Radiator® AAA Server

Installation and reference manual for Radiator® 4.30. Last revised on September 26, 2025

Copyright © 1998-2025 Radiator Software Oy.

Table of Contents

1. Introduction to Radiator® AAA Server 1
2. Installing and upgrading Radiator 3
    2.1. System requirements 3
        2.1.1. Perl 3
        2.1.2. CPAN 3
        2.1.3. Digest::MD5 4
        2.1.4. Digest::SHA 4
        2.1.5. MD4 digest for MSCHAP and MSCHAPv2 5
        2.1.6. Net::SSLeay and OpenSSL 5
        2.1.7. SQL 5
        2.1.8. Memory requirements 5
        2.1.9. Radiator Radius::UtilXS 6
        2.1.10. SCTP 6
    2.2. Updating from demo or locked version to full version 6
    2.3. Installing and upgrading on Linux: Red Hat Enterprise Linux, CentOS, Oracle Linux, AlmaLinux and Rocky Linux 7
    2.4. Installing and upgrading on Linux: SUSE Linux Enterprise Server and openSUSE Leap 8
    2.5. Installing and upgrading on Linux: Ubuntu and Debian 9
    2.6. Installing and upgrading on Linux: generic RPM package 10
    2.7. Installing and upgrading from repository 11
    2.8. Installing and upgrading on Windows 11
        2.8.1. Tips for Radiator on Windows 12
        2.8.2. Installing and upgrading on Windows with MSI package 12
        2.8.3. Installing and upgrading on Windows using Strawberry Perl 13
        2.8.4. Troubleshooting Windows service 15
    2.9. Installing and upgrading with full source distribution 15
    2.10. Installing and upgrading on Solaris 16
    2.11. Installing and upgrading on macOS 16
    2.12. Installing, upgrading and managing with Ansible 17
    2.13. Docker containers for Radiator 17
    2.14. Troubleshooting 17
3. Configuring Radiator 17
    3.1. Tips for using Radiator configuration files 20
    3.2. Including parts of configuration from external files 21
    3.3. Special formatters 21
    3.4. Date formatting 26
    3.5. Address binding 27
    3.6. IPv6 support 28
        3.6.1. IPv6 Clients 28
        3.6.2. IPv6 wildcard listen address 29
        3.6.3. IPv4-mapped IPv6 address 29

3.7. Global parameters 29
3.7.1. Foreground 29
3.7.2. LogStdout 30
3.7.3. Trace 30
3.7.4. LogTraceId 30
3.7.5. LogRejectLevel 31
3.7.6. AuthPort 31
3.7.7. AcctPort 32
3.7.8. KeepSocketsOnReload 33
3.7.9. BindAddress 33
3.7.10. BindV6Only 33
3.7.11. DbDir 34
3.7.12. LogDir 34
3.7.13. LogFile 34
3.7.14. LogMicroseconds 35
3.7.15. PidFile 35
3.7.16. DictionaryFile 35
3.7.17. ProxyUnknownAttributes 35
3.7.18. DiameterDictionaryFile 35
3.7.19. DictionaryReloadInterval 36
3.7.20. Syslog 36
3.7.21. LicenseFile 36
3.7.22. SnmpgetProg 36
3.7.23. SnmpwalkProg 36
3.7.24. FingerProg 37
3.7.25. PmwhoProg 37
3.7.26. LivingstonMIB 37
3.7.27. LivingstonOffs 37
3.7.28. LivingstonHole 37
3.7.29. RewriteUsername 37
3.7.30. SocketQueueLength 38
3.7.31. DefineFormattedGlobalVar 38
3.7.32. DefineGlobalVar 38
3.7.33. StartupHook 38
3.7.34. ShutdownHook 39
3.7.35. DelayedShutdownTime 39
3.7.36. DelayedShutdownHook 39
3.7.37. PreClientHook 39
3.7.38. ClientHook 40
3.7.39. HandlerFindHook 40
3.7.40. USR1Hook 40
3.7.41. USR2Hook 40

3.7.42. WINCHHook 41
3.7.43. MainLoopHook 41
3.7.44. UsernameCharset 41
3.7.45. User 41
3.7.46. Group 41
3.7.47. MaxChildren 41
3.7.48. SnmpNASErrorTimeout 41
3.7.49. ForkClosesFDs 42
3.7.50. ResponseTimeThreshold 42
3.7.51. GlobalMessageLog 42
3.7.52. FarmSize 42
3.7.53. LogFarmInstance 43
3.7.54. FarmChildHook 43
3.7.55. DupCache 43
3.7.56. DupCacheFile 44
3.7.57. EAP_UseState 44
3.7.58. Identifier 44
3.7.59. DisableMTUDiscovery 45
3.7.60. PacketDumpOmitAttributes 45
3.7.61. StatusServer 45
3.7.62. DisabledRuntimeChecks 45
3.7.63. PBKDF2_MinRounds 46
3.7.64. PBKDF2_MaxRounds 46
3.8. SQL configuration 46
3.8.1. SQL bind variables 47
3.8.2. DBSource 48
3.8.3. DBUsername 49
3.8.4. DBAuth 49
3.8.5. Timeout 49
3.8.6. FailureBackoffTime 50
3.8.7. SQLRetries 50
3.8.8. RoundRobinOnFailure 50
3.8.9. ConnectTimeout 50
3.8.10. SQLRecoveryFile 51
3.8.11. ConnectionHook 51
3.8.12. ConnectionAttemptFailedHook 51
3.8.13. NoConnectionsHook 52
3.8.14. ConnectSQLAtStartup 52
3.8.15. AsynchronousSQL 52
3.8.16. AsynchronousSQLConnections 52
3.8.17. RoundRobinQueries 53
3.9. LDAP configuration 53

3.9.1. BaseDN 53
3.9.2. SearchFilter 54
3.9.3. Scope 54
3.9.4. AuthDN 54
3.9.5. AuthPassword 54
3.9.6. Host 55
3.9.7. Port 55
3.9.8. ResolveHost 55
3.9.9. UseSSL 56
3.9.10. UseTLS 56
3.9.11. Debug 57
3.9.12. DebugTLS 57
3.9.13. Timeout 58
3.9.14. FailureBackoffTime 58
3.9.15. BindFailedHook 58
3.9.16. HoldServerConnection 58
3.9.17. NoBindBeforeOp 59
3.9.18. SSLVerify 59
3.9.19. SSLCiphers 60
3.9.20. SSLCAPath 60
3.9.21. SSLCAFile 60
3.9.22. SSLCAClientCert 60
3.9.23. SSLCAClientKey 60
3.9.24. SSLCAClientKeyPassword 61
3.9.25. SSLExpectedServerName 61
3.9.26. Version 61
3.9.27. Deref 61
3.9.28. UseSASL 62
3.9.29. SASLUser 62
3.9.30. SASLPassword 62
3.9.31. SASLMechanism 62
3.9.32. BindAddress 63
3.9.33. MultiHomed 63
3.10. EAP configuration 63
3.10.1. EAPType 63
3.10.2. EAPContextTimeout 64
3.10.3. EAPAnonymous 64
3.10.4. EAPTLS_CAFile 65
3.10.5. EAPTLS_CAPath 65
3.10.6. EAPTLS_CertificateFile 65
3.10.7. EAPTLS_CertificateChainFile 65
3.10.8. EAPTLS_CertificateType 66

3.10.9. EAPTLS_PrivateKeyFile 66
3.10.10. EAPTLS_PrivateKeyPassword 66
3.10.11. EAPTLS_Protocols 66
3.10.12. EAPTLS_Ciphers 67
3.10.13. EAPTLS_SecurityLevel 67
3.10.14. EAPTLS_RandomFile 67
3.10.15. EAPTLS_DHFile 67
3.10.16. EAPTLS_ECDH_Curve 67
3.10.17. EAPTLS_AllowUnsafeLegacyRenegotiation 67
3.10.18. EAPTLS_MaxFragmentSize 68
3.10.19. EAPTLS_CRLCheck 68
3.10.20. EAPTLS_CRLCheckUseDeltas 68
3.10.21. EAPTLS_CRLCheckAll 69
3.10.22. EAPTLS_CRLFile 69
3.10.23. EAPTLS_SessionResumption 69
3.10.24. EAPTLS_SessionResumptionLimit 69
3.10.25. EAPTLS_SessionContextId 70
3.10.26. EAPTLSRewriteCertificateCommonName 70
3.10.27. EAPTLS_PEAPVersion 70
3.10.28. EAPTLS_PEAPBrokenV1Label 70
3.10.29. EAP_PEAP_MSCHAP_Convert 70
3.10.30. EAPTLS_RequireClientCert 70
3.10.31. EAPTLS_PolicyOID 70
3.10.32. EAP_LEAP_MSCHAP_Convert 71
3.10.33. EAP_GTC_MaxLength 71
3.10.34. EAP_GTC_PAP_Convert 71
3.10.35. EAP_MSCHAPv2_UseMultipleAuthBys 71
3.10.36. EAPTLS_NoCheckId 72
3.10.37. EAPTLS_CertificateVerifyHook 72
3.10.38. EAPTLS_CertificateVerifyFailedHook 72
3.10.39. EAPTLS_CommonNameHook 73
3.10.40. EAPTLS_TraceState 74
3.10.41. EAPTLS_CopyToInnerRequest 74
3.10.42. EAPTLS_CAPartialChain 74
3.10.43. EAPTLS_UseCADefaultLocations 74
3.10.44. EAPTLS_NoClientCert 75
3.10.45. EAPTLS_OCSPCheck 75
3.10.46. EAPTLS_OCSPAsyncCheck 75
3.10.47. EAPTLS_OCSPStapling 75
3.10.48. EAPTLS_OCSPURI 75
3.10.49. EAPTLS_OCSPFailureBackoffTime 75
3.10.50. EAPTLS_OCSPStrict 75

3.10.51. EAPTLS_OCSPCacheTime 76
3.10.52. EAPTLS_OCSPCacheSize 76
3.10.53. EAPFAST_PAC_Lifetime 76
3.10.54. EAPFAST_PAC_Reprovision 76
3.10.55. EAP_TTLS_AllowInRequest 76
3.10.56. EAP_TTLS_AllowInReply 76
3.10.57. EAP_Identity_MaxLength 77
3.10.58. EAP_PWD_PrepMethod 77
3.10.59. UsernameCharset 77
3.10.60. NoEAP 78
3.10.61. PreHandlerHook 78
3.10.62. EAPTLS_KeylogFilename 79
3.11. TLS configuration 80
3.11.1. TLS_Protocols 80
3.11.2. TLS_CAFile 81
3.11.3. TLS_CAPath 81
3.11.4. TLS_CertificateFile 82
3.11.5. TLS_CertificateChainFile 82
3.11.6. TLS_CertificateType 82
3.11.7. TLS_PrivateKeyFile 82
3.11.8. TLS_PrivateKeyPassword 83
3.11.9. TLS_Ciphers 83
3.11.10. TLS_Ciphersuites 83
3.11.11. TLS_SecurityLevel 83
3.11.12. TLS_RequireClientCert 84
3.11.13. TLS_Verify 84
3.11.14. TLS_RandomFile 84
3.11.15. TLS_DHFile 84
3.11.16. TLS_ECDH_Curve 84
3.11.17. TLS_CRLCheck 85
3.11.18. TLS_CRLCheckUseDeltas 85
3.11.19. TLS_CRLCheckAll 85
3.11.20. TLS_CRLFile 85
3.11.21. TLS_PolicyOID 86
3.11.22. TLS_ExpectedPeerName 86
3.11.23. TLS_SubjectAltNameURI 86
3.11.24. TLS_SNIHostname 87
3.11.25. TLS_SubjectAltNameDNS 87
3.11.26. TLS_CertificateFingerprint 87
3.11.27. TLS_CertificateVerifyHook 88
3.11.28. TLS_CertificateVerifyFailedHook 88
3.11.29. TLS_SRVName 89

3.11.30. TLS_UseCADefaultLocations 90
3.11.31. TLS_CAPartialChain 90
3.11.32. TLS_NoClientCert 90
3.11.33. TLS_OCSPCheck 90
3.11.34. TLS_OCSPStapling 90
3.11.35. TLS_OCSPURI 90
3.11.36. TLS_OCSPFailureBackoffTime 90
3.11.37. TLS_OCSPStrict 91
3.11.38. TLS_OCSPCacheTime 91
3.11.39. TLS_OCSPCacheSize 91
3.11.40. TLS_KeylogFilename 91
3.11.41. TLS_TraceState 91
3.11.42. UseTLS 92
3.11.43. UseSSL 92
3.11.44. TLS_SessionResumption 92
3.11.45. TLS_SessionResumptionLimit 92
3.12. HTTP client configuration 93
3.12.1. URL 93
3.12.2. FormatURL 93
3.12.3. RequestMethod 93
3.12.4. RequestContentType 93
3.12.5. RequestHeader 94
3.12.6. HTTP_AuthenticationScheme 94
3.12.7. HTTP_AuthenticationHook 94
3.12.8. HTTP_Username 94
3.12.9. HTTP_Password 95
3.12.10. HTTP_Version 95
3.12.11. NoreplyTimeout 95
3.12.12. FailureBackoffTime 95
3.12.13. MaxFailedRequests 95
3.12.14. MaxFailedGraceTime 95
3.12.15. Connections 95
3.12.16. MaxConnections 96
3.12.17. ReconnectTimeout 96
3.12.18. ConnectOnDemand 96
3.12.19. MapResponseHook 96
3.12.20. LocalAddress and LocalPort 96
3.12.21. MaxBufferSize 97
3.12.22. Debug 97
3.13. Gossip configuration 97
3.13.1. Identifier 97
3.13.2. Debug 97

3.13.3. Secret 97
3.13.4. EncryptedSecret 98
3.14. <Client xxxxxx> 98
3.14.1. Secret 100
3.14.2. EncryptedSecret 100
3.14.3. DynAuthSecret 101
3.14.4. EncryptedDynAuthSecret 101
3.14.5. TACACSPLUSKey 101
3.14.6. EncryptedTACACSPLUSKey 101
3.14.7. DefaultRealm 101
3.14.8. DupInterval 102
3.14.9. RewriteUsername 102
3.14.10. IdenticalClients 102
3.14.11. PreHandlerHook 103
3.14.12. StatusServer 104
3.14.13. StatusServerShowClientDetails 104
3.14.14. Identifier 105
3.14.15. PacketTrace 105
3.14.16. DefaultReply 105
3.14.17. StripFromReply 106
3.14.18. AllowInReply 106
3.14.19. AllowInReject 106
3.14.20. AddToReply 106
3.14.21. AddToReplyIfNotExist 106
3.14.22. NoIgnoreDuplicates 107
3.14.23. StripFromRequest 107
3.14.24. AddToRequest 107
3.14.25. AddToRequestIfNotExist 107
3.14.26. ClientHook 107
3.14.27. RequireMessageAuthenticator 107
3.14.28. LimitProxyState 108
3.14.29. DynAuthPort 108
3.14.30. UseMessageAuthenticator 108
3.14.31. VsaVendor 108
3.14.32. VsaType 108
3.14.33. VsaTranslateIn 109
3.14.34. VsaTranslateOut 109
3.14.35. NasType 109
3.14.36. SNMPCommunity 111
3.14.37. FramedGroup 112
3.14.38. FramedGroupBaseAddress 112
3.14.39. FramedGroupMaxPortsPerClassC 113

3.14.40. UseContentsForDuplicateDetection 113
3.14.41. DynamicReply 114
3.14.42. IgnoreAcctSignature 114
3.14.43. LivingstonOffs 115
3.14.44. LivingstonHole 115
3.14.45. UseOldAscendPasswords 115
3.15. <ClientListLDAP> 115
3.15.1. BaseDN 116
3.15.2. SearchFilter 116
3.15.3. ClientAttrDef 116
3.15.4. RefreshPeriod 117
3.15.5. FarmWorkerSpacing 117
3.15.6. PostSearchHook 118
3.16. <ClientListSQL> 118
3.16.1. GetClientQuery 118
3.16.2. ClientColumnDef 120
3.16.3. RefreshPeriod 120
3.16.4. DisconnectAfterQuery 121
3.16.5. ConnectionHook 121
3.16.6. ConnectionAttemptFailedHook 121
3.16.7. NoConnectionsHook 121
3.16.8. FarmWorkerSpacing 121
3.17. <SessionDatabase xxxxxx> 122
3.17.1. Identifier 122
3.18. <SessionDatabase SQL> 122
3.18.1. SQL Bind Variables 123
3.18.2. AddQuery 123
3.18.3. DeleteQuery 123
3.18.4. UpdateQuery 123
3.18.5. ClearNasQuery 124
3.18.6. CountQuery 124
3.18.7. ReplaceQuery 124
3.18.8. CountNasSessionsQuery 125
3.18.9. ClearNasSessionQuery 125
3.18.10. SessionIdentifier 125
3.18.11. AddSessionQuery 125
3.18.12. GetSessionQuery 125
3.18.13. UpdateSessionQuery 126
3.18.14. DeleteSessionQuery 126
3.19. <SessionDatabase REDIS> 126
3.19.1. SessionIdentifier 126
3.19.2. SessionKey 126

3.19.3. SessionEndedKey 127
3.19.4. SessionUpdateKey 127
3.19.5. NasKey 127
3.19.6. UserKey 127
3.19.7. AddSessionParamDef 128
3.19.8. UpdateSessionParamDef 128
3.19.9. DeleteSessionParamDef 129
3.19.10. ContextTimeout 130
3.20. <SessionDatabase DBM> 130
3.20.1. Filename 130
3.20.2. DBType 130
3.20.3. SessionIdentifier 131
3.21. <SessionDatabase NULL> 131
3.22. <ServiceDatabase xxxxxx> 131
3.22.1. Identifier 131
3.22.2. Tenant 132
3.23. <ServiceDatabase INTERNAL> 132
3.23.1. Service 132
3.24. <ServiceDatabase SQL> 133
3.24.1. SQL Bind Variables 133
3.24.2. AddServiceQuery 133
3.24.3. GetServiceQuery 134
3.24.4. UpdateServiceQuery 134
3.24.5. DeleteServiceQuery 134
3.24.6. AddSubscriptionQuery 134
3.24.7. GetSubscriptionQuery 135
3.24.8. UpdateSubscriptionQuery 135
3.24.9. DeleteSubscriptionQuery 135
3.25. <Log FILE> 135
3.25.1. Filename 136
3.25.2. Trace 136
3.25.3. LogMicroseconds 137
3.25.4. LogTraceId 137
3.25.5. Identifier 137
3.25.6. IgnorePacketTrace 137
3.25.7. LogFormat 137
3.25.8. LogFormatHook 138
3.26. <Log SYSLOG> 138
3.26.1. Facility 139
3.26.2. Trace 139
3.26.3. IgnorePacketTrace 139
3.26.4. Identifier 139

3.26.5. LogSock 140
3.26.6. LogPath 140
3.26.7. LogHost 140
3.26.8. LogOpt 141
3.26.9. LogIdent 141
3.26.10. LogPort 141
3.26.11. LogFormat 142
3.26.12. LogFormatHook 142
3.26.13. MaxMessageLength 142
3.27. <Log SQL> 142
3.27.1. Table 143
3.27.2. Trace 144
3.27.3. IgnorePacketTrace 144
3.27.4. Identifier 144
3.27.5. LogQuery 144
3.27.6. LogQueryParam 145
3.27.7. MaxMessageLength 145
3.28. <Log EMERALD> 145
3.29. <SNMPAgent> 145
3.29.1. Port 146
3.29.2. BindAddress 146
3.29.3. Community 147
3.29.4. ROCommunity 147
3.29.5. RWCommunity 147
3.29.6. Managers 147
3.29.7. Identifier 147
3.29.8. SNMPVersion 147
3.29.9. PacketTrace 148
3.30. <Realm realmname> 148
3.31. <Handler attribute=value,attribute=value, ...> 149
3.31.1. RewriteUsername 151
3.31.2. RewriteFunction 151
3.31.3. AcctLogFileName 151
3.31.4. AcctLogFileFormat 152
3.31.5. AcctLogFileFormatHook 152
3.31.6. AccountingHandled 152
3.31.7. AccountingAccepted 153
3.31.8. PreProcessingHook 153
3.31.9. PostProcessingHook 153
3.31.10. PreAuthHook 154
3.31.11. PostAuthHook 154
3.31.12. AuthByPolicy 154

