.aut file 3, 4, 7, 11, 12, 24
- ldapauth.aut file. 11
- LdapVariables object 12, 20
- LdapVariables.Add() method 12, 20
- LdapVariables.Get() method 12, 20
- LdapVariables.Reset() method 12, 21
- license 17
- Lightweight Directory Access Protocol, see LDAP
- log file 11, 12
- logLevel 21

M

- MaxScriptSteps 7, 8
- methods 19

N

- nItem 20
- normalization 25

O

OnFound 2
OnNotFound 2

P

plugin return codes 5, 13
profile 4, 24

Q

query tree 2

R

radhup command 12
radius.lic file 17
radiusattrs attribute 24
radiusdir vii, 17
Radius-Profile attribute 26
Request section 1, 11
Requests for Comments viii
Response section 1, 11
return code 13
return codes 5, 22
RFCs viii

S

SBR_RET_DO_NOT_AUTHENTICATE 13, 14, 22
SBR_RET_FAILURE 13, 15, 22
SBR_RET_NOT_AUTHENTICATED 13, 14, 22
SBR_RET_SUCCESS 13, 14, 22
SBR_RET_TRY_NEXT_AUTH_METHOD 22
SbrTrace() 21
SbrTrace() function 12
SbrWriteToLog() 15, 21
SbrWriteToLog() function 12
script return codes 5, 14, 22
Script section 9, 11
script tracing 15
scriptcheck utility 11, 16
scriptcheck.exe 17
scripted queries 3
ScriptTrace section 8
ScriptTraceLevel 7, 8

Search strings 1
Search/name section 12
Search/name sections 1
syntax errors 11

T

technical support, contacting ix
tracing 15

U

unscripted searches 2
upgrade license 17

V

var 8
variable table 1, 11
VariableName 20
vertical bar, meaning of vii