3.31.13. AuthBy 154
3.31.14. Identifier 155
3.31.15. StripFromRequest 155
3.31.16. AddToRequest 155
3.31.17. AddToRequestIfNotExist 155
3.31.18. <AuthBy xxxxxx> 155
3.31.19. UsernameCharset 156
3.31.20. RejectHasReason 156
3.31.21. SessionDatabase 157
3.31.22. SessionDatabaseOptions 157
3.31.23. SessionDatabaseUseRewrittenName 158
3.31.24. AcctLog 158
3.31.25. AuthLog 158
3.31.26. <AuthLog xxxxxx> 159
3.31.27. PacketTrace 159
3.31.28. LogRejectLevel 160
3.31.29. DefaultReply 160
3.31.30. StripFromReply 160
3.31.31. AllowInReply 160
3.31.32. AddToReply 161
3.31.33. AddToReplyIfNotExist 161
3.31.34. AddExtraCheck 161
3.31.35. RejectReason 161
3.31.36. DynamicReply 161
3.31.37. MaxSessions 162
3.31.38. AutoClass 162
3.31.39. WtmpFileName 163
3.31.40. FramedGroup 164
3.31.41. PasswordLogFileName 164
3.31.42. ExcludeFromPasswordLog 164
3.31.43. ExcludeRegexFromPasswordLog 164
3.31.44. HandleAscendAccessEventRequest 165
3.32. <AuthBy xxxxxx> 165
3.32.1. Identifier 165
3.32.2. StripFromReply 166
3.32.3. AllowInReply 166
3.32.4. AllowInReject 166
3.32.5. AddToReply 166
3.32.6. AddToReplyIfNotExist 167
3.32.7. DefaultReply 167
3.32.8. FramedGroup 167
3.32.9. NoDefault 167

3.32.10. NoDefaultIfFound 168
3.32.11. DefaultLimit 168
3.32.12. AcceptIfMissing 168
3.32.13. IgnoreIfMissing 168
3.32.14. DefaultSimultaneousUse 169
3.32.15. AuthenProto 169
3.32.16. CaseInsensitivePasswords 170
3.32.17. RejectEmptyPassword 170
3.32.18. AuthenticateAccounting 170
3.32.19. IgnoreAuthentication 171
3.32.20. IgnoreAccounting 171
3.32.21. RcryptKey 171
3.32.22. TranslatePasswordHook 171
3.32.23. CheckPasswordHook 172
3.32.24. AutoMPPEKeys 172
3.32.25. PacketTrace 172
3.32.26. CachePasswords 173
3.32.27. CachePasswordExpiry 173
3.32.28. CachePasswordKey 173
3.32.29. CacheReplyHook 174
3.32.30. ClearTextTunnelPassword 174
3.32.31. NoCheckPassword 174
3.32.32. ConsumePassword and LeavePassword 174
3.32.33. AuthenticateAttribute 176
3.32.34. UsernameMatchesWithoutRealm 176
3.32.35. Blacklist 176
3.32.36. EAPErrorReject 176
3.32.37. PasswordPrompt 176
3.32.38. PostAuthHook 176
3.32.39. AllowNULInUsername 177
3.32.40. AddExtraCheck 177
3.32.41. RejectReason 177
3.32.42. Fork 177
3.32.43. UseAddressHint 177
3.32.44. DynamicReply 178
3.32.45. DynamicCheck 178
3.33. <AuthBy ACE> 179
3.33.1. Using RSA Security token cards to log in with AuthBy ACE 180
3.33.2. ConfigDirectory 181
3.33.3. Timeout 181
3.33.4. EnableFastPINChange 181
3.34. <AuthBy TEST> 181

3.34.1. Identifier 181
3.35. <AuthBy FILE> 181
3.35.1. Filename 182
3.35.2. GroupFilename 182
3.35.3. Nocache 182
3.36. <AuthBy DBFILE> 182
3.36.1. Filename 183
3.36.2. DBType 183
3.37. <AuthBy CDB> 184
3.37.1. Filename 184
3.38. <AuthBy GROUP> 184
3.38.1. AuthByPolicy 185
3.38.2. RewriteUsername 186
3.38.3. StripFromRequest 186
3.38.4. AddToRequest 186
3.38.5. AddToRequestIfNotExist 187
3.38.6. HandleAcctStatusTypes 187
3.39. <AuthBy UNIX> 187
3.39.1. Filename 188
3.39.2. Match 188
3.39.3. GroupFilename 188
3.39.4. Nocache 189
3.40. <AuthBy EXTERNAL> 189
3.40.1. Command 189
3.40.2. Interpreting command stdin, stdout, and exit status 189
3.40.3. DecryptPassword 190
3.40.4. ResultInOutput 190
3.41. <AuthBy SQL> 190
3.41.1. AuthSelect 191
3.41.2. AuthSelectParam 192
3.41.3. AuthColumnDef 192
3.41.4. AccountingTable 194
3.41.5. EncryptedPassword 194
3.41.6. HandleAcctStatusTypes 195
3.41.7. AccountingStartsOnly 195
3.41.8. AccountingAlivesOnly 195
3.41.9. AccountingStopsOnly 195
3.41.10. DateFormat 196
3.41.11. AcctColumnDef 196
3.41.12. AuthSQLStatement 199
3.41.13. AcctSQLStatement 199
3.41.14. AcctInsertQuery 199

3.41.15. AcctFailedLogFileName 199
3.41.16. AcctLogFileFormat 200
3.41.17. AcctLogFileFormatHook 200
3.41.18. PostAuthSelectHook 200
3.41.19. GroupMembershipQuery 201
3.41.20. GroupMembershipQueryParam 201
3.41.21. AcctTotalQuery 201
3.41.22. AcctTotalSinceQuery 201
3.41.23. AcctTotalOctetsSinceQuery 201
3.41.24. AcctTotalOctetsQuery 201
3.41.25. AcctTotalGigawordsSinceQuery 202
3.41.26. AcctTotalGigawordsQuery 202
3.41.27. NullPasswordMatchesAny 202
3.42. <AuthBy RADIUS> 202
3.42.1. Failure algorithm 204
3.42.2. Host 205
3.42.3. Secret 206
3.42.4. AuthPort 206
3.42.5. AcctPort 207
3.42.6. OutPort 207
3.42.7. Retries 207
3.42.8. RetryTimeout 207
3.42.9. KeepaliveTimeout 208
3.42.10. KeepaliveNoreplyTimeout 208
3.42.11. KeepaliveRequestType 208
3.42.12. AddToKeepaliveRequest 208
3.42.13. UseStatusServerForFailureDetect 209
3.42.14. FailureBackoffTime 209
3.42.15. MaxFailedRequests 209
3.42.16. MaxFailedGraceTime 209
3.42.17. HostStateChangeHook 209
3.42.18. MaxTargetHosts 210
3.42.19. StripFromRequest 210
3.42.20. AddToRequest 210
3.42.21. AddToRequestIfNotExist 211
3.42.22. RequireMessageAuthenticator 211
3.42.23. NoForwardAuthentication 211
3.42.24. NoForwardAccounting 211
3.42.25. HandleAcctStatusTypes 211
3.42.26. LocalAddress 211
3.42.27. ReplyHook 212
3.42.28. NoReplyHook 213

3.42.29. ForwardHook 213
3.42.30. ReplyTimeoutHook 214
3.42.31. IgnoreReject 214
3.42.32. NoReplyReject 214
3.42.33. Asynchronous 214
3.42.34. Synchronous 215
3.42.35. UseExtendedIds 215
3.42.36. UseOldAscendPasswords 216
3.42.37. ServerHasBrokenPortNumbers 216
3.42.38. ServerHasBrokenAddresses 216
3.42.39. CachePasswords 217
3.42.40. CacheOnNoReply 217
3.42.41. IgnoreReplySignature 217
3.42.42. IgnoreAccountingResponse 218
3.42.43. AcctFailedLogFileName 218
3.42.44. AcctLogFileFormat 218
3.42.45. AcctLogFileFormatHook 219
3.42.46. AccountingStartsOnly 219
3.42.47. AccountingAlivesOnly 219
3.42.48. AccountingStopsOnly 219
3.42.49. ClearTextTunnelPassword 219
3.42.50. AllowInRequest 219
3.42.51. DisableMTUDiscovery 220
3.42.52. Gossip 220
3.42.53. NoKeepaliveTimeoutForChildInstances 220
3.42.54. GossipNoReply 220
3.42.55. GossipHostDown 220
3.42.56. GossipHostUp 220
3.43. <Host xxxxxx> within <AuthBy RADIUS> 220
3.43.1. BogoMips 221
3.44. <AuthBy RADMIN> 221
3.44.1. AuthSelect 222
3.44.2. LogQuery 222
3.44.3. MaxMessageLength 222
3.44.4. UserAttrQuery 223
3.44.5. ServiceAttrQuery 223
3.44.6. AttrQueryParam 223
3.44.7. IncrementBadloginsQuery 223
3.44.8. ClearBadloginsQuery 223
3.44.9. MaxBadLogins 223
3.45. <AuthBy EMERALD4> 224
3.45.1. AuthSelect 224

3.45.2. ConcurrencyControl 224
3.45.3. TimeBanking 224
3.45.4. HonourServers 224
3.45.5. HonourServerPortAccess 224
3.45.6. HonourDNISGroups 224
3.45.7. HonourRoamServers 224
3.45.8. DNISGroupQuery 225
3.45.9. PortAccessQuery 225
3.45.10. RadUserQuery 225
3.45.11. ClientQuery 225
3.45.12. RoamQuery 225
3.46. <AuthBy PLATYPUS> 225
3.47. <AuthBy LDAP2> 226
3.47.1. Using <AuthBy LDAP2> with Microsoft Active Directory 226
3.47.2. BaseDN 227
3.47.3. SearchFilter 227
3.47.4. EncryptedPasswordAttr 227
3.47.5. PasswordAttr 228
3.47.6. UsernameAttr 228
3.47.7. AuthAttrDef 229
3.47.8. AttrsWithBaseScope 230
3.47.9. PostSearchHook 230
3.47.10. ServerChecksPassword 231
3.47.11. UnbindAfterServerChecksPassword 231
3.47.12. AuthCheckDN 232
3.47.13. MaxRecords 232
3.47.14. GetNovellUP 232
3.47.15. GroupSearchFilter 232
3.47.16. GroupNameCN 232
3.47.17. GroupBaseDN 233
3.47.18. CheckAttr 233
3.47.19. ReplyAttr 233
3.48. <AuthBy SYSTEM> 234
3.48.1. UseGetspnam and UseGetspnamf 234
3.49. <AuthBy TACACSPLUS> 234
3.49.1. Host 234
3.49.2. Key 234
3.49.3. Port 234
3.49.4. Timeout 235
3.49.5. AuthType 235
3.50. <AuthBy PAM> 235
3.50.1. Service 235

3.50.2. UsePamEnv 235
3.50.3. PasswordPrompt 236
3.51. <AuthBy ADSI> 236
3.51.1. BindString 236
3.51.2. AuthUser 237
3.51.3. AuthFlags 237
3.51.4. AuthAttrDef 237
3.51.5. GroupBindString 238
3.51.6. GroupUserBindString 238
3.51.7. CheckGroupServer 239
3.51.8. CheckGroup 239
3.52. <AuthBy PORTLIMITCHECK> 239
3.52.1. CountQuery 240
3.52.2. SessionLimit 240
3.52.3. ClassForSessionLimit 240
3.52.4. LimitQuery 241
3.52.5. IgnoreErrors 241
3.53. <AuthBy DYNADDRESS> 241
3.53.1. AddressAllocator 242
3.53.2. PoolHint 242
3.53.3. MapAttribute 243
3.53.4. MapResultHook 243
3.53.5. RunWhenMissing 244
3.54. <AuthBy ROUNDROBIN>, <AuthBy VOLUMEBALANCE>, <AuthBy LOADBALANCE>, <AuthBy HASHBALANCE>, <AuthBy EAPBALANCE> 244
3.54.1. <AuthBy ROUNDROBIN> 244
3.54.2. <AuthBy VOLUMEBALANCE> 245
3.54.3. <AuthBy LOADBALANCE> 245
3.54.4. <AuthBy HASHBALANCE> 246
3.54.5. <AuthBy EAPBALANCE> 246
3.55. <AuthBy LDAPRADIUS> 247
3.55.1. BaseDN 247
3.55.2. SearchFilter 248
3.55.3. NumHosts 248
3.55.4. HostAttrDef 248
3.56. <AuthBy SQLRADIUS> 249
3.56.1. HostSelect 249
3.56.2. HostSelectParam 251
3.56.3. NumHosts 251
3.56.4. StartHost 252
3.56.5. HostColumnDef 252
3.57. <AuthBy INTERNAL> 252

3.57.1. DefaultResult 254
3.57.2. AuthResult 254
3.57.3. AcctResult 254
3.57.4. AcctStartResult 254
3.57.5. AcctStopResult 254
3.57.6. AcctAliveResult 254
3.57.7. RequestHook 254
3.57.8. AuthHook 255
3.57.9. AcctHook 255
3.57.10. AcctStartHook 255
3.57.11. AcctStopHook 255
3.57.12. AcctAliveHook 255
3.57.13. AcctOtherHook 255
3.57.14. OtherHook 255
3.57.15. StripFromRequest 255
3.57.16. AddToRequest 255
3.57.17. AddToRequestIfNotExist 256
3.58. <AuthBy POP3> 256
3.58.1. Host 256
3.58.2. Port 256
3.58.3. LocalAddr 256
3.58.4. AuthMode 257
3.58.5. Timeout 257
3.58.6. Debug 257
3.58.7. SSLVerify 257
3.58.8. SSLCAFile 257
3.58.9. SSLCAPath 257
3.58.10. SSLCAClientCert 258
3.58.11. SSLCAClientKey 258
3.58.12. SSLCAClientKeyPassword 258
3.59. <AuthBy IMAP> 258
3.59.1. Host 258
3.59.2. Port 259
3.59.3. LocalAddr 259
3.59.4. Timeout 259
3.59.5. Debug 259
3.59.6. SSLVerify 259
3.59.7. SSLCAFile 259
3.59.8. SSLCAPath 259
3.59.9. SSLCAClientCert 260
3.59.10. SSLCAClientKey 260
3.59.11. SSLCAClientKeyPassword 260

3.60. <AuthBy LSA> 260
3.60.1. Domain 261
3.60.2. DefaultDomain 261
3.60.3. Workstation 261
3.60.4. ProcessName 262
3.60.5. Origin 262
3.60.6. Source 262
3.60.7. LSARewriteHook 262
3.60.8. Group 262
3.60.9. DomainController 262
3.61. <AuthBy SOAP> 263
3.61.1. Endpoint 263
3.61.2. URI 263
3.61.3. SOAPTrace 263
3.61.4. Timeout 264
3.62. <AuthBy OTP> 264
3.62.1. ChallengeHook 265
3.62.2. VerifyHook 265
3.62.3. PasswordPattern 266
3.62.4. ContextTimeout 266
3.63. <AuthBy RSAAM> 266
3.63.1. Host 268
3.63.2. Endpoint 268
3.63.3. Protocol 269
3.63.4. URI 269
3.63.5. Policy 269
3.63.6. SessionUsername 269
3.63.7. SessionPassword 269
3.63.8. ChallengeHasPrompt 269
3.63.9. SessionRealm 270
3.63.10. Timeout 270
3.63.11. SOAPTrace 270
3.63.12. Message 270
3.63.13. SSLVerify 270
3.63.14. SSLCAFile 270
3.63.15. SSLCAPath 270
3.63.16. SSLVerifyCNName and SSLVerifyCNScheme 271
3.63.17. SSL_CertificateFile 271
3.63.18. SSL_PrivateKeyFile 271
3.64. <AuthBy SQLDIGIPASS> 271
3.64.1. AuthSelect 274
3.64.2. UpdateQuery 274

3.64.3. ITimeWindow 274
3.64.4. IThreshold 274
3.64.5. SyncWindow 274
3.64.6. CheckChallenge 275
3.64.7. ChkInactDays 275
3.64.8. DeriveVector 275
3.64.9. EventWindow 275
3.64.10. HSMSlotId 275
3.64.11. StorageKeyId 275
3.64.12. TransportKeyId 275
3.64.13. StorageDeriveKey1, StorageDeriveKey2, StorageDeriveKey3, StorageDeriveKey4 275
3.64.14. ChallengeMessage 275
3.64.15. SupportVirtualDigipass 276
3.64.16. VirtualTokencodeHook 276
3.64.17. ChallengeTimeout 276
3.65. <AuthBy LDAPDIGIPASS> 276
3.65.1. SearchFilter 277
3.65.2. UsernameAttr 277
3.65.3. TokenDataAttr 277
3.65.4. MaxRecords 277
3.65.5. BaseDN 277
3.65.6. Vasco Controller Library parameters 278
3.66. <AuthBy LDAP_APS> 278
3.66.1. PasswordServerAddress 279
3.67. <AuthBy REST> 279
3.67.1. RestAuthRequestDef 280
3.67.2. RestAcctRequestDef 280
3.67.3. NoReplyReject 281
3.67.4. ServerChecksPassword 281
3.67.5. RestAuthReplyDef 281
3.67.6. EncryptedPasswordAttr 282
3.67.7. PasswordAttr 282
3.67.8. HandleAcctStatusTypes 283
3.68. <AuthBy URL> 283
3.68.1. AuthUrl 283
3.68.2. AcctUrl 284
3.68.3. UrlMethod 284
3.68.4. Debug 284
3.68.5. Timeout 284
3.68.6. UserParam 284
3.68.7. PasswordParam 284
3.68.8. AuthOKKeyword 284

3.68.9. AuthChallengeKeyword 285
3.68.10. BadUserKeyword 285
3.68.11. BadPasswordKeyword 285
3.68.12. PasswordEncryption 285
3.68.13. ChapChallengeParam 285
3.68.14. ChapResponseParam 285
3.68.15. MSChapChallengeParam 285
3.68.16. MSChapResponseParam 285
3.68.17. MSChapV2ChallengeParam 285
3.68.18. MSChapV2ResponseParam 285
3.68.19. CopyRequestItem 286
3.68.20. CopyReplyItem 286
3.69. <AuthBy KRB5> 286
3.69.1. KrbRealm 286
3.69.2. KrbServerRealm 286
3.69.3. KrbKeyTab 286
3.69.4. KrbService 287
3.69.5. KrbServer 287
3.70. <AuthBy MULTICAST> 287
3.70.1. LoopDetection 287
3.71. <AuthBy RADSEC> 288
3.71.1. Host 290
3.71.2. Secret 290
3.71.3. Port 290
3.71.4. LocalAddress and LocalPort 290
3.71.5. NoreplyTimeout 290
3.71.6. KeepaliveTimeout 290
3.71.7. KeepaliveNoreplyTimeout 291
3.71.8. KeepaliveRequestType 291
3.71.9. AddToKeepaliveRequest 291
3.71.10. UseStatusServerForFailureDetect 291
3.71.11. StripFromRequest 292
3.71.12. AddToRequest 292
3.71.13. AddToRequestIfNotExist 292
3.71.14. IgnoreReject 292
3.71.15. IgnoreAccountingResponse 292
3.71.16. HandleAcctStatusTypes 292
3.71.17. AcctFailedLogFileName 293
3.71.18. AcctLogFileFormat 293
3.71.19. AcctLogFileFormatHook 293
3.71.20. ReplyHook 294
3.71.21. NoReplyHook 294

3.71.22. NoReplyReject 295
3.71.23. Asynchronous 295
3.71.24. ForwardHook 295
3.71.25. NoForwardAuthentication 296
3.71.26. NoForwardAccounting 296
3.71.27. AllowInRequest 296
3.71.28. MaxBufferSize 296
3.71.29. DisconnectTraceLevel 296
3.71.30. ReconnectTimeout 296
3.71.31. StreamStateChangeHook 296
3.71.32. FailureBackoffTime 297
3.71.33. Protocol 297
3.71.34. ConnectOnDemand 297
3.71.35. Gossip 297
3.71.36. NoKeepaliveTimeoutForChildInstances 298
3.71.37. GossipNoReply 298
3.71.38. GossipHostDown 298
3.71.39. GossipHostUp 298
3.71.40. ProxyAlgorithm 298
3.71.41. HashAttributes 299
3.72. <Host xxxxxx> within <AuthBy RADSEC> 299
3.73. <AuthBy SASLAUTHD> 300
3.73.1. SocketPath 301
3.73.2. Service 301
3.74. <AuthBy NTLM> 301
3.74.1. Domain 302
3.74.2. DefaultDomain 302
3.74.3. NtlmAuthProg 302
3.74.4. UsernameMatchesWithoutRealm 303
3.74.5. NtlmRewriteHook 303
3.74.6. UsernameFormat 303
3.74.7. DomainFormat 303
3.75. <AuthBy DNSROAM> 303
3.75.1. RewriteTargetRealm 305
3.75.2. RedespatchIfNoTarget 305
3.75.3. HandleAcctStatusTypes 305
3.75.4. <Route> parameters 305
3.75.5. <AuthBy RADIUS> parameters 306
3.75.6. <AuthBy RADSEC> parameters 306
3.76. <Route> 307
3.76.1. Realm 307
3.76.2. Address 308

3.76.3. Transport 308
3.76.4. Protocol 308
3.76.5. Port 308
3.76.6. UseTLS and TLS_Protocols 308
3.76.7. Secret 308
3.76.8. <AuthBy RADIUS> parameters 308
3.76.9. <AuthBy RADSEC> parameters 309
3.77. <AuthBy SAFEWORD> 311
3.77.1. Host 311
3.77.2. Port 311
3.77.3. SSLVersion 311
3.77.4. SSLCipherList 311
3.77.5. SSLVerify, SSLCAFile, SSLCAPath, SSLCAClientCert, SSLCAClientKey, SSLCAClientKeyPassword 312
3.78. <AuthBy FREERADIUSSQL> 312
3.78.1. ConvertCleartextPassword 312
3.78.2. AuthCheck 312
3.78.3. AuthReply 312
3.78.4. AuthGroupCheck 313
3.78.5. AuthGroupReply 313
3.78.6. AcctOnoffQuery, AcctStartQuery, AcctStartQueryAlt, AcctUpdateQuery, AcctUpdateQueryAlt, AcctStopQuery, AcctStopQueryAlt 313
3.78.7. AcctFailedLogFileName, AcctLogFileFormat and AcctLogFileFormatHook 313
3.79. <AuthBy RADIUSdeskSQL> 313
3.79.1. RealmChkDynamicClients 317
3.79.2. RealmChkNas 317
3.79.3. AutoRegister 317
3.79.4. RegUrl 317
3.79.5. RegSslCaFile 317
3.79.6. RegToken 318
3.79.7. RegCloudId 318
3.79.8. RegProfile 318
3.79.9. RegRealm 318
3.79.10. AutoRegisterHook 319
3.79.11. RecordExtraAttr1 319
3.79.12. AcctGroupname 319
3.79.13. AcctGroupId 319
3.79.14. ConvertCleartextPassword, AuthCheck, AuthReply and other <AuthBy FREERADIUSSQL> parameters 319
3.80. <AuthBy HTGROUP> 320
3.80.1. GroupFilename 320
3.81. <AuthBy HOTSPOT> 320
3.81.1. ServiceId 320

3.81.2. SubscriptionId 320
3.81.3. SessionId 321
3.81.4. ServiceAttribute 321
3.81.5. ServiceAttributePrefix 321
3.81.6. SubscriptionAttribute 321
3.81.7. ServiceAttributePrefix 321
3.81.8. SessionAttribute 321
3.81.9. SessionAttributePrefix 321
3.81.10. DefaultService 321
3.81.11. DefaultServiceDefinition 322
3.81.12. ConfirmSubscription 322
3.81.13. ConfirmationMessage 322
3.81.14. PreProvisionSubscription 322
3.81.15. PreProvisionSession 322
3.81.16. AuthBy 322
3.81.17. ServiceDatabase 322
3.81.18. SessionDatabase 323
3.81.19. PoolIPv4Attribute 323
3.81.20. PoolIPv6Attribute 323
3.81.21. UsageMonitoring 323
3.81.22. ReplyWithBandwidthControl 323
3.81.23. UploadRateAttribute 323
3.81.24. DownloadRateAttribute 324
3.81.25. ReplyWithDataQuota 324
3.81.26. DataLimitAttribute 324
3.81.27. DataLimitGigawordsAttribute 324
3.81.28. ChargeHook 324
3.81.29. ReplyAdjustHook 324
3.81.30. SessionQuotaSupplyFraction 325
3.81.31. SessionQuotaResupplyThreshold 325
3.81.32. SessionQuotaResupplyTimeMin 325
3.81.33. SessionQuotaResupplyDataMin 325
3.82. <AuthBy FIDELIO> 325
3.82.1. Protocol 326
3.82.2. Port 326
3.82.3. Host 326
3.82.4. Baudrate 326
3.82.5. Databits 326
3.82.6. Parity 327
3.82.7. Stopbits 327
3.82.8. Handshake 327
3.82.9. ReadCharTimeout 327

3.82.10. TransmitTimeout 327
3.82.11. ReconnectTimeout 327
3.82.12. InterfaceFamily 327
3.82.13. FieldSeparator 327
3.82.14. BindAddress 327
3.82.15. MaxBufferSize 327
3.82.16. UseChecksums 327
3.82.17. LinkRecords 328
3.82.18. GuestNameField 328
3.82.19. GuestPasswordField 328
3.82.20. ComputeCostHook 328
3.82.21. UserPasswordHook 328
3.82.22. CentsPerSecond 328
3.82.23. PostingRecordID 328
3.82.24. PostingExtraFields 329
3.82.25. MessageHook 329
3.82.26. CheckoutGraceTime 329
3.82.27. HandleAcctStatusTypes 329
3.83. <AuthBy HOTSPOTFIDELIO> 330
3.83.1. BlockDuration 330
3.83.2. BlockPrice 330
3.83.3. PostSendQuery 330
3.83.4. PostSendQueryParam 331
3.83.5. PostAnswerQuery 331
3.83.6. PostAnswerQueryParam 331
3.84. <AuthBy FIDELIOHOTSPOT> 331
3.84.1. BlockDuration 332
3.84.2. BlockPrice 332
3.84.3. ServiceAttribute 332
3.84.4. ServiceAttributePrefix 333
3.84.5. ConfirmUpgradeOrRenew 333
3.84.6. ConfirmationMessage 333
3.84.7. ConfirmationQuery 333
3.84.8. ConfirmationQueryParam 333
3.84.9. PostSendQuery 333
3.84.10. PostSendQueryParam 334
3.84.11. PostAnswerQuery 334
3.84.12. PostAnswerQueryParam 334
3.84.13. GetServiceQuery 334
3.84.14. GetServiceQueryParam 335
3.84.15. GetCurrentServiceQuery 335
3.84.16. GetCurrentServiceQueryParam 335

3.84.17. AddSessionQuery 335
3.84.18. AddSessionQueryParam 335
3.84.19. GetSessionQuery 336
3.84.20. GetSessionQueryParam 336
3.84.21. UpdateSessionQuery 336
3.84.22. UpdateSessionQueryParam 336
3.85. <AuthBy PRESENCESQL> 337
3.86. <AuthBy HANDLER> 337
3.86.1. HandlerId 337
3.87. <AuthBy WIMAX> 337
3.87.1. KeyLifetime 338
3.87.2. HAPassword 338
3.87.3. ProfileHotlining 338
3.87.4. RulebasedHotlining 338
3.87.5. HTTPRedirectionHotlining 338
3.87.6. IPRedirectionHotlining 338
3.87.7. MSKInMPPEKeys 338
3.87.8. GetCachedKeyQuery 338
3.87.9. InsertSessionQuery 339
3.87.10. UpdateSessionQuery 339
3.87.11. GetHotlineProfileQuery 339
3.87.12. GetQosProfileQuery 339
3.88. <AuthBy SQLYUBIKEY> 339
3.88.1. AuthSelect 341
3.88.2. UpdateQuery 341
3.88.3. Require2Factor 341
3.88.4. CheckSecretId 341
3.89. <AuthBy YUBIKEYVALIDATIONSERVER> 342
3.89.1. ValidationServerURL 342
3.89.2. OTPProtocol 342
3.89.3. APIVersion 342
3.89.4. ClientID 343
3.89.5. APIKey 343
3.89.6. OTPCharset 343
3.89.7. Timeout 343
3.89.8. SSLVerify 343
3.89.9. SSLCAPath 344
3.89.10. SSLCAFile 344
3.90. <AuthBy SQLHOTP> 344
3.90.1. AuthSelect 344
3.90.2. AuthSelectParam 345
3.90.3. UpdateQuery 345

3.90.4. UpdateQueryParam 345
3.90.5. Require2Factor 345
3.90.6. DefaultDigits 345
3.90.7. MaxBadLogins 345
3.90.8. BadLoginWindow 346
3.90.9. ResyncWindow 346
3.91. <AuthBy SQLTOTP> 346
3.91.1. AuthSelect 346
3.91.2. AuthSelectParam 347
3.91.3. UpdateQuery 347
3.91.4. UpdateQueryParam 347
3.91.5. Require2Factor 348
3.91.6. EncryptedPIN 348
3.91.7. DefaultDigits 348
3.91.8. MaxBadLogins 348
3.91.9. BadLoginWindow 348
3.91.10. DelayWindow 348
3.91.11. TimeStep 348
3.91.12. TimeStepOrigin 348
3.92. <AuthBy SQLAUTHBY> 348
3.92.1. AuthBySelect 349
3.92.2. AuthBySelectParam 349
3.92.3. Class 349
3.92.4. DefaultParam 349
3.92.5. ParamColumnDef 349
3.93. <AuthBy HEIMDALDIGEST> 350
3.93.1. KdigestPath 350
3.93.2. KdigestSuffix 350
3.93.3. KdigestRealm 350
3.94. <AuthBy SIP2> 350
3.94.1. Port 351
3.94.2. Host 351
3.94.3. Delimiter 351
3.94.4. Timeout 351
3.94.5. Retries 351
3.94.6. FailureBackoffTime 351
3.94.7. LoginUserID 351
3.94.8. LoginPassword 351
3.94.9. LocationCode 352
3.94.10. TerminalPassword 352
3.94.11. Institution 352
3.94.12. SendChecksum 352

3.94.13. VerifyChecksum 352
3.94.14. SIP2Hook 352
3.95. <AuthBy DUO> 353
3.95.1. Hostname, SecretKey and IntegrationKey 353
3.95.2. DefaultFactor 353
3.95.3. PreAuth 353
3.95.4. Address 353
3.95.5. SSLVerify, SSLCAFile and SSLCAPath 354
3.95.6. SSLVerifyCNName and SSLVerifyCNScheme 354
3.95.7. SSLCertificateVerifyHook 354
3.95.8. PollTimerInterval 354
3.95.9. CheckTimerInterval 354
3.95.10. FailureBackoffTime 354
3.95.11. Slots 355
3.95.12. Timeout 355
3.95.13. MaxRequestTime 355
3.95.14. Failmode 355
3.95.15. ProxyHost 355
3.95.16. ProxyPort 355
3.95.17. EndpointPrefix 355
3.95.18. Protocol 355
3.96. <AuthBy DIAMETER> 356
3.96.1. DiaPeerDef 356
3.96.2. RelayAlgorithm 356
3.96.3. Peer 357
3.96.4. SCTPPeer 357
3.96.5. Port 357
3.96.6. DestinationHost 358
3.96.7. DestinationRealm 358
3.96.8. OriginHost 358
3.96.9. OriginRealm 358
3.96.10. PostDiaToRadiusConversionHook 358
3.96.11. PostRadiusToDiaConversionHook 358
3.96.12. EAP_ApplicationId 358
3.96.13. Protocol 359
3.96.14. AuthApplicationIds 359
3.96.15. AcctApplicationIds 359
3.96.16. SupportedVendorIds 359
3.96.17. LocalAddress and LocalPort 359
3.96.18. ReconnectTimeout 359
3.96.19. StreamStateChangeHook 359
3.96.20. DisconnectTraceLevel 360

3.97. <AuthBy RATELIMIT> 360
3.97.1. MaxRate 360
3.97.2. MaxRateResult 360
3.98. <AuthBy GOSSIP> 360
3.98.1. Gossip 361
3.99. <AuthBy DYNAUTH> 361
3.99.1. NasAddrAttribute 361
3.99.2. SessionCheck 361
3.99.3. PreHandlerHook 361
3.100. <AuthBy RADIUSBYATTR> 363
3.100.1. HostsInfoAttribute 363
3.100.2. HostParamDef 363
3.100.3. <Host xxxxxx> within <AuthBy RADIUSBYATTR> 363
3.100.3.1. DynauthPort 364
3.100.3.2. DynAuthSecret 364
3.101. <AuthBy RADIATORPROXY> 364
3.101.1. DynAuthPort 364
3.102. <AuthBy RATELIMITSOURCE> 365
3.102.1. SourceKey1 365
3.102.2. SourceKey2 365
3.102.3. MaxRate1 365
3.102.4. MaxRate2 365
3.102.5. Policer1_Size 365
3.102.6. Policer2_Size 365
3.102.7. TimeWindow1 365
3.102.8. TimeWindow2 365
3.102.9. MaxRateResult 365
3.103. <AuthBy FAILUREPOLICY> 366
3.103.1. DefaultResult 366
3.103.2. PolicyResult 366
3.103.3. FailurePolicyContext 366
3.103.4. ConsecutiveFailures 367
3.103.5. ConsecutiveLockTime 367
3.103.6. ConsecutiveWindow 367
3.103.7. CumulativeFailures 367
3.103.8. CumulativeLockTime 367
3.103.9. CumulativeWindow 367
3.104. <AuthBy SQLFAILUREPOLICY> 368
3.104.1. AddQuery and AddQueryParam 368
3.104.2. UpdateQuery and UpdateQueryParam 368
3.104.3. GetQuery and GetQueryParam 369
3.105. <AuthLog xxxxxx> 369

3.105.1. Identifier 369
3.105.2. LogSuccess 369
3.105.3. LogFailure 370
3.105.4. LogIgnore 370
3.106. <AuthLog FILE> 370
3.106.1. Filename 370
3.106.2. SuccessFormat 371
3.106.3. FailureFormat 371
3.106.4. IgnoreFormat 371
3.106.5. LogFormatHook 372
3.107. <AuthLog SQL> 372
3.107.1. SuccessQuery 372
3.107.2. SuccessQueryParam 373
3.107.3. FailureQuery 373
3.107.4. FailureQueryParam 373
3.107.5. IgnoreQuery 373
3.107.6. IgnoreQueryParam 373
3.108. <AuthLog SYSLOG> 373
3.108.1. Facility 374
3.108.2. Priority 374
3.108.3. SuccessFormat 374
3.108.4. FailureFormat 374
3.108.5. IgnoreFormat 374
3.108.6. LogSock 375
3.108.7. LogPath 375
3.108.8. LogHost 376
3.108.9. LogOpt 376
3.108.10. LogIdent 376
3.108.11. LogPort 377
3.108.12. MaxMessageLength 377
3.109. <AuthLog EVENTLOG> 377
3.109.1. SuccessFormat 377
3.109.2. FailureFormat 377
3.109.3. IgnoreFormat 377
3.110. <AcctLog xxxxxx> 377
3.110.1. LogFormatHook 378
3.111. <AcctLog EVENTLOG> 378
3.111.1. LogFormat 378
3.112. <AcctLog FILE> 378
3.112.1. LogFormat 378
3.112.2. OutputFormat 379
3.112.3. AcctLogOutputDef 379

3.112.4. Filename 379
3.113. <AcctLog SQL> 380
3.113.1. LogQuery 380
3.113.2. LogQueryParam 380
3.114. <AcctLog SYSLOG> 380
3.114.1. Facility 380
3.114.2. Priority 380
3.114.3. LogFormat 381
3.114.4. LogSock 381
3.114.5. LogPath 382
3.114.6. LogPort 382
3.114.7. LogIdent 382
3.114.8. LogOpt 382
3.114.9. LogHost 382
3.114.10. MaxMessageLength 383
3.115. <AddressAllocator SQL> 383
3.115.1. Identifier 384
3.115.2. DefaultLeasePeriod 384
3.115.3. LeaseReclaimInterval 384
3.115.4. FindQuery 384
3.115.5. FindQueryBindVar 385
3.115.6. AllocateQuery 385
3.115.7. AllocateQueryBindVar 385
3.115.8. UpdateQuery 386
3.115.9. UpdateQueryBindVar 386
3.115.10. CheckPoolQuery 386
3.115.11. CheckPoolQueryBindVar 386
3.115.12. AddAddressQuery 386
3.115.13. AddAddressQueryBindVar 387
3.115.14. DeallocateQuery 387
3.115.15. DeallocateQueryBindVar 387
3.115.16. DeallocateByNASQuery 387
3.115.17. DeallocateByNASQueryBindVar 387
3.115.18. ReclaimQuery 387
3.115.19. ReclaimQueryBindVar 387
3.115.20. NoAddressHook 388
3.115.21. NasId 388
3.115.22. DelayedPoolCheckTime 388
3.115.23. AddressPool 388
3.116. <AddressAllocator DHCP> 389
3.116.1. Identifier 390
3.116.2. Host 390

3.116.3. Port 390
3.116.4. LocalAddress 390
3.116.5. DHCPClientIdentifier 391
3.116.6. DefaultLease 391
3.116.7. TimeoutMinimum 391
3.116.8. TimeoutMaximum 391
3.116.9. TimeoutFactor 391
3.116.10. ServerPort 391
3.116.11. Synchronous 391
3.116.12. SubnetSelectionOption 391
3.116.13. UserClass 392
3.116.14. ClientHardwareAddress 392
3.116.15. UseClassForAllocationInfo 392
3.116.16. DHCPHostName 392
3.116.17. DHCPVendorClass 392
3.117. <AddressAllocator DHCPv6> 393
3.117.1. Identifier 393
3.117.2. Host 394
3.117.3. Port 394
3.117.4. ClientPort 394
3.117.5. LocalAddress 394
3.117.6. InterfaceName 394
3.117.7. InterfaceIndex 395
3.117.8. DHCPClientIdentifier 395
3.117.9. DefaultLease 395
3.117.10. Synchronous 395
3.117.11. UserClass 395
3.117.12. AllocateDoneHook 395
3.118. <Resolver> 396
3.118.1. Nameservers 397
3.118.2. Debug 397
3.118.3. Recurse 397
3.118.4. TCPTimeout 398
3.118.5. UDPTimeout 398
3.118.6. TCPPersistent 398
3.118.7. UDPPersistent 398
3.118.8. GetIPV4 398
3.118.9. GetIPV6 398
3.118.10. GetRadius 398
3.118.11. GetRadSec 399
3.118.12. DirectAddressLookup 399
3.118.13. NAPTR-Pattern 399

3.118.14. NegativeCacheTtl 399
3.118.15. FailureBackoffTime 399
3.119. <ServerDIAMETER> 399
3.119.1. Port 400
3.119.2. BindAddress 400
3.119.3. Protocol 400
3.119.4. ReadTimeout 400
3.119.5. OriginHost 401
3.119.6. OriginRealm 401
3.119.7. ProductName 401
3.119.8. AddToRequest 401
3.119.9. DefaultRealm 401
3.119.10. PreHandlerHook 401
3.119.11. SupportedVendorIds 403
3.119.12. ConvertCommand 403
3.119.13. AuthApplicationIds 403
3.119.14. AcctApplicationIds 403
3.119.15. MaxBufferSize 403
3.119.16. DisconnectTraceLevel 403
3.119.17. StreamStateChangeHook 403
3.119.18. StreamMaxClients 404
3.119.19. PacketTrace 404
3.119.20. PostDiaToRadiusConversionHook 405
3.119.21. PostRadiusToDiaConversionHook 405
3.119.22. Clients 405
3.120. <ServerTACACSPLUS> 405
3.120.1. Key 408
3.120.2. EncryptedKey 408
3.120.3. Port 408
3.120.4. BindAddress 408
3.120.5. AuthorizationReplace 409
3.120.6. AuthorizationAdd 409
3.120.7. AddToRequest 409
3.120.8. AuthorizationTimeout 409
3.120.9. DefaultRealm 409
3.120.10. GroupMemberAttr 410
3.120.11. AuthorizeGroup 410
3.120.12. AuthorizeGroupAttr 413
3.120.13. AllowAuthorizeOnly 413
3.120.14. UsernamePrompt 413
3.120.15. PasswordPrompt 413
3.120.16. IdleTimeout 413

3.120.17. AuthenticationStartHook 413
3.120.18. AuthenticationContinueHook 414
3.120.19. AccountingHook 414
3.120.20. PreHandlerHook 414
3.120.21. SingleSession 415
3.120.22. PacketTrace 416
3.120.23. DisconnectTraceLevel 416
3.120.24. StreamStateChangeHook 416
3.120.25. DisconnectWhenIgnore 417
3.120.26. ContextId 417
3.120.27. GroupCacheFile 417
3.120.28. Clients 417
3.121. <ServerRADSEC> 418
3.121.1. Port 419
3.121.2. BindAddress 419
3.121.3. Secret 419
3.121.4. RewriteUsername 419
3.121.5. StripFromRequest 419
3.121.6. AddToRequest 419
3.121.7. AddToRequestIfNotExist 420
3.121.8. AllowInReject 420
3.121.9. DefaultRealm 420
3.121.10. Protocol 420
3.121.11. PreHandlerHook 421
3.121.12. Identifier 422
3.121.13. PacketTrace 422
3.121.14. StatusServer 422
3.121.15. MaxBufferSize 423
3.121.16. DisconnectTraceLevel 423
3.121.17. StreamStateChangeHook 423
3.121.18. StreamMaxClients 423
3.121.19. Clients 424
3.122. <ServerHTTP> 424
3.122.1. Port 425
3.122.2. Username 425
3.122.3. Password 426
3.122.4. AuthBy 426
3.122.5. AuthByPolicy 426
3.122.6. AuthLog 426
3.122.7. AuditTrail 426
3.122.8. LogMicroseconds 426
3.122.9. SessionTimeout 426

3.122.10. LogMaxLines 426
3.122.11. Trace 426
3.122.12. DefaultPrivilegeLevel 426
3.122.13. PageNotFoundHook 426
3.122.14. Clients 426
3.123. <StatsLog xxxxxx> 427
3.123.1. Interval 428
3.123.2. StatsType 428
3.123.3. RateCalculationInterval 429
3.123.4. OutputFormat 429
3.123.5. StatsExcludeObject 429
3.123.6. StatsInclude 430
3.123.7. FarmWorkerSpacing 430
3.124. <StatsLog FILE> 430
3.124.1. Filename 430
3.124.2. Format 431
3.124.3. Header 431
3.125. <StatsLog SQL> 431
3.125.1. InsertQuery 431
3.126. <StatsLog REDIS> 432
3.126.1. StatsKey 432
3.127. <DiaStatsLog xxxxxx> 432
3.127.1. PeerAliveDetectionInterval 432
3.127.2. PeerRemovalThreshold 433
3.128. <DiaStatsLog FILE> 433
3.128.1. Filename 433
3.129. <DiaStatsLog REDIS> 433
3.129.1. StatsKey 433
3.129.2. OutputFormat 433
3.130. <DiaStatsLog SQL> 433
3.130.1. InsertQuery 433
3.130.2. TableName 433
3.131. <MessageLog xxxxxx> 434
3.131.1. Identifier 434
3.131.2. LogSelectHook 434
3.132. <MessageLog FILE> 435
3.132.1. Filename 435
3.132.2. Format 435
3.132.3. Encoding 435
3.133. <Monitor> 436
3.133.1. Port 436
3.133.2. Clients 437

3.133.3. BindAddress 437
3.133.4. AuthBy, <AuthBy xxxxxx> and AuthByPolicy 437
3.133.5. Username 437
3.133.6. Password 438
3.133.7. TraceOnly 438
3.133.8. StatisticsOnly 438
3.133.9. LogMicroseconds 439
3.133.10. LogTraceId 439
3.133.11. LogFarmInstance 439
3.134. <GossipRedis> 439
3.134.1. Host 439
3.134.2. Port 439
3.134.3. Sock 439
3.134.4. Sentinels 440
3.134.5. SentinelService 440
3.134.6. SentinelPort 440
3.134.7. Password 440
3.134.8. Prefix 440
3.134.9. InstanceId 440
3.134.10. DbIndex 440
3.134.11. Timeout 441
3.134.12. FailureBackoffTime 441
3.135. <GossipUDP> 441
4. Running radiusd 441
5. Details on starting Radiator during system start-up 444
5.1. Systemd service unit file 444
5.2. System Service on Windows 445
5.3. Unix SYSV startup script 446
5.4. Using restartWrapper 446
5.5. Using inetd 448
5.6. Using init 448
6. Testing tools 449
6.1. radpwtst 449
6.1.1. radpwtst option file 456
6.1.2. radpwtst examples 456
6.1.3. radpwtst GUI 457
6.2. builddbm 458
6.3. buildsql 460
7. Check and reply items 462
7.1. Check items 462
7.1.1. User-Password, Password 463
7.1.2. Encrypted-Password 466

7.1.3. Realm 466
7.1.4. ExistsInRequest 466
7.1.5. Expiration, ValidTo 467
7.1.6. ValidFrom 467
7.1.7. Auth-Type 468
7.1.8. Group 468
7.1.9. GroupList 469
7.1.10. Group-Authorization 469
7.1.11. Block-Logon-From 469
7.1.12. Block-Logon-To 469
7.1.13. Prefix 469
7.1.14. Suffix 470
7.1.15. Time 470
7.1.16. Simultaneous-Use 470
7.1.17. Connect-Rate 471
7.1.18. NAS-Address-Port-List 471
7.1.19. Client-Id 471
7.1.20. Client-Identifier 472
7.1.21. NasType 472
7.1.22. Request-Type 473
7.1.23. MS-Login-Hours 473
7.1.24. Tagged Tunnel attributes and other tagged attributes 473
7.1.25. EAPType 474
7.1.26. EAPTypeName 474
7.1.27. TunnelledByTTLS 474
7.1.28. TunnelledByPEAP 474
7.1.29. TunnelledByFAST 474
7.1.30. RecvFromAddress 474
7.1.31. RecvFromName 475
7.1.32. RecvName 475
7.1.33. RecvAddress 476
7.1.34. RecvPort 476
7.1.35. Max-All-Session 477
7.1.36. Max-Hourly-Session 477
7.1.37. Max-Daily-Session 477
7.1.38. Max-Monthly-Session 477
7.1.39. Max-All-Octets 477
7.1.40. Max-All-Gigawords 477
7.1.41. Max-Hourly-Octets 477
7.1.42. Max-Hourly-Gigawords 477
7.1.43. Max-Daily-Octets 477
7.1.44. Max-Daily-Gigawords 478

7.1.45. Max-Monthly-Octets 478
7.1.46. Max-Monthly-Gigawords 478
7.1.47. Variable names prefixed with GlobalVar 478
7.1.48. Attributes prefixed with Reply 478
7.1.49. Attributes prefixed with DiaRequest 478
7.1.50. Any other attribute defined in your dictionary 479
7.2. Reply items 480
7.2.1. Framed-Group 480
7.2.2. Ascend-Send-Secret 480
7.2.3. Tunnel-Password and other tagged attributes 481
7.2.4. MS-CHAP-MPPE-Keys 481
7.2.5. MS-MPPE-Send-Key 481
7.2.6. MS-MPPE-Recv-Key 481
7.2.7. Fall-Through 481
7.2.8. Session-Timeout 481
7.2.9. Exec-Program 482
7.2.10. Ascend-Data-Filter, Ascend-Call-Filter 482
7.2.11. Any other attribute defined in your dictionary 484
8. Rewriting user names 484
9. File formats 485
9.1. Dictionary file 485
9.1.1. ATTRIBUTE attrname attrnum type [flags] 485
9.1.2. VALUE attrname valuename value 487
9.1.3. VENDORATTR vendornum attrname attrnum type [flags] 488
9.1.4. VENDOR vendorname vendornumber 488
9.1.5. BEGIN-VENDOR vendorname [parent=attributename] 488
9.1.6. END-VENDOR 489
9.1.7. include filename 489
9.1.8. $INCLUDE filename 489
9.1.9. Available dictionaries 489
9.1.10. Using VSA framework for customised attributes 490
9.2. Flat file user database 490
9.3. DBM user database 491
9.4. Unix password file 491
9.5. Accounting log file 491
9.6. Password log file 492
9.7. Portlist file 492
9.8. Group membership file 493
10. Configuring Radiator with GUI 493
10.1. Login page 494
10.2. Home Page 494
10.3. Administration 495

10.3.1. Server status 495
10.3.2. View log 496
10.3.3. Logout 497
10.4. Configuration 497
10.4.1. Edit 497
10.4.2. Load config file 499
10.4.3. Save 499
10.5. Miscellaneous 500
10.5.1. Licence 500
10.5.2. Support 501
10.5.3. System 502
10.5.4. Perl 503
10.5.5. Modules 504
10.6. Advanced 505
10.6.1. Manual edit 505
10.6.2. Reset server 506
11. Using Gossip framework 507
12. Adding custom AuthBy modules 507
12.1. Loading and configuring 507
12.2. Handling Requests 508
12.3. AuthGeneric 508
12.4. Step-by-step 509
12.5. Class Hierarchy 510
13. Compatibility with other servers 510
14. Execution sequence and hook processing 511
15. Interoperation with iPASS Roaming 513
15.1. iPASS Outbound 514
15.2. iPASS Inbound 515
16. RadSec (RFC 6614) 516
16.1. RadSec Certificate Validation 517
17. Extensible Authentication Protocol (EAP) 518
17.1. EAP MD5-Challenge 519
17.2. EAP One-Time-Password 519
17.3. EAP Generic-Token 519
17.4. EAP TLS 519
17.5. EAP LEAP 520
17.6. EAP TTLS 520
17.7. EAP SIM 521
17.8. EAP AKA 521
17.9. EAP AKA’ 521
17.10. EAP PEAP 521
17.11. EAP MSCHAPV2 522

17.12. EAP PAX 522
17.13. EAP PSK 522
17.14. EAP PWD 523
18. Monitor command language 523
18.1. Object naming 524
18.2. Commands 525
18.2.1. BINARY 525
18.2.2. CHALLENGE 525
18.2.3. DESCRIBE objectname 525
18.2.4. GET objectname 525
18.2.5. HELP 525
18.2.6. ID 525
18.2.7. LIST objectname 525
18.2.8. LOGIN username password 525
18.2.9. RESTART 526
18.2.10. TRACE n 526
18.2.11. TRACE_USERNAME user name 526
18.2.12. SET objectname parameter value 526
18.2.13. STATS objectname 526
18.2.14. QUIT 526
19. Using SQL with various database vendors 527
19.1. General 527
19.2. MySQL and MariaDB 527
19.3. PostgreSQL 528
19.4. Oracle 528
19.5. Microsoft SQL Server 528
19.6. SQLite 529
19.7. Firebird 529
19.8. ODBC 529
19.9. Sybase 530
19.10. InterBase 530
19.11. Informix 530
19.12. CSV 531
19.13. DB2 and other database servers 531
20. Performance and tuning 531
21. Getting help 532
21.1. Support contract holders 533
21.2. No support contract 533
21.3. What to do if you need help 533
21.4. Announcements 533
21.5. Bug reports 534
21.6. RFCs 534

1. Introduction to Radiator® AAA Server

This document describes how to install and configure the Radiator® AAA Server (later Radiator) from Radiator Software.

RADIUS is the de facto standard protocol for authenticating users and recording accounting information. It is commonly used by Wireless Access Points (APs), Terminal Servers, and Network Access Servers (NAS), whenever a user logs on and off network access devices or dial-up Internet service. It is supported and used by most vendors, such as Cisco, Ericsson, Huawei, Juniper, Ruckus, Aruba, Alcatel-Lucent, 3Com, and US Robotics. See RFCs 2865 and 2866 for more details about the RADIUS protocol.

Radiator is a highly configurable and extensible RADIUS, TACACS+, and Diameter server that allows you to easily customise and control how to authenticate users and record accounting information. You can easily integrate Radiator with current and legacy systems and software. Radiator includes special features not found in other servers, such as user name rewriting and vendor-specific RADIUS attributes.

Radiator can authenticate users from passwords held in:

- Remote RADIUS servers (proxying)
- SQL databases, including Oracle, Microsoft SQL Server, PostgreSQL, MySQL, MariaDB, SQLite, Firebird, Sybase, Informix and others
- SQL databases that support ODBC
- LDAP
- Microsoft Active Directory
- RSA Securid
- VASCO Digipass
- YubiKey
- Duo Security
- SIP2, the 3M Standard Interchange Protocol (SIP) 2.0
- RADIUSdesk, a powerful and user-friendly RADIUS management platform
- RAdmin, the RADIUS User Administration system from Radiator Software
- Flat files
- DBM files
- Unix password files and similar formats
- iPASS Roaming Network
- RSA Mobile
- Freeside billing system
- Platypus ISP billing system from ispbilling.com
- Emerald ISP billing system from IEA
- BillMax Unix ISP billing system
- Other ISP billing packages
- PAM
- TacacsPlus

- Your own legacy user database
- External programs
- Any external one-time password system
- Most external SOAP-based authentication methods
- Hotel Property Management Systems
- Other methods contributed by Radiator users

Radiator can record user accounting information in:

- Flat files in standard RADIUS detail file format
- Unix wtmp format files
- SQL databases, including Oracle, Microsoft SQL Server, PostgreSQL, MySQL, MariaDB, SQLite, Firebird, Sybase, Informix and others
- SQL databases that support ODBC
- Remote RADIUS servers
- RADIUSdesk, a powerful and user-friendly RADIUS management platform
- RAdmin, the RADIUS User Administration system from Radiator Software
- Platypus ISP billing system from ispbilling.com
- Emerald ISP billing system from IEA
- BillMax Unix ISP billing system
- Other ISP billing packages
- Your own legacy usage database
- External programs

Radiator provides external interfaces for the following protocols:

- RADIUS (including WiMAX). A full suite of load balancing modules is included.
- TACACS+
- SNMP
- Telnet
- HTTP
- Diameter
- RadSec

Radiator works with a wide range of EAP (Extensible Authentication Protocol) methods, such as the following:

- TLS
- TTLS
- PEAP
- pwd
- MD5
- MSCHAPV2

- OTP
- GTC
- LEAP
- FAST
- PAX
- PSK
- SIM
- AKA
- AKA'

Radiator runs on the following platforms:

- Most Unix and Linux hosts
- Windows 8/8.1/10/11
- Windows Server 2008/2012/2016/2019/2022/2025
- macOS

Radiator is written entirely in Perl and is therefore highly portable. Full source code is supplied.

You need to be familiar with system administration to install Radiator. You also need to have a basic understanding of RADIUS and your network's authentication and accounting requirements in order to configure RADIUS.

‘Radiator’ and the Radiator logo are registered trademarks of Open System Consultants Pty. Ltd., a subsidiary of Radiator Software Oy.

2. Installing and upgrading Radiator

This section provides instructions for installing and upgrading Radiator. Radiator runs on a wide range of platforms and the installation procedure depends on the platform and the type of package selected.

2.1. System requirements

Radiator requires a number prerequisite packages to be installed before you can install it.

2.1.1. Perl

This is the main program that runs Radiator. It must be installed on your Radiator host. Platform-specific methods for installing Perl are discussed below. Radiator requires Perl 5.8.8 or better. Install and test Perl before proceeding further.

Many Unix distributions include Perl as part of the standard installation. Common Windows Perl distributions include Strawberry Perl.

2.1.2. CPAN

Some of the Radiator features require Perl modules that may not be part of the Perl distribution you are using. For example, there is no prebuilt RPM for Red Hat, or the Windows Strawberry Perl distribution does not include the required module. In this case, you need to install the module from CPAN.

On Windows with Strawberry Perl, review Windows installation on page 11 before continuing. You can use the tools the Perl distribution provides.

CPAN stands for **Comprehensive Perl Archive Network**. CPAN is a repository of Perl modules including the core Perl itself. You can use CPAN to search and download Perl modules, view their documentation, source code and change logs.

You can search CPAN for Perl modules with several methods, for example:

- CPAN website [http://www.cpan.org]
- MetaCPAN website [https://metacpan.org]

Perl comes with tools for downloading and building Perl modules automatically. For more information, see How to install CPAN modules [http://www.cpan.org/modules/INSTALL.html]. **`cpan`** and **`cpanm`** are the common tools, they are run from command line:

```
cpan Digest::MD5
```

```
cpanm Digest::MD5
```

If you do not have network access, the general procedure for installing a Perl module is:

1. Unpack the downloaded module in a work area.

   ```
   tar zxvf Digest-MD5-2.55.tar.gz
   ```

2. Move to the module directory. See README, INSTALL, and similar files for additional information.

   ```
   cd Digest-MD5-2.55
   ```

3. Create the makefile.

   ```
   perl Makefile.PL
   ```

4. Build the module.

   ```
   make
   ```

5. Run the tests.

   ```
   make test
   ```

6. Install the module.

   ```
   make install
   ```

2.1.3. Digest::MD5

Radiator requires the Digest::MD5 Perl library package (version 2.02 or later). Most modern versions of Perl already include the Digest::MD5 library.

If it is not present in your Perl distribution, see Section 2.1.2. CPAN on page 3 for how to obtain and install it.

2.1.4. Digest::SHA

Radiator requires the Digest::SHA Perl library package (version 5 or later). Most modern versions of Perl already include the Digest::SHA library.

If it is not present in your Perl distribution, see Section 2.1.2. CPAN on page 3 for how to obtain and install it.

2.1.5. MD4 digest for MSCHAP and MSCHAPv2

If MSCHAP, MSCHAPv2 or EAP-MSCHAP-V2 support is required, Radiator needs a module that provides support for the MD4 digest. Support for MD4 is provided by Radiator Radius::UtilXS and Digest::MD4. There is no need to install the both modules. You can choose either one.

MD4 support was added in version Radius::UtilXS version 2.0. See Section 2.1.9. Radiator Radius::UtilXS on page 6 for more information.

If Digest::MD4 is not present in your Perl distribution, see Section 2.1.2. CPAN on page 3 for how to obtain and install it.

2.1.6. Net::SSLeay and OpenSSL

If support for EAP-TLS, PEAP, EAP-TTLS, EAP-FAST, or any other modules requiring TLS is required, Radiator needs the Net::SSLeay Perl library package. The Perl distributions typically include `Net::SSLeay` as a prepackaged module.

CAUTION

`Net::SSLeay` 1.83 or later is required if you use Radiator with SSL/TLS library that has TLSv1.3 enabled. `Net::SSLeay` 1.92 or later is recommended for all TLS features. Radiator has been tested with TLSv1.3 but TLSv1.3 remains disabled by default.

On Windows with Strawberry Perl, see Section 2.8. Installing and upgrading on Windows on page 11 for more about installing `Net::SSLeay` on Windows.

If it is not present in your Perl distribution, see Section 2.1.2. CPAN on page 3 for how to obtain and install it.

Many TLS-based configurations also require message digest modules. See Section 2.1.5. MD4 digest for MSCHAP and MSCHAPv2 on page 5.

2.1.7. SQL

To use Radiator's modules that utilise SQL for authenticating, recording, accounting, and other purposes, install the DBI Perl library, the DBD Perl library for your particular SQL database. Radiator does not include an SQL database server. Operating system vendors and Windows Perl distributions typically include DBI and DBD libraries for the common SQL databases as prepackaged modules.

To use SQL:

1. Install and test your chosen SQL database server.
2. Install and test DBI.
3. Install and test the DBD Perl module for the installed database server

On Windows with Strawberry Perl, you can use the cpan or cpanm commands to install the appropriate DBD support for your SQL server.

2.1.8. Memory requirements

The Radiator installation requires approximately 32Mb of disk space. RAM requirements depend strongly on your Radiator configuration and the types of authentication methods being used.

If EAP authentication is not used, Radiator typically starts running at about 20Mb and may grow to around 50Mb and plateau as RADIUS requests are received.

If EAP authentication types, such as TLS, TTLS or PEAP, are used, more memory is needed. This is because Radiator must cache per-user TLS session information in its process memory. If you are handling large numbers of TLS authentications, you also need more memory.

2.1.9. Radiator Radius::UtilXS

Radiator's `Radius::UtilXS` package contains functionality that is not readily available as pre-built modules. This includes an alternative for `Digest::MD4` for MSCHAP based authentication methods and SCTP extended API support. Radiator SIM Pack requires special SHA transformation and IMSI decryption support, which are also available in `Radius::UtilXS`.

Radiator's `Radius::UtilXS` package is available from the same location where you can download Radiator from. This package is supported directly by Radiator Software. Availability of `Radius::UtilXS` is logged during *`radiusd`* startup when logging level is set to *info* (Trace 3 or higher number).

2.1.10. SCTP

Stream Control Transmission Protocol (SCTP) support depends on operating system Radiator runs on. When the OS supports SCTP, you can configure Radiator to use one-to-one style sockets.

If you install Radiator's Radius::UtilXS package for SCTP extended API support, SCTP multihoming is automatically enabled for Stream based clauses, such as AuthBy RADSEC and ServerDIAMETER. Multiple addresses configured with parameters BindAddress on page 400, LocalAddress on page 96 and SCTPPeer on page 357 are used with the extended API to bind and connect to all addresses with one call.

Radiator's Radius::UtilXS package is available from the same location where you can download Radiator from. See Section 2.1.9. Radiator Radius::UtilXS on page 6 for more information. Availability of SCTP extended API is logged during *`radiusd`* startup when logging level is set to debug (Trace 4 or higher number).

2.2. Updating from demo or locked version to full version

If your locked version was installed from Linux `el10`, `el9`, `el8` RPM, Linux `suse` RPM, Linux `deb`, or Windows MSI package, update as follows:

Red Hat Enterprise Linux 10 RPM: `radiator-locked-4.xx-nn.el10.noarch.rpm`
: update by installing `radiator-4.xx-nn.el10.noarch.rpm`

Red Hat Enterprise Linux 9 RPM: `radiator-locked-4.xx-nn.el9.noarch.rpm`
: update by installing `radiator-4.xx-nn.el9.noarch.rpm`

Red Hat Enterprise Linux 8 RPM: `radiator-locked-4.xx-nn.el8.noarch.rpm`
: update by installing `radiator-4.xx-nn.el8.noarch.rpm`

SUSE Linux Enterprise Server 15 SP3 or openSUSE Leap 15.3 RPM: `radiator-locked-4.xx-nn.suse.noarch.rpm`
: update by installing `radiator-4.xx-nn.suse.noarch.rpm`

Linux Ubuntu 20.04, 22.04, 24.04, Debian 11 or 12 deb: `radiator-locked_4.xx-nn_all.deb`
: update by installing `radiator_4.xx-nn_all.deb`

Windows MSI: `Radiator-Locked-4.xx.nn.msi`
: update by installing `Radiator-4.xx.nn.msi`

If you have installed Radiator from `.zip` or `.tgz` source code package, update by doing source code installation using a full version source code package.

If your locked version was running during update, you need to restart Radiator

```
sudo systemctl restart radiator
```

2.3. Installing and upgrading on Linux: Red Hat Enterprise Linux, CentOS, Oracle Linux, AlmaLinux and Rocky Linux

These packages have been tested on Red Hat Enterprise Linux 8, 9 and 10, and compatible systems such as CentOS, Oracle Linux, AlmaLinux and Rocky Linux.

To install Radiator:

1. Download the distribution package for your operating system from Radiator downloads [https://downloads.radiatorsoftware.com/packages/]
2. Install the package. On RHEL 8 and compatible systems use package that ends with `el8.noarch.rpm`:

   ```
   sudo yum install ./radiator-4.xx-nn.el8.noarch.rpm
   ```

   On RHEL 9 and compatible systems use package that ends with `el9.noarch.rpm`:

   ```
   sudo yum install ./radiator-4.xx-nn.el9.noarch.rpm
   ```

   On RHEL 10 and compatible systems use package that ends with `el10.noarch.rpm`:

   ```
   sudo yum install ./radiator-4.xx-nn.el10.noarch.rpm
   ```

3. Start Radiator and set it to start automatically after you reboot your Linux server.

   ```
   sudo systemctl start radiator
   sudo systemctl enable radiator
   ```

4. Test authentication. You should see OK printed to the screen 3 times. In case of a problem, check the Radiator process log, by default `/var/log/radiator/radiator.log`, for any errors during Radiator start.

   ```
   /opt/radiator/radiator/radpwtst
   ```

5. Edit `/etc/radiator/radiator.conf` to suit your site and needs. Remember to restart Radiator after configuration change with **`systemctl restart radiator`**. This reference manual describes the options and parameters.

The package creates the following system user, system group and an empty home directory for Radiator service:

- **`radiator`**
- **`radiator`**
- `/var/lib/radiator`

By default, it creates the following directories for configuration, logs, Radiator itself, its utilities and other files. See `/opt/radiator/radiator/goodies/` for configuration samples:

- `/etc/radiator/`
- `/var/log/radiator/`
- `/opt/radiator/`
- `/usr/share/doc/radiator`

Default log rotate configuration is installed as

- `/etc/logrotate.d/radiator`

You can find documentation, additional dictionaries, and the goodies collection in `/opt/radiator/radiator/` directory.

Upgrading from an el7, el8, el9 or el10 RPM

Upgrades between el7, el8, el9 and el10 packages do not require any special commands. Use **`yum install`** or any other commands that you typically use to upgrade packages. Note: remember to restart radiator after each upgrade.

Upgrading from a generic, non-el RPM to an el7 RPM

Note: this applies only to very old installations. To upgrade from generic non-el RPM packaged Radiator `Radiator-x.yy-z.noarch.rpm` to el7 RPM, you need to do the following:

1. Stop the old Radiator instance.

   ```
   sudo /etc/init.d/radiator stop
   ```

2. Download the distribution package for your operating system from Radiator downloads [https://downloads.radiatorsoftware.com/packages/]

3. Install the new el7 RPM. The installer removes non-el RPM and notes that old Radiator files are removed. Configuration files installed by the old package will be saved as `dictionary.rpmsave`, `radius.cfg.rpmsave` and `users.rpmsave`. Only files that were locally changed are saved as `.rpmsave` files.

   ```
   sudo yum install ./radiator-4.22-nnn.el7.noarch.rpm
   ```

4. Copy `radius.cfg.rpmsave` to `radiator.conf`. This is the configuration file used by new RPMs. Review log file path, dictionary file name and to match the new RPM. See if the possible other `.rpmsave` files need to be addressed.

5. Start the server.

   ```
   sudo systemctl start radiator
   ```

2.4. Installing and upgrading on Linux: SUSE Linux Enterprise Server and openSUSE Leap

These packages have been tested on SUSE Linux Enterprise Server 15 SP3 and openSUSE Leap 15.6.

To install Radiator:

1. Download the distribution package for your operating system from Radiator downloads [https://downloads.radiatorsoftware.com/packages/]

2. Install the package. Use package that ends with `.suse.noarch.rpm`:

   ```
   sudo zypper install ./radiator-4.xx-nn.suse.noarch.rpm
   ```

3. Start Radiator and set it to start automatically after you reboot your Linux server.

   ```
   sudo systemctl start radiator
   sudo systemctl enable radiator
   ```

4. Test authentication. You should see OK printed to the screen 3 times. In case of a problem, check the Radiator process log, by default `/var/log/radiator/radiator.log`, for any errors during Radiator start.

   ```
   /opt/radiator/radiator/radpwtst
   ```

5. Edit `/etc/radiator/radiator.conf` to suit your site and needs. Remember to restart Radiator after configuration change with **`systemctl restart radiator`**. This reference manual describes the options and parameters.

The package creates the following system user, system group and an empty home directory for Radiator service:

- **`radiator`**
- **`radiator`**
- `/var/lib/radiator`

By default, it creates the following directories for configuration, logs, Radiator itself, its utilities and other files. See `/opt/radiator/radiator/goodies/` for configuration samples:

- `/etc/radiator/`
- `/var/log/radiator/`
- `/opt/radiator/`
- `/usr/share/doc/packages/radiator`

Default log rotate configuration is installed as

- `/etc/logrotate.d/radiator`

You can find documentation, additional dictionaries, and the goodies collection in `/opt/radiator/radiator/` directory.

Upgrading from an RPM

Upgrades between SUSE/openSUSE packages do not require any special commands. Use **`zypper install`** or any other commands that you typically use to upgrade packages. Note: remember to restart radiator after each upgrade.

2.5. Installing and upgrading on Linux: Ubuntu and Debian

These packages have been tested on Ubuntu 20.04, 22.04, 24.04 and Debian 11 and 12.

To install Radiator:

1. Download the distribution package for your operating system from Radiator downloads [https://downloads.radiatorsoftware.com/packages/]
2. Install the package. On Ubuntu 20.04, 22.04, 24.04 and Debian 11 (Bullseye) or 12 (Bookworm) the package is named as `radiator_4.xx-nn_all.deb`:

```
sudo apt install ./radiator_4.xx-nn_all.deb
```

3. Start Radiator and set it to start automatically after you reboot your Linux server.

```
sudo systemctl start radiator
sudo systemctl enable radiator
```

4. Test authentication. You should see OK printed to the screen 3 times. In case of a problem, check the Radiator process log, by default `/var/log/radiator/radiator.log`, for any errors during Radiator start.

```
/opt/radiator/radiator/radpwtst
```

5. Edit `/etc/radiator/radiator.conf` to suit your site and needs. Remember to restart Radiator after configuration change with **`systemctl restart radiator`**. This reference manual describes the options and parameters.

The package creates the following system user, system group and an empty home directory for Radiator service:

- **`radiator`**
- **`radiator`**
- `/var/lib/radiator`

By default, it creates the following directories for configuration, logs, Radiator itself, its utilities and other files. See `/opt/radiator/radiator/goodies/` for configuration samples:

- `/etc/radiator/`
- `/var/log/radiator/`
- `/opt/radiator/`
- `/usr/share/doc/radiator`

Default log rotate configuration is installed as

- `/etc/logrotate.d/radiator`

You can find documentation, additional dictionaries, and the goodies collection in `/opt/radiator/radiator/` directory.

Upgrading from a deb

Upgrades between deb packages do not require any special commands. Use 'apt install' or any other commands that you typically use to upgrade packages. Note: remember to restart radiator after each upgrade.

2.6. Installing and upgrading on Linux: generic RPM package

This RPM should only be used on legacy systems that do not have their specific packages. It is not recommended for new installations. Prerequisite is Perl 5.8.8 or better and Digest-MD5 version 2.02 or better, which are both installed by default in most recent Linux versions.

Important

In Radiator 4.21 and earlier, this was the only type of RPM package available. If you run a system that has a specific RPM available, such as CentOS 7, consider using it instead.

To install Radiator:

1. Log in as root.
2. Install the package. Make sure you have the correct package: the package name starts with `Radiator` and ends with `noarch.rpm`. The name starts with an upper case R and it must not have any distribution part, such as `el7`.

```
rpm -Uvh Radiator-x.yy-z.noarch.rpm
```

3. Start the server.

```
/etc/init.d/radiator start
```

4. Test authentication. You should see OK printed to the screen 3 times. In case of a problem, check the Radiator process log, by default `/var/log/radiator/radiator.log`, for any errors during Radiator start.

   ```
   radpwtst
   ```

5. Edit `/etc/radiator/radius.cfg` to suit your site and needs. This reference manual describes the options and parameters.

The RPM package will arrange for Radiator to start automatically each time you reboot your Linux host. Systemd users may want to see `goodies/radiator.service` for an alternative startup method. By default, it creates the following directories:

- `/etc/radiator/`
- `/usr/lib/perl5/site_perl/Radius/`
- `/var/log/radius/`
- `/usr/share/doc/packages/Radiator-x.yy/`

You can find documentation, additional dictionaries, and the goodies collection in `/usr/share/doc/packages/Radiator-x.yy`.

Upgrading generic RPM package

To upgrade, use RPM and restart radiusd:

```
rpm -Uvh Radiator-x.yy-z.noarch.rpm
/etc/init.d/radiator restart
```

Check Radiator log file after installation for any errors or other log messages that may require further action.

2.7. Installing and upgrading from repository

Follow the installation instructions from Radiator repository pages available in Radiator Linux package repositories [https://downloads.radiatorsoftware.com/repo/]

Upgrading from repository

Upgrades from repository do not require any special commands, use commands that you typically use to upgrade packages. Note: remember to restart radiator after each upgrade.

2.8. Installing and upgrading on Windows

Radiator works on Windows 7/8/8.1/10/11 and Server 2008/2012/2016/2019/2022. We recommend installing Radiator with a MSI package that bundles Radiator and Strawberry Perl.

An alternative option is to first install a Perl distribution and then install Radiator from a source code package. This is similar to using a process that is similar to full source distribution installation. For more information, see Section 2.9. Installing and upgrading with full source distribution on page 15.

If you'd like to use Radiator source code package, we recommend using Strawberry Perl. For installation packages and support, see their website Strawberry Perl website [http://strawberryperl.com]. Strawberry Perl is also quick and easy to install with a large number of optional modules.

Radiator comes with precompiled Perl modules required for *<AuthBy LSA>* authentication. *<AuthBy ACE>* and Digipass authentication require a separate binary module. If you require one of these authentication methods, it limits your choice of Perl version to install.

If you require RSA *<AuthBy ACE>* support, download `Authen::ACE4` from CPAN and compile it for your chosen Perl distribution. For more information, see Section 2.1.2. CPAN on page 3. In case you need help with `Authen::ACE4` setup, contact Radiator Software.

If you require Digipass module for your chosen Perl distribution, contact Radiator Software.

Radiator currently comes with precompiled modules for `Win32::LSA` that have been tested with Strawberry Perl. This module comes pre-installed with Radiator MSI packages.

2.8.1. Tips for Radiator on Windows

This section lists some useful notes for installing and using Radiator in Windows environment:

- Radiator can be installed as a Windows Service. See file:/data/radiator-reference-manual/source/HighAvailabilityradiusd/SystemServiceWindows_HighAvailability.dita#SystemServiceWindows_HighAvailability for more information.
- Perl on Windows automatically maps Unix style file names to DOS style (for example, changes `/` to `\`), so when you specify file names in the Radiator configuration file on Windows, you can use either Unix or Windows style. To avoid conflicts, choose one and use it consistently.

Notes for special configurations:

- Some ODBC drivers (notably Oracle) intercept the SIGINT handler, which makes it hard to kill **`radiusd`** with Control-C from within an Command Prompt window. To avoid this, create a shortcut to run **`radiusd`**, then you can always shut the window to kill **`radiusd`**.
- Some DBM file formats produced by AnyDBM_File on a PC and used by *<AuthBy DBM>* are not compatible with DBM files formats produced by Unix. If you create them with **`builddbm`** on one host, they may not be readable by Radiator on a different host. If in doubt, build the DBM file in the similar environment as the target host.
- **`radpwtst`** in **`-gui`** mode does not work properly on Windows, due to a bug in Tk.

2.8.2. Installing and upgrading on Windows with MSI package

Radiator MSI package includes Strawberry Perl for convenience. No separate installation of Perl is needed. However, if your environment already has a Perl installation, Strawberry Perl in the package does not disturb it. Supported Windows versions are Windows Server 2012 or newer, but older Windows Server versions can also be used provided they have at least PowerShell 3.0 installed.

To install Radiator, double click the MSI file, approve licence screens, and the installation is done. It is not possible to select where Radiator is installed, it will automatically install to the drive with most space available under `\Radiator\Radiator`. The package will arrange for Radiator service to start automatically as a Windows service each time you reboot your host.

To test your installation, use **`radpwtst`** to send authentication and accounting messages to Radiator:

1. Click "Radiator Software" -> "Radiator configuration" on Windows Start menu. This opens a Windows Explorer window that shows the contents of Radiator configuration and log directory under "Program Files" folder.
2. Double click "Perl command line" to open a Command Prompt window
3. Run **`radpwtst`** to send one authentication request and two accounting requests.

```
perl radpwtst
```

You see `OK` for all requests. The default configuration accepts all authentication attempts from the host Radiator runs on. Use Windows Explorer to open and view Radiator log files and configuration.

4. Optional step: If you send authentication requests from a remote host, these requests will be logged but rejected by the default configuration.

Radiator MSI package also supports silent operations, so it is possible to start the installation, upgrade, or uninstallation from command line without any UIs.

NOTICE

Radiator MSI package is not signed.

By default, the installer creates the following directories for configuration, logs, Radiator itself, its utilities and other files. See `\Radiator\Radiator\goodies\` for configuration samples:

`\Program Files\Radiator\`
: Configuration, logs and other possible files configuration requires

`\Radiator\Radiator\`
: Radiator configuration samples in `goodies\` directory, dictionary file and Radiator itself

`\Radiator\StrawberryPerl-Radiator\`
: Strawberry Perl distribution that comes with a Radiator MSI package

You can find documentation, additional dictionaries, and the goodies collection in `\Radiator\Radiator\` directory. For quick access to the above locations, *Radiator Software* is added to Start menu.

Upgrading and uninstalling the Windows MSI Package

Upgrading works similar way, double clicking the MSI file on the target machine will launch the upgrade functionality. Upgrade will not remove anything from `\Program Files\Radiator\`, and Radiator Windows Service will be available as it was after the upgrade.

Uninstalling can be done either from double clicking the MSI file used to install the software or from Control Panel - Add/Remove Programs. When uninstalling via MSI file, the only option available is Remove. Repair or Change is not available. All Radiator Software specific items are cleared out during the uninstall, except the `\Program Files\Radiator\` folder. This is to ensure your configurations are not lost.

Attention

Windows MSI package can only upgrade installations done with a MSI package.

2.8.3. Installing and upgrading on Windows using Strawberry Perl

To install Strawberry Perl and Radiator:

1. Download and install Strawberry Perl from StrawberryPerl website [http://strawberryperl.com/]

   During installation, we recommend installing it in `C:\Strawberry`. This is the default for the MSI package.

2. Connect your computer to the Internet so you are able download any required Perl modules from CPAN. For more information, see Section 2.1.2. CPAN on page 3.

3. Open a Command Prompt window. Install the prerequisite Perl modules.

```
cd \
cpan Win32::Daemon
cpan Digest::MD4
```

4. If you are going to use SQL authentication, find the database specific DBD module(s) from CPAN. For example, to install DBD::ODBC:

```
cpan DBD::ODBC
```

5. If you are going to use LDAP for authentication and accounting, obtain and install Net::LDAP:

```
cpan Net::LDAP
```

6. If you are going to use EAP TLS, TTLS or PEAP for 802.1x authentication obtain and install Net::SSLeay:

```
cpan Net::SSLeay
```

7. Download Radiator Zip file from Radiator downloads [https://radiatorsoftware.com/downloads/]. Unpack it to the default location, `C:\Radiator`. When using the default location, Radiator distribution is in `C:\Radiator\Radiator-x.yy` where x.yy is the version number.
8. Open a Command Prompt window with administrator access and move to the distribution directory.
9. Check that your distribution is complete:

```
perl Makefile.PL
```

10. Run the regression tests

```
perl test.pl
```

    You see lots of lines like `ok xx`, and none saying `not ok xx`.

11. Install Radiator:

```
perl Makefile.PL install
```

    This installs the Radiator programs and libraries in the standard places. It also creates a basic Radiator configuration file in `C:\Program Files\Radiator\radius.cfg` and a sample users file in `C:\Program Files\Radiator\users`.

12. Run Radiator to test the sample configuration:

```
perl c:\strawberry\perl\bin\radiusd
```

    You see some messages, followed by `NOTICE: Server started:`. Radiator is now waiting for RADIUS requests.

13. In another Command Prompt window run the test client program:

```
perl c:\strawberry\perl\bin\radpwtst -user mikem -password fred
```

    You see `OK` for all requests. This proves that Radiator has correctly authenticated the user mikem, whose login details are in the users file in `C:\Program Files\Radiator\users`.

14. Rerun radpwtst, this time with the wrong password for mikem:

```
perl c:\strawberry\perl\bin\radpwtst -user mikem -password wrong
```

You see `Rejected:` for Access-Request.

15. If you configure a test NAS to use this server, you are to log in as the user mikem with password fred.

Upgrading Radiator

To upgrade Radiator, repeat the installation using the a Radiator distribution package file. The files in the new distribution will overwrite any files in the old distribution

2.8.4. Troubleshooting Windows service

Windows service does not start

When Windows service does not start at all, the problem can be caused by a missing Perl module. The exact error can be usually found from the Radiator log file, search a line with ERR information or line specifying missing Perl module, for example `Can't locate Win32/NetAdmin.pm in @INC (you may need to install the Win32::NetAdmin module)`.

Windows service does not stay up

When Windows service does start but stops after a while, the problem might also be a missing Perl module. In this case the module is loaded only when it is tried to be used instead of loading the module during startup. Radiator log file can contain information why the service stopped, but this situation can also be troubleshooted with command line.

- Stop Radiator Windows service if it is running.
- Start Radiator manually from command line window. Use the shortcut for elevated command line window `\Program Files\Radiator\Perl command line Elevated` to get Perl environment and have Radiator log file permissions. Radiator log file permissions are needed because debug information is both logged to file and printed to command line window.
- To start *`radiusd`* type for example: **`perl C:\Radiator\Radiator\radiusd -log_stdout -foreground -trace 4 -config_file "C:\Program Files\Radiator\radiator.conf"`** The default configuration from the command can be changed to suit the troubleshooting needs.

This will start *`radiusd`* and print all the debug information to the command line window and to Radiator log file. Once the request that caused Radiator Windows service to stop is sent to the server again, it is likely that the command line window will show the exact problem.

SE_TCB_PRIVILEGE errors when using command line

When Radiator is run from command line in Windows and *`<AuthBy LSA>`* is used, following error might appear: `ERR: Could not AdjustPrivilege SE_TCB_PRIVILEGE: A required privilege is not held by the client.` on the command line. This means that Radiator must be run on Windows as a user that has the ‘Act as part of the operating system's security policy' (SE_TCB_PRIVILEGE) enabled. The 'Local System' account that the Radiator Windows service runs as has this privilege by default. If all the other errors have been cleared by troubleshooting with command line and only this error is left, Radiator Windows service will work.

2.9. Installing and upgrading with full source distribution

Also known as the tar or zip package, this is the general and widely portable procedure for installing Radiator. However, a number of platform-specific installation methods are also available.

This installation method requires `ExtUtils::MakeMaker`, which is part of Perl. Some recent Linux distributions, such as Red Hat Enterprise Linux, may not have `ExtUtils::MakeMaker` installed by default. Install it with the following command:

```
yum install perl-ExtUtils-MakeMaker
```

The minimum prerequisite is Perl `Digest::SHA` module, which is part of core Perl since Perl 5.10.0. For example, on Red Hat Enterprise Linux, it is packaged separately as `perl-Digest-SHA` and can be installed with yum. For more information, see Section 2.1.2. CPAN on page 3.

The Radiator full source distribution is supplied as a gzipped, tarred distribution file. The standard distribution file name is `Radiator-x.yy.tgz`, where "`x.yy`" is the revision number. Radiator zip distribution is named similarly with file name ending as zip instead of tgz. Save the distribution archive somewhere suitable, such as `/usr/local/src/`, and unpack it as follows:

```
zcat Radiator-x.yy.tgz | tar xvf -
```

In this case, zcat is the GNU zcat command. If your path does not include the GNU zcat:

```
cat Radiator-x.yy.tgz | gunzip -c| tar xvf -
```

To unpack a zip distribution, use a GUI tool or command line:

```
unzip Radiator-x.yy.zip
```

In either case, this creates a directory `Radiator-x.yy` into the current directory:

```
cd Radiator-x.yy
perl Makefile.PL
make test
```

This runs a fairly exhaustive test suite on your RADIUS server. It can take few minutes.

```
make install
```

This optional command installs the RADIUS modules that Radiator requires in your site-Perl directory (for example `/usr/local/lib/perl5/site_perl/`). It installs the RADIUS daemon (*`radiusd`*, the command line password test program `radpwtst`, the DBM file builder `builddbm`, and the SQL database builder `buildsql` in your default directory for local executables, which is typically `/usr/local/bin/`.

You can omit **`make install`** command and run your Radiator directly in the distribution directory.

Upgrading

To upgrade Radiator with `.zip` or `.tgz` package, repeat the installation using the new distribution package file. The files in the new distribution will overwrite any files in the old distribution.

2.10. Installing and upgrading on Solaris

On Solaris, we recommend install and upgrade from the tar or zip distribution as described above. Previous Radiator versions were packaged for Solaris, but required Perl from Sunfreeware [http://www.sunfreeware.com] to find the correct installation locations. We now recommend to install Perl from your preferred source and then use the Radiator tar package for Radiator installation.

2.11. Installing and upgrading on macOS

On macOS, we recommend install and upgrade from the tar or zip distribution as described above. We recommend installing Xcode and Perl installation management tool, such as Perlbrew, which can be used to install, for example, *`cpanm`* to obtain modules from CPAN. For more information, see Section 2.1.2. CPAN on page 3. This allows you to start working with Radiator without modifying the Perl installation that comes with macOS.

Perl that comes with macOS works with Radiator, but may not have all the modules your configuration might require.

2.12. Installing, upgrading and managing with Ansible

To help manage and install Radiator with a more automatic way, Radiator Software Ansible playbooks are available in `goodies` directory. On RPM and deb based installations see `/opt/radiator/radiator/goodies/Ansible`. A file named `README.md` in this directory contains the latest information about the available playbooks. These playbooks make it easy to:

- **Install** Radiator and its basic prerequisites to multiple servers with a single command
- **Upgrade** or **downgrade** Radiator on multiple servers with a single command
- **Deploy** with a single command:
  - Multiple Radiator configurations to multiple servers
  - One Radiator configuration to a single server running multiple Radiator instances
- **Rollback** the latest Radiator configuration deployment quickly to the previous Radiator configuration
- **Restart**, **start** or **stop** all Radiator instances on multiple servers with a single command

The playbooks require Ansible 2.7 or later on the Ansible control node. Supported Linux distributions are Ubuntu 18.04, Debian 10, Red Hat Enterprise Linux 7 and compatible systems such as CentOS, Oracle Linux, AlmaLinux and Rocky Linux. All newer versions of these distributions are also supported.

2.13. Docker containers for Radiator

Radiator Software Dockerfiles for building Docker containers with Radiator are available in `goodies` directory. On RPM and deb based installations see `/opt/radiator/radiator/goodies/Docker`. A file named `README.md` in this directory contains the latest information about the available Dockerfiles. Currently available installation source and target variations are:

- From Radiator **public repository** to an Alma Linux 8 and 9 container
- From Radiator **public repository** to a Ubuntu 20.04 and 22.04 container
- From Radiator **local RPM packages** to an Alma Linux 8 and 9 container
- From Radiator **local deb packages** to a Ubuntu 20.04 and 22.04 container
- From Radiator **local MSI package** to a Windows Server Core 2019 and 2022 container

Each Dockerfile has a command ready, but commented out, for copying your own Radiator configuration into the container image during the build phase. Linux based Dockerfiles use `ENTRYPOINT` for running Radiator. Windows containers run Radiator as a Windows service. See the `README.md` file for tips and ideas on how to customise the Dockerfiles for your own requirements.

2.14. Troubleshooting

If you have trouble installing or running Radiator, see the Radiator log file for hints. For more information about getting help, see Section 21. Getting help on page 532.

3. Configuring Radiator

This section describes the Radiator configuration file and the statements that control the behaviour of the Radiator server, *`radiusd`*.

When *radiusd* starts, it reads the configuration file. The default file name and path for the configuration file depends on the operating system and how Radiator is installed.

- Unix and macOS: `/etc/radiator/radius.cfg` or `radiator.conf`
- Windows: `C:\Program Files\Radiator\radius.cfg` or `radiator.conf`

Radiator deb, RPM and Windows MSI packages use startup configuration that uses `radiator.conf` to be consistent with log directory, package name and other names.

You can specify an alternate configuration file with radiusd's *-config_file* command line parameter. There is a test configuration file (`radius.cfg`) in the Radiator distribution that shows most of parameters and clauses that you can use in a configuration file, and examples of how to use them. There is also a very simple example (`simple.cfg`) in the `goodies/` directory in the Radiator distribution. It is a good starting point for your own configuration file.

In general terms, the configuration file allows you control the following things:

- Behaviour of the server in general, including logging
- Which RADIUS clients the server responds to
- Which RADIUS realms the server works with
- Handlers for specifying special ways to handle requests
- For each realm, what methods are used for authenticating users and storing accounting information
- For each authentication method for each realm, the configuration of the authentication module
- Custom Perl hooks that are run during the processing of each request. For more information, see Section 14. Execution sequence and hook processing on page 511

The configuration file is an ASCII text file, it can be edited by any text editor. Leading white space in each line is ignored, so you can use indentation to make your configuration file easier to read. Case is important in all parameter names and clauses.

Tip

An alternative to editing the configuration file directly is to use the ServerHTTP clause (for more information, see Section 3.122. <ServerHTTP> on page 424 and Section 10. Configuring Radiator with GUI on page 493) which allows you to connect to Radiator with standard web browser and examine, change and test the configuration with an easy to use point and click web interface.

The configuration file consist of the following things:

- Comment line

  The comment line starts with a # as the first character. Anything including and after the # is ignored. Blank lines are also ignored. Here is an example:

```
# This is a comment
```

CAUTION

Comments that do not start at the beginning of the line are likely to be taken as part of a parameter value:

```
SomeParameter xxxxxx # INCORRECT:this is not a comment
```

- Parameter setting

  The first word is the name of the parameter to set, all the following words and digits are the value to be used for the parameter. All the parameters you can set in the configuration file are described in this document. Here is an example:

  ```
  Trace 4
  ```

- Parameter setting from file

  Parameter values can be retrieved from an external file. The line consist of the parameter name and filename, here is the format:

  ```
  parameter file:"filename"
  ```

  This is a useful feature for putting long parameters, such as a hook, in an external file. Here is an example that loads the code for *`PreAuthHook`* from the external file `hook.pl`:

  ```
  PreAuthHook file:"hook.pl"
  ```

- Parameter setting from SQL database query.

  Parameter values can be retrieved from an SQL database. The line consist of the parameter name, SQL identifier, and SQL query, here is the format:

  ```
  ParameterName sql:identifier:query
  ```

  It looks for a previously defined *`<AuthBy SQL>`* clause with an Identifier and runs the given SQL query. The first row in the result is used to set the parameter. The SQL database lookup is only done once at startup time. Here is an example of using the SQL query:

  ```
  <AuthBy SQL>
        Identifier        myidentifier
        DBSource          dbi:mysql:radius
        DBUsername        mikem
        DBAuth            fred
  </AuthBy>
  Trace sql:myidentifier:select value from configuration where\
        name = 'Trace'
  ```

- Start or end of clause

  A clause is a collection of parameter settings related to a single feature in the server. The first line in a clause is surrounded by angle brackets, for example *`<Client fred>`*, which marks the beginning of the configuration for client with the DNS name "fred". Subsequent lines are interpreted as parameter settings for the feature, until the end of the clause is seen. The end of the clause is surrounded by angle brackets with a slash. Here is a usage example:

  ```
  <Client DEFAULT>
        # Configuration parameters for the Client go here
        .....
  </Client>
  ```

3.1. Tips for using Radiator configuration files

This section discusses some practical tips for using Radiator configuration files.

Security

The configuration file usually contains the shared secrets that allow your RADIUS clients to communicate with the Radiator RADIUS server. It can also contain passwords for access to databases and such. This means that for security reasons, keep the configuration file as secure as possible. On Unix, make sure that it is readable only by the user that *radiusd* runs as.

Enabling and disabling flags

Parameters that are on/off flags, such as *LogSuccess* or *LogFailure*, can be enabled or disabled in a number of ways. Values of **0**, **no**, or **false** (case insensitive) turn the flag off, whereas any other value, including the empty string, turn the flag on.

These turn the flag parameter off:

```
IgnoreAcctSignature 0
IgnoreAcctSignature no
IgnoreAcctSignature NO
IgnoreAcctSignature false
IgnoreAcctSignature FALSE
```

These turn the flag parameter on:

```
IgnoreAcctSignature
IgnoreAcctSignature 1
IgnoreAcctSignature yes
IgnoreAcctSignature anythingatall
DefineGlobalVar myspecialflag yes
IgnoreAcctSignature %{GlobalVar:myspecialflag}
```

Splitting lines

Long lines in your configuration file can be split over multiple lines by using the \ character at the end of each line except the last:

```
AuthSelect select s.password, g.session_timeout \
      s.check_items s.reply_items \
      from subscribers s, groups g \
      where username=? and s.group \
      = g.name
```

Escaped octal characters

Parameter values can contain escaped octal characters. Here is an example how to specify an *AcctLogFileFormat* with newline (octal 012) separated lines:

```
AcctLogFileFormat %{Timestamp}\012%{Acct-Session-Id}\
      \012%{User-Name}\012
```

Clause order

The order of clauses in the Radiator configuration file is significant. All the clauses are parsed and internal data structures constructed during the initial parse of the configuration file. They are constructed in the order they

appear in the configuration file. For example, if a <Log xxxxxx> clause is encountered, that logger is created immediately and used to log all subsequent parsing and startup errors. This means that if a <Log xxxxxx> clause is encountered in the configuration file, only errors in clauses that appear after the Log clause is logged using that method.

3.2. Including parts of configuration from external files

The include directive in the Radiator configuration file is followed by a file name. That external file is opened and read to the end as a configuration file before processing of the current configuration file continues. Special filename characters are permitted (for more information, see Section 3.3. Special formatters on page 21). Files can be recursively included to any depth. The include keyword is case insensitive, here are examples, which are both correct:

```
include %D/clients.cfg
Include %D/realms.cfg
```

Filename can include csh(1) style wildcards and expansions such as *, ?, [...], {....}, ~, and so on. Files whose first character is a '.' are ignored unless explicitly matched. Wildcard files are included in lexicographic order of their filename. Here are examples of using wildcards:

```
# Reads all files with a .cfg extension:
include %D/*.cfg
# Matches radiator1.cfg, radiator2.cfg etc:
include /etc/radiator/radiator[0123456].cfg
```

You can also use the include mechanism to capture the output of a script. A program name followed by a vertical bar runs the named program and includes its output into the configuration file. This is a useful way for generating some or all of your Radiator configuration programmatically. For example, you can write a script to generate all your <Client> clauses by processing some external description of all your NASs. Here is an example of running a program:

```
include %D/myClientScript.pl|
```

3.3. Special formatters

Wherever you can specify a file name in the Radiator configuration file, you can use some special formatters in the path name. These special formatters can also be used in a number of other configuration file parameters. These special formatters will be replaced at startup or run time, so you can dynamically change file paths and the like so they depend on such things as the date, realm, user name, and so on.

Special formatters are introduced by a %, followed by a single character or a format name in curly brackets.

The following formatters are available:

Table 1. Special characters, which are replaced from the current time according to the Radiator host

Specifier	Replacement
%l	Current time in long format, for example, Thu Jul 1 08:38:21 1999
%B	Current time in common SQL date time format, for example, Sep 12, 2003 15:48
%G	Current time in extended SQL date format including seconds, for example, Sep 12, 2003 15:48:59
%t	Current time in seconds since Jan 1, 1970
%S	Current second (00-59)

Specifier	Replacement
%M	Current minute (00-59)
%H	Current hour (00-23)
%d	Current day of the month (2 digits)
%m	Current month number (2 digits, 01-12)
%Y	Current year (4 digits)
%y	Last 2 digits of the current year (2 digits)
%q	Day of the week, abbreviated (for example, Sun, Mon, Tue)
%Q	Day of the week (for example, Sunday, Monday, Tuesday)
%v	Month of the year, abbreviated (for example, Jan, Feb, Mar)
%V	Month of the year (for example, January, February, March)
%s	Microseconds in the current second

Note

Timestamp is only available with Accounting-Request messages. For more about Timestamp, see Section 3.41.11. AcctColumnDef on page 196

*Table 2. Special characters, which are replaced from the Timestamp of the current **Accounting-Request** packet*

Specifier	Replacement
%o	Timestamp in long format, for example, Thu Jul 1 08:38:21 1999
%A	Timestamp in common SQL date time format, for example, Sep 12, 2003 15:48
%J	Timestamp in another common SQL date time format, for example, 2003-09-12 15:48:00
%F	Timestamp in extended SQL date format including seconds, for example, Sep 12, 2003 15:48:59
%b	Timestamp in seconds since Jan 1 1970
%p	Timestamp second (0-59)
%k	Timestamp minute (0-59)
%j	Timestamp hour (0-23)
%i	Timestamp day of the month (2 digits)
%g	Timestamp month number (2 digits)
%f	Timestamp year (4 digits)
%e	Last 2 digits of the Timestamp year (2 digits)
%E	The elapsed time in seconds since the packet was received. Can be used, for example, to log processing time for proxied packets.

Table 3. Special characters, which are replaced with other information of the current request

Specifier	Replacement
%c	IP address of the client who sent the current request, if any
%C	Client name of the client who sent the current request, if any **Note** This does a reverse name lookup on the address and depending on your environment, this may take a number of seconds to resolve
%R	The realm of the user name in the current request, if any, after any `RewriteUsername` is applied. This is everything following the first @ sign in the `User-Name`
%K	The trailing realm of the user name named in the current request, if any, after any `RewriteUsername` is applied. This is everything following the last @ sign in the `User-Name`
%N	First available: `NAS-IP-Address`, `NAS-IPv6-Address`, `NAS-Identifier` or sender IPv4 address
%n	Full `User-Name`, including the realm, currently being authenticated, after any `RewriteUsername` is applied
%U	`User-Name` currently being authenticated with the realm, if any, stripped off, after any `RewriteUsername` is applied
%u	Full original `User-Name` that was received, before any `RewriteUsername` is applied
%w	User name part of the full original user name before any `RewriteUsername` rules were applied
%W	Realm part of the full original user name before any `RewriteUsername` rules were applied
%P	Decrypted `User-Password` from the current request
%T	Request type of the current request, if any. This may be, for example, `Access-Request` or `Accounting-Request`
%z	`User-Name` in the current packet, hashed with MD5
%I	NAS identifier as an integer instead of dotted decimal character string, useful for speeding up SQL queries
%X	EAP identity of the EAP request, with any trailing `@realm` stripped off
%x	EAP identity of the EAP request
%Z	The RADIUS Identifier of the incoming request
%{attr}	The value of the named attribute in the current packet (if any). For example, %{User-Name} is the same as %n

Table 4. Miscellaneous special characters and format names

Specifier	Replacement
%%	Percent character

Specifier	Replacement
%r	Literal newline character
%D	Value of *DbDir* as configured in your Radiator configuration file
%L	Value of *LogDir* as configured in your Radiator configuration file
%h	Hostname this server is running on
%O	The server instance number, when FarmSize is used to specify a server farm. 0 is the main (supervising) server.
%{Special:X}	Same as *%X*, where X is any of the single special characters listed above. For example, *%{Special:a}* will produce the same result as just *%a*
%{GlobalVar:name}	The value of the global variable called name. Global variables can be set with **name=value** on the command line, or with "*DefineFormattedGlobalVar* name value" in the configuration file. If the variable "name" has not been defined, replaced with an empty string.
%a	Framed-IP-Address in the reply message being created, if any
%{Request:name}	Value of the named attribute in the current request, if any. This is the same as just *%{name}*, but may be used instead for clarity.
%{OuterRequest:name}	Value of the named attribute in the outer request, of the current request, if any. May be used where the request has been tunnelled using PEAP or TTLS.
%{Reply:name}	Value of the named attribute in the reply currently being created, if any. For example, *%{Reply:Framed-IP-Address}* is the same as *%a*. If there is no current reply, or the attribute is not present in the reply, replaced with an empty string
%{RequestAttrs:name}	All values of the named attribute in the current request, separated by commas
%{OuterRequestAttrs:name}	All values of the named attribute in the outer request of the current request, separated by commas
%{ReplyAttrs:name}	All values of the named attribute in the current reply, separated by commas
%{Client:name}	Value of the named parameter from the Client clause that accepted the current packet, if any.
%{Handler:name}	Value of the named parameter from the Handler clause that is handling the current packet, if any.
%{AuthBy:name}	Value of the named parameter from the AuthBy clause that is handling the current packet, if any.
%{Server:name}	Value of the named parameter from the global server configuration, for example, *%{Server:Trace}* is replaced by the current value of the global Trace parameter.
%{RequestVar:name}	Value of the current request object similar to Client, Handler, and AuthBy specials above.
%{ReplyVar:name}	Value of the current reply object similar to Client, Handler, and AuthBy specials above.

Specifier	Replacement
%{IntegerVal:name}	Value of the named attribute in the current packet, if any, expressed as an integer, instead of as a value name from the dictionary, for example, *%{IntegerVal:Tunnel-Type}* is replaced by **3** if the Tunnel-Type is **L2TP**.
%{TimestampVal:number}	Value of the current Unix time stamp + the number. Number can be a positive or negative integer, request attribute name ,or a special character. For example, *%{TimestampVal:3000}*, *%{TimestampVal:Session-Timeout}* or *%{TimestampVal:%{Reply:Session-Timeout}}*. This is useful for replacing hooks with formatters for calculating time stamps.
%{HexAddress:name}	Replaced by the named IPv4 attribute in the current packet, if any, expressed as a hexadecimal string. For example, *%{HexAddress: NAS-IP-Address}* is replaced by **CB3F9A01** if the NAS-IP-Address in the current request is **203.63.154.1**.
%{Quote:somestring}	When used with SQL modules, replaced by *somestring* quoted with the appropriate quoting style for the SQL database in use. For example, when used with a mysql database, *%{Quote:somestring}* is replaced by **somestring**.
%{LDAPDN:somestring}	Replaced by *somestring* escaped with LDAP DN rules. Requires Perl Net::LDAP module.
%{LDAPFilter:somestring}	Replaced by *somestring* escaped with LDAP filter rules. Requires Perl Net::LDAP module.
%{SQL:identifier:query}	Replaced with a value fetched from an SQL database. Looks for a previously defined AuthBy SQL clause with the Identifier of *identifier* and runs the SQL query given by *query*. The first row in the result will be used as the value of the special character. This type of lookup is done whenever the special character is evaluated.
%{EAPTLS:name}	Value of named TLS session parameter for the current TLS-based EAP authentication. The valid parameter names are: *Protocol*, *Cipher*, *Session_ID*, *Start_Time*, and *Timeout*. For more information, see OpenSSL sess_id [https://www.openssl.org/docs/manmaster/man1/openssl-sess_id.html].
%{URIEncode:somestring}	Replaced by *somestring* escaped with RFC 3986 percent-encoding rules. Requires Perl URI::Escape module.
%{URIEncodeUTF8:somestring}	Replaced by *somestring* escaped with RFC 3986 percent-encoding rules. Encodes *somestring* as UTF-8 before percent-encoding. Requires Perl URI::Escape module.
%0 - %99	Depending on the context, these may be replaced with context-specific values, which are documented in this reference manual.

Note that some of these specifiers are only valid when a RADIUS message is being processed. In any other context, such a specifier will be replaced by an empty string.

In the following example, the log file will be stored in LogDir, with a name that starts with the current year. If LogDir is `/var/log` and the current year was 1998, this would result in a log file name of `/var/log/1998-logfile`.

```
LogFile %L/%Y-logfile
```

Special characters of the form **%{x:y}** may be nested and contain other **%{a:b}** or %h special forms, such as **%{x:%{y:z}}**, or **%{x:%h}** This can be useful in an example, where the resulting Host parameter will be *desthostname.com*:

```
DefineFormattedGlobalVar hostname myhostname
DefineFormattedGlobalVar role myrolename
DefineFormattedGlobalVar myhostname_myrolename desthostname.com
.....
Host %{GlobalVar:%{GlobalVar:hostname}_%{GlobalVar:role}}
```

3.4. Date formatting

When Radiator is required to format a date for use with SQL and other databases it uses a configurable DateFormat on page 196. A DateFormat can include special characters which are interpreted according to the following table. Any other characters will be used verbatim.

Table 5. DateFormat special characters

Specifier	Replacement
%%	Percent character
%a	Day of the week, abbreviated
%A	Day of the week
%b	Month, of the year, abbreviated
%B	Month of the year
%c	Ctime format, for example, Sat Nov 19 21:05:57 1994
%d	Current day of the month (2 digits)
%e	Numeric day of the month, no leading 0
%D	MM/DD/YY
%h	Month of year, abbreviated
%H	Current hour (00-23)
%I	Hour, 12-hour clock, leading 0
%j	Day of the year
%k	Hour
%l	Hour, 12-hour clock
%m	Current month number (2 digits, 01-12)
%M	Current minute (00-59)
%n	NEWLINE character
%o	Ornate day of month, for example, 1st, 2nd, 25th
%p	AM or PM
%r	Time format: 09:05:57 PM

Specifier	Replacement
%R	Time format: 21:05
%S	Current second (00-59)
%t	TAB character
%T	Time format: 21:05:57
%U	Week number, Sunday as first day of week
%w	Day of the week, numerically, Sunday = 0
%W	Week number, Monday as first day of week
%x	Date format: 11/19/94
%X	Time format: 21:05:57
%y	Last 2 digits of the current year (2 digits)
%Y	Current year (4 digits)
%Z	Timezone in ASCII. eg: PST

For example, DateFormat `%b %e, %Y %H:%M` would format a date and time like: `Sep 3, 1995 13:37`.

3.5. Address binding

One of the main functions of Radiator is to listen for UDP packets and TCP connections from other systems according to the Radiator configuration. The various Radiator clauses that can accept packets or connections from other systems all support the BindAddress parameter, which controls which IP addresses Radiator will listen on. IP packets sent to an IP address which is on the Radiator host, but which Radiator has not bound with BindAddress will not be received by Radiator.

The driver for this is that a single host may have multiple IP addresses, and those addresses may be IPv4, IPv6 and/or IPv6-over-IPv4. You may require Radiator to only honour requests directed to one of or a subset of the IP addresses for the host.

With BindAddress you can control which destination IP addresses Radiator will accept. You can specify one or more IPv4 or IPv6 addresses, including wildcard addresses, with the BindAddress parameter. The following forms may be used:

- 0.0.0.0 (the default) Any IPv4 address on the host
- 1.2.3.4 A specific IPv4 address on the host
- :: Any IPv6 address on the host (and this may include any IPv4 address too, depending on how the host is configured)
- 2001:db8:148:100::31 A specific IPv6 address on the host

They may be combined in one BindAddress parameter like so:

```
BindAddress 0.0.0.0
BindAddress 192.87.30.31,2001:db8:148:100::31
BindAddress ::, 0.0.0.0
```

For more information about IPv6 support and IPv6 wildcard addresses, see Section 3.6. IPv6 support on page 28.

CAUTION

Wildcard addresses may require special attention. When a host is multihomed or uses alias IP addresses, the Radius replies from that host typically use the main address of the outgoing interface. In other words, requests sent to a certain address may be replied from a different address. You can work around this by configuring the addresses individually with BindAddress instead of using the wildcard address.

3.6. IPv6 support

Radiator supports IPv6 transport for RADIUS over UDP, RadSec (RFC 6614), DIAMETER, and TACACS+. This means that Radiator can send, receive, and proxy requests over IPv6.

Note

Older Perl versions require Socket6.pm for IPv6 support. If Radiator cannot find working IPv6 support during the startup, it logs an INFO level message. If there is no working IPv6 support, RADIUS attributes with IPv6 types are available as binary values, and IPv6 sockets are not supported.

Note

Radiator 4.13 does not require ipv6:-prefix for IPv6 addresses anymore. It is still allowed but should be avoided in new configurations. This prefix can be used to request IPv6 address resolution for names. By default, the names are currently resolved to IPv4 addresses only.

Various modules describe their IPv6 support in more detail. For example, `<AuthBy LDAP2>` can connect to LDAP servers over IPv6 with detailed IPv6 information. For more information, see Section 3.47. <AuthBy LDAP2> on page 226. Another example is the Resolver clause for `<AuthBy DNSROAM>`, which can query DNS servers over IPv6 and optionally fetch IPv6 records if configured to do so.

For more information about SNMP support for IPv6, see Section 3.29. <SNMPAgent> on page 145.

Supported RADIUS specific IPv6 RFCs include:

- RFC 3162 which defines Framed-IPv6-Prefix and such
- RFC 4818 which defines Delegated-IPv6-Prefix
- RFCs 4669 and 4671 which define IPv6 RADIUS server MIBs
- RFC 6911 which defines Framed-IPv6-Address and uch
- RFC 6930 which defines the 6rd IPv6 deployment mechanism

Many protocols, such as DIAMETER, do not require specifically IPv4 or IPv6. Radiator supports these protocols over both IP versions as long as the underlying operating system and Perl modules offer support for the both protocols.

3.6.1. IPv6 Clients

IPv6 Client clauses support single IPv6 addresses and IPv6 CIDR notation. When IPv6 CIDR client clauses are used, you may want to consider installing Math::BigInt::GMP or Math::BigInt::Pari Perl modules for more efficient IPv6 network mask calculation.

3.6.2. IPv6 wildcard listen address

If you are using recent Perl or Socket.pm you can configure separate IPv4 and IPv6 wildcard listen sockets by specifying both IPv6 and IPv4 wildcard addresses and turning on IPV6_V6ONLY socket option:

```
BindAddress ::,0.0.0.0
BindV6Only
```

For more information about this parameter, see Section 3.7.10. BindV6Only on page 33.

Note

There are differences between operating systems. For example, on Linux IPV6_V6ONLY is off by default. When it is off, the following will BindAddress parameter not work as expected. The bind to IPv6 wildcard address is done first and this does not allow binding to IPv4 wildcard address afterwards:

```
BindAddress ::, 0.0.0.0
```

If IPV6_V6ONLY is set to 1, the IPv6 bind will not affect the IPv4 bind and the example BindAddress will work as expected. On OS X 10.9 it is possible to bind to IPv4 wildcard address first followed by IPv6 wildcard address. On Windows IPV6_V6ONLY behaviour is the default.

Note

Linux has a sysctl kernel parameter net.ipv6.bindv6only and special file to control the system wide behaviour: `/proc/sys/net/ipv6/bindv6only`.

3.6.3. IPv4-mapped IPv6 address

When IPv4 packets are received by a system that uses IPv6 wildcard listen sockets, the client addresses may show as IPv4-mapped IPv6 addresses. See Section 3.7.10. BindV6Only on page 33 for more info about this feature and how to control IPv4 mapped IPv6 addresses.

Tip

Configuring the operating system or Radiator to not use IPv4-mapped IPv6 addresses allows you to keep IPv4 and IPv6 clearly separate.

3.7. Global parameters

These parameters apply to the server as a whole, and do not appear inside a clause. They are used to control the behaviour of the server as a whole.

3.7.1. Foreground

If this parameter is set, it makes the server run in the foreground instead of as a detached daemon. No argument is required. The default behaviour is to run as a daemon. You must run in the foreground if you want to run Radiator in a console window, from inetd or restartWrapper. For more information, see file:/data/radiator-reference-manual/source/HighAvailabilityradiusd/UsingInetd_HighAvailability.dita#UsingInit_HighAvailability and file:/data/radiator-reference-manual/source/HighAvailabilityradiusd/UsingRestartWrapper_HighAvailability.dita#UsingRestartWrapper_HighAvailamility.

```
# Run in the foreground
Foreground
```

3.7.2. LogStdout

If this parameter appears, it makes all logging output appear on STDOUT as well as in the log file. No argument is required. The default behaviour is not to log to STDOUT. You must be running in Foreground mode for this to have an effect.

```
# Log to stdout
LogStdout
```

3.7.3. Trace

Sets the priority level of trace messages to be logged in the log file (and printed on stdout if LogStdout is defined). The argument is an integer from 0 to 5, with the following meanings:

- 0 ERR. Error conditions. Serious and unexpected failures
- 1 WARNING. Warning conditions. Unexpected failures
- 2 NOTICE. Normal but significant conditions
- 3 INFO. Informational messages
- 4 DEBUG. Debugging messages
- 5 EXTRA_DEBUG. Incoming raw packet dumps in hexadecimal

A trace level of 4 or more will produce all the possible tracing messages, including dumps of every RADIUS message received and sent: you probably do not want that in a production environment. The default tracing level is 0. You can change the current tracing level while Radiator is running on Unix platforms by signalling it with SIGUSR1 and SIGUSR2. For more information, see Section 4. Running radiusd on page 441.

```
# Show everything up to and including INFO level
Trace       3
```

3.7.4. LogTraceId

LogTraceId flag parameter allows logging messages related to an authentication exchange and to its subsequent accounting session with a tracing identifier. *LogTraceId* can be configured for global level and Log clause level. *LogTraceId* enables prepending a tracing ID to messages logged to STDOUT, when *LogStdout* is enabled, and to log file configured with <*Log FILE*> and <*Log SYSLOG*>. For more information, see Section 3.7.2. LogStdout on page 30.

Tip

Support for using the same tracing identifier with accounting messages requires enabling parameter in the authenticating Handler clause. For more information, see Section 3.31.38. AutoClass on page 162.

Tracing ID works in conjunction with the Radiator load balancer allowing coordinated log message indexing and lookup between frontend load balancers and backend workers.

```
# Prepend tracing id to log messages
LogTraceId
```

3.7.5. LogRejectLevel

Log level for rejected authentication attempts. Can be overridden by Handler. Defaults to 3 (INFO).

3.7.6. AuthPort

Specifies which port(s) Radiator will listen on for RADIUS authentication requests. The argument may be either a numeric port number or an alphanumeric service name as specified in `/etc/services/` (or its equivalent on your system). Multiple comma-separated ports may be specified. The default port is 1645. Note that the officially assigned port number for RADIUS authentication has been changed to 1812. AuthPort may contain special formatting characters. A typical use of special formatting characters is with GlobalVar and command line arguments.

```
# Listen for authentication requests on port 1812 as per RFC
# 2865
AuthPort 1812
```

Note

Actually Radiator will also service accounting requests received on the authentication port without complaint.

Tip

You can prevent Radiator from binding to an authentication port by undefining AuthPort:

```
# Do not bind to an auth port:
AuthPort
```

Tip

You can pass any port number as a command line argument to radiusd with a configuration like this:

```
AuthPort %{GlobalVar:authport}
```

and then run radiusd with an argument to set the port number like this:

```
radiusd authport=1810 ...
```

Tip

You could listen for requests on the 2 most common RADIUS authentication port numbers with

```
AuthPort 1645,1812
```

Tip

Radiator will listen for requests on each given AuthPort (default 1645), on each IP address specified by BindAddress (default 0.0.0.0). For more information, see Section 3.5. Address binding on page 27.

3.7.7. AcctPort

Specifies which port Radiator will listen on for RADIUS accounting requests. The argument may be either a numeric port number or an alphanumeric service name as specified in `/etc/services` (or equivalent on your system). Multiple comma-separated ports may be specified. The default port is 1646. Note that the officially assigned port number for RADIUS accounting has been changed to 1813. AcctPort may contain special formatting characters. A typical use of special formatting characters is with GlobalVar and command line arguments.

```
# Listen for accounting requests on port 1813 as
# per RFC 2866
AcctPort 1813
```

Note

Actually Radiator will also service authentication requests received on the accounting port without complaint.

Tip

You can prevent Radiator from binding to an accounting port by undefining Acct- Port:

```
# Do not bind to an accounting port:
AcctPort
```

Tip

You could listen for requests on the 2 most common RADIUS accounting port numbers with

```
AcctPort 1646,1813
```

Tip

Radiator will listen for requests on each given AcctPort (default 1646), on each IP address specified by BindAddress (default 0.0.0.0). For more information, see Section 3.5. Address binding on page 27.

3.7.8. KeepSocketsOnReload

Note

This option was introduced in Radiator 4.13 and is currently considered experimental.

This optional flag controls whether opened RADIUS listen sockets should be left intact on a reload request. When enabled, the changes in BindAddress, AuthPort and AcctPort are ignored during reload. You may consider enabling this option when incoming RADIUS requests should be buffered during the reload instead of ICMP unreachable messages being sent back to the RADIUS clients.

3.7.9. BindAddress

This optional parameter specifies one or more IPv4 and IPv6 addresses to listen for RADIUS requests on. It is only useful if you are running Radiator on a multi-homed host (i.e. a host that has more than one network address). Defaults to 0.0.0.0 (i.e. listens on all IPv4 networks connected to the host). Multiple addresses can be separated with a comma. Radiator will listen for requests on each AuthPort and AcctPort on each BindAddress address.

Using this parameter, you can run multiple instances of Radiator on the one computer, where each Radiator listens to RADIUS requests directed to a different host address. BindAddress can include special formatting characters and multiple IPv4 and IPv6 addresses.

```
# Only listen on one network address
BindAddress 203.63.154.1
```

You can listen for requests on only some of many multi-homed addresses on this host:

```
BindAddress 200.10.5.4,200.10.7.3,::1
```

You can listen for requests directed to a particular IPv6 address:

```
BindAddress 2001:db8:0100:f101:0210:a4ff:fee3:9566
```

For more information, see Section 3.6. IPv6 support on page 28. Also see Section 3.7.10. BindV6Only on page 33 for more about IPv6 wildcard address :: special handling.

3.7.10. BindV6Only

This optional parameter allows you to explicitly set the `IPV6_V6ONLY` option for the sockets listening to IPv6 wildcard address.

RFC 3493 “Basic Socket Interface Extensions for IPv6” specifies a boolean socket option `IPV6_V6ONLY`. When this option is turned off, IPv6 wildcard listen socket can receive both IPv6 and IPv4 packets. Received IPv4 packets use special IPv4-mapped address format where the IPv4 address is encoded after the 96-bit prefix 0:0:0:0:FFFF.

For example, request from IPv4 address 172.16.172.2 is mapped to IPv6 address ::ffff:172.16.172.2. In this case you may need to configure your *`Client`* clause as *`<Client ::ffff:172.16.172.2>`*

`IPV6_V6ONLY` socket option is by default turned on by some operating systems and off by some others.

Perl 5.16 or later or recent enough `Socket.pm` CPAN module is required for this parameter.

For more information about IPv6 support and address binding, see Section 3.6. IPv6 support on page 28 and Section 3.7.9. BindAddress on page 33.

Note

This parameter covers only RADIUS authentication and accounting sockets up to Radiator 4.24. Starting with 4.25 it was expanded to other TCP, SCTP and UDP listen sockets Radiator creates.

3.7.11. DbDir

Specifies the directory to be used for user database files. Defaults to `/usr/local/etc/raddb` on Unix and Windows. For convenience, the DbDir directory name can be referred to as %D in any file name path in this configuration file.

```
# Look in /opt/etc/raddb for username database
DbDir /opt/etc/raddb
```

3.7.12. LogDir

Specifies the directory to be used to store log files. Defaults to `/var/log/radius` on Unix and Windows. For convenience, the LogDir directory name can be referred to as %L in any file name path in this configuration file.

```
# Put log files in /opt/radius instead
LogDir /opt/radius
```

3.7.13. LogFile

This defines the name of the log file. All logging messages are time stamped and written to this file. Each time this file is written to by Radiator, it opens, writes, and then closes the file. This means that you can safely rotate the log file at any time. The file name can include special path name characters. For more information about special characters, see Section 3.3. Special formatters on page 21. The default is *`%L/logfile`*, i.e. a file named `logfile` in *`LogDir`*. For more information, see Section 3.7.12. LogDir on page 34.

Here is an example of using LogFile and special characters:

```
# Log file goes in /var/log, with year number
LogFile /var/log/%Y-radius.log
```

You can disable all logging to the log file by setting *`LogFile`* to be the empty string:

```
# Disable logging to log file completely
LogFile
```

If the file name starts with a vertical bar character `|` then the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output is appended to the named file.

```
# Pipe to my-log-prog
LogFile |/usr/local/bin/my-log-prog
```

Note

If *`LogFile`* is defined in your configuration file, *`<Log FILE>`* is invisibly created to handle it. For more information, see Section 3.25. <Log FILE> on page 135. You can customise the logging format, and also log microseconds by using *`<Log FILE>`* and its parameters instead of *`LogFile`*.

3.7.14. LogMicroseconds

For all configured loggers and the LogStdout logger, enables microseconds in the time stamp. *`LogMicroseconds`* requires `Time::HiRes` Perl module. If this module is not installed, the microseconds part in the time stamp will be all zeroes `000000`. `Time::HiRes` is typically part of the Perl default installation or packaged separately on some systems such as Red Hat derived systems where the RPM is called `perl-Time-HiRes`

Tip

LogMicroseconds can also be enabled separately for specific loggers using the *`LogMicroseconds`* parameter in the respective Log clause.

3.7.15. PidFile

The name of the file where *`radiusd`* will write its process ID (PID) at start-up. Defaults to **`%L/radiusd.pid`** on Unix and Windows. The file name can include special path name characters as defined in Section 3.3. Special formatters on page 21. If the directory containing the file does not exist, it will attempt to create the directory first.

If *`PidFile`* is defined in configuration with no value, no file is written.

```
# So we do not conflict with another radiusd
PidFile /tmp/radiusd2.pid
```

3.7.16. DictionaryFile

The name of the RADIUS dictionary file. The dictionary file defines the names to be used for RADIUS attributes and their values. Its format is described in Section 9.1. Dictionary file on page 485. The file name can include special path name characters as defined in Section 3.3. Special formatters on page 21. It can also include multiple comma-separated file names. The default is **`%D/dictionary`**, i.e. a file called “dictionary” in DbDir. A dictionary file called “dictionary” that will work with most NASs and Terminal Servers is included in the Radiator distribution.

```
# Dictionary file is in the default installation directory
DictionaryFile /opt/radiator/radiator/dictionary
```

You can load the normal dictionary and locally configured dictionary with something like this:

```
# Need the old Ascend non-vendor-specific attributes too
DictionaryFile /opt/radiator/radiator/dictionary, %D/dictionary.local
```

3.7.17. ProxyUnknownAttributes

If this optional parameter is set, Radiator will forward attributes that are not present in the dictionary. Unknown attributes are not proxied by default.

3.7.18. DiameterDictionaryFile

This optional parameter specifies additional Diameter dictionary entries. The entries in *`DiameterDictionaryFile`* can replace or override any of the default entries hardwired into `DiaDict.pm`. Unlike *`DictionaryFile`*, only one dictionary file name can be specified. The Diameter dictionary is only used if you have a *`ServerDIAMETER`* clause in your configuration file. The file name can include special path name

characters as defined in Section 3.3. Special formatters on page 21. The default is to use only the hardwired dictionary in `DiaDict.pm`.

```
DiameterDictionaryFile %D/my_private_diameter_attrs.dat
```

3.7.19. DictionaryReloadInterval

DictionaryReloadInterval is an optional parameter that sets an interval in seconds for checking whether the files defined by DictionaryFile on page 35 have changed. If there are changes, all files are reloaded. Not enabled by default and the files are only loaded during server initialisation.

```
# Check every 30 minutes for dictionary changes
DictionaryReloadInterval 1800
```

3.7.20. Syslog

This parameter is now obsolete, and is replaced by the <Log SYSLOG> clause. For more information, see Section 3.26. <Log SYSLOG> on page 138.

3.7.21. LicenseFile

The name of the file from which *`radiusd`* reads its license configuration parameters. There is no default. The file name can include special path name characters as defined in Section 3.3. Special formatters on page 21. If the file does not exist, or Radiator is fully licensed, *`LicenseFile`* and its contents are ignored.

License configuration parameters are not needed with fully licensed Radiator. Only certain evaluation or locked configurations need these parameters, as instructed by Radiator sales.

```
# Read custom parameters for this evaluation license
LicenseFile %D/license.conf
```

3.7.22. SnmpgetProg

Specifies the full path name to the snmpget program. This optional parameter is only used if you are using Simultaneous-Use with a NasType of Livingston or any other NAS type that uses SNMP (see table in Section 3.14.35. NasType on page 109) in one of your Client clauses. Defaults to `/usr/bin/snmpget`.

Tip

You should use the snmpget from Net-SNMP [http://www.net-snmp.org/]. Do not use the snmpget from CMU: its style of output is not understood by Radiator.

```
SnmpgetProg /usr/local/bin/snmpget
```

3.7.23. SnmpwalkProg

Specifies the full path name to the snmpwalk program. This optional parameter is only used if you are using Simultaneous-Use with a NasType of Ascend, CiscoVPDN or TigrisOld in one of your Client clauses. Defaults to `/usr/bin/snmpwalk`.

Tip

You should use the snmpwalk from Net-SNMP [http://www.net-snmp.org/]. Do not use the snmpwalk from CMU: its style of output is not understood by Radiator.

```
SnmpwalkProg /usr/local/bin/snmpwalk
```

3.7.24. FingerProg

Specifies the full path name to an external finger program. This optional parameter is only used if you are using Simultaneous-Use with a NasType of Portslave, Ascend, Shiva, Computone or any other NAS type that uses finger (see table in Section 3.14.35. NasType on page 109) in any of your Client clauses. Defaults to using the standard Perl Net::Finger client that does not require an external program.

```
FingerProg /usr/local/bin/finger
```

3.7.25. PmwhoProg

Specifies the full path name to the pmwho program. This optional parameter is only used if you are using Simultaneous-Use with a NasType of TotalControl or any other NAS type that uses pmwho (see table in Section 3.14.35. NasType on page 109) in one of your Client clauses. Defaults to `/usr/local/sbin/pmwho`.

```
PmwhoProg /usr/local/bin/pmwho
```

3.7.26. LivingstonMIB

This optional parameter specifies the name of the Livingston SNMP MIB. It is only used if you are using Simultaneous-Use with a NasType of Livingston in one of your Client clauses. Defaults to .iso.org.dod.internet.private.enterprises.307

This parameter is deprecated and will not be supported in the future.

3.7.27. LivingstonOffs

Specifies the global default value of where the last S port is before the one or two ports specified in LivingstonHole are skipped (usually 22 for US, 29 for Europe). This optional parameter is only used if you are using Simultaneous-Use with a NasType of Livingston in one of your Client clauses. Defaults to 29.This value can be overridden on a per-Client basis by using LivingstonHole in a Client clause. For more information, see Section 3.14.43. LivingstonOffs on page 115.

3.7.28. LivingstonHole

Specifies the global default value of the size of the hole in the port list (usually 1 for US, 2 for Europe) that occurs at LivingstonOffs. This optional parameter is only used if you are using Simultaneous-Use with a NasType of Livingston in one of your Client clauses. Defaults to 2. This value can be overridden on a per-Client basis by using LivingstonHole in a Client clause. For more information, see Section 3.14.44. LivingstonHole on page 115.

3.7.29. RewriteUsername

This is an optional parameter. It enables you to alter the username in authentication and accounting requests. For more details and examples, see Section 8. Rewriting user names on page 484.

3.7.30. SocketQueueLength

This optional parameter allows you to alter the lengths of the radius socket queues from their default Operating System specific value. You may wish to set the queue lengths to be longer than the default if your Radiator server is handling very large numbers of requests, and is near its performance limits. You should never need to set them to shorter than the default. SocketQueueLength affects the length of both the authentication and the accounting socket queues. SocketQueueLength has no effect on Windows.

Tip

You may need special privileges, or you may need to change your Operating System configuration to permit longer queue lengths than the default. Consult your operating system manuals for details on how to do this.

```
# Make a long queue length
SocketQueueLength 1000000
```

3.7.31. DefineFormattedGlobalVar

Defines a value for a global variable that can be accessed anywhere special formatting characters are permitted. The syntax is:

```
DefineFormattedGlobalVar variablename value
```

This example defines the global variable called **`variablename`** to be the string **`value`**. The value can be accessed where special formatting characters are permitted with **`%{GlobalVar:variablename}`**.

Within value, special formatting characters are permitted, so you can have one global variable that depends on another global variable.

In the following example, the log file will be called **`./detail-server1`**:

```
DefineFormattedGlobalVar servername server1
LogFile ./detail-%{GlobalVar:servername}
```

3.7.32. DefineGlobalVar

Similar to *`DefineFormattedGlobalVar`*, except that special formatting characters in value are not honoured.

3.7.33. StartupHook

This optional parameter allows you to define a Perl function that will be called during server startup and restarts. Only one argument is passed to the hook: $_[0] will be set to *`undef`* during startup and 1 for a restart (usually due to a SIGHUP).

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

StartupHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change or set up environment variables, umasks etc.

```
# Set up a umask to use for the life of this process
StartupHook sub { umask(0222);}
```

3.7.34. ShutdownHook

This optional parameter allows you to define a Perl function that will be called just before exiting after receiving a SIGTERM. No arguments are passed.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

ShutdownHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change or set up environment variables, umasks etc.

```
# Delete a lock file
ShutdownHook sub { unlink '/tmp/xyzzy.lck';}
```

3.7.35. DelayedShutdownTime

DelayedShutdownTime instructs *radiusd* to delay restart or termination for the configured time. When the configured time has passed, restart or termination is done when there are no more requests to serve from the sockets.

The delay has two phases:

1. Wait for the configured amount of seconds before the requested restart or termination action is started.
2. Serve the remaining requests from the incoming sockets.

This allows *radiusd* to process any queued requests before restart or termination.

For more information, see Section 3.7.36. DelayedShutdownHook on page 39.

```
# When shutdown is triggered, wait for Radius clients to stop sending
DelayedShutdownTime 5
```

3.7.36. DelayedShutdownHook

DelayedShutdownHook is called immediately when *radiusd* is signalled to restart or terminate, and *DelayedShutdownTime* has been set. This hook can, for example, signal upstream proxies about the impending shutdown.

For more information, see Section 3.7.35. DelayedShutdownTime on page 39. *DelayedShutdownHook* is passed the following argument:

- String variable that defines the action. Either **restart** or **termination**

```
# Tell NASes to stop sending traffic to us
DelayedShutdownHook file:"%D/delayed-shutdown-hook.pl"
```

3.7.37. PreClientHook

This optional parameter allows you to define a Perl function that will be called during packet processing. *PreClientHook* is called for each request before it passed to a *Client* clause. A reference to the current request is passed as the only argument.

CAUTION

At the time this hook is run, integer, address and other special type attributes such as `Service-Type` and `NAS-IP-Address`, have not yet been unpacked and decoded, and encrypted attributes have not yet been decrypted. If you need unpacked, decrypted versions of these attributes, consider using a per-client *`ClientHook`* instead.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

`PreClientHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. The current request has its `{Client}` member set to a pointer to the *`Client`* clause handling the request.

```
# Fake a new attribute into the request
PreClientHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.7.38. ClientHook

This hook will be called for each request after the request has been decoded but before any other per-Client processing is done. A reference to the current request is passed as the first argument and the Client object as the second argument.

3.7.39. HandlerFindHook

Specifies an optional Perl hook that can be used to avoid linear Handler lookup when there are multiple Handlers. This is advantageous for configurations, such as proxying based on realm, where maximum packet throughput is required. The hook is expected to return the Handler to use or nothing. If nothing is returned, lookup continues as defined by Section 3.31. <Handler attribute=value,attribute=value, ...> on page 149.

The hook is passed the following arguments

- $_[0] A reference to the current RADIUS request
- $_[1] Current username without realm part
- $_[2] Realm part of the username, if any

HandlerFindHook is not set by default. See `goodies/handler-find-hook.cfg` for a configuration sample.

3.7.40. USR1Hook

This optional parameter allows you to define a Perl function that will be called when a SIGUSR1 signal is received by Radiator. On Unix, SIGUSR1 normally increases the logging level by 1. If you define a USR1Hook, your hook will be called instead.

3.7.41. USR2Hook

This optional parameter allows you to define a Perl function that will be called when a SIGUSR2 signal is received by Radiator. On Unix, SIGUSR2 normally decreases the logging level by 1. If you define a USR2Hook, your hook will be called instead.

3.7.42. WINCHHook

This optional parameter allows you to define a Perl function that will be called when a SIGWINCH signal is received by Radiator.

3.7.43. MainLoopHook

This optional parameter allows you to define a Perl function that will be called once per second from within the main dispatch loop.

3.7.44. UsernameCharset

This optional parameter checks that every user name consists only of the characters in the specified character set. This can be useful to reject requests that have user names that cannot be valid. The value of the parameter is a Perl character set specification. See your Perl reference manual for details about how to construct Perl character set specifications. Note that the some special characters must be escaped with a backslash. This parameter is not set by default and no character set check is done.

`UsernameCharset` is available as a global and Handler level parameter. The character set checks are done for both User-Name attribute and EAP identity.

When a request is processed by a Handler, User-Name attribute must pass both global and per Handler *`UsernameCharset`* checks. When an EAP-Response/Identity message is handled by an AuthBy, the EAP identity must pass both global and per Handler *`UsernameCharset`* checks. The Handler is the last Handler that processed the request before it was passed to the AuthBy.

This example permits only alphanumeric, period, underscore, the @-sign, and dash. Note that a dash at the end of character class needs not to be escaped with a backslash:

```
UsernameCharset a-zA-Z0-9._@-
```

3.7.45. User

On Unix, this optional parameter sets both the effective and real user ID (UID) that *`radiusd`* runs as, provided *`radiusd`* starts as a suitably privileged user, such as root. The value can be a valid Unix user name or an integer UID.

3.7.46. Group

On Unix, this optional parameter sets both the effective and real group ID (GID) and supplementary groups that *`radiusd`* runs as, provided *`radiusd`* starts as a suitably privileged user, such as root. The value can be a comma-separated list of valid Unix group names or integer GIDs. The first group is set as the effective group ID.

If any of the group names fail to resolve, the call to change the groups is not done.

3.7.47. MaxChildren

Specifies the maximum number of Fork children permitted at any one time. Any attempt by an AuthBy to Fork (if so configured) will fail and the RADIUS request will be ignored if there are already that many Forked children in existence. Defaults to 0, meaning no limit.

3.7.48. SnmpNASErrorTimeout

This optional parameter specifies for how long (in seconds) SNMP simultaneous use checks will be blocked after an SNMP error during communications with a given NAS. Defaults to 60 seconds.

3.7.49. ForkClosesFDs

This optional parameter tells Radiator to forcibly close all the child processes files descriptors after a fork() on Unix. This is only necessary in very unusual circumstances where child processes interfere with the parents connections to an SQL database.

3.7.50. ResponseTimeThreshold

ResponseTimeThreshold tells Radiator to log a warning when the processing time exceeds configured millisecond threshold. The warning contains the request's User-Name and information about the Client, Handler, and the AuthBy that processes the request.

```
# Log if request handling takes more than 0.1 seconds
ResponseTimeThreshold 100
```

3.7.51. GlobalMessageLog

This optional parameter defines the Identifier of global MessageLog clause to log all Radius, Diameter, and TACACS+ protocol requests. This parameter is not set by default and no message logging is done.

You can specify multiple *GlobalMessageLog* parameters, typically one for each protocol. The general format is:

GlobalMessageLog protocol, identifier[, extras ...]

- *protocol* defines the protocol. The possible values are:
  - *radius*
  - *radsec*
  - *diameter*
  - *tacacsplus*
- *identifier* is the Identifier or the *MessageLog* clause that does the logging.
- *extras* is currently unused.

For more information about MessageLog clauses, see Section 3.131. <MessageLog xxxxxx> on page 434.

Here is an example of using *GlobalMessageLog*:

```
# Log all RADIUS and Diameter messages with separage MessageLog clauses
GlobalMessageLog radius,messagelograd
GlobalMessageLog diameter,messagelogdia
```

Note

In Radiator 4.20 or later, radius protocol does not include the messages sent and received over RadSec. To log these messages, define the protocol as *radsec*.

3.7.52. FarmSize

This optional parameter allows you to specify how many server instances to create in a server farm. A server farm can improve performance in some cases on Unix platforms that support **fork** system call.

A server farm is a set of identical Radiator servers, all monitoring the same RADIUS sockets. Incoming RADIUS requests are distributed among the child servers in the server farm. The main server acts as a

supervisor, and restarts children that die or are terminated. Defaults to not set, which means no server farm, and only a single instance of Radiator is run. When *FarmSize* is configured, a shared duplicate cache is also strongly recommended. For more information, see *DupCache* on page 43.

Tip

To log server farm instance, see special formatting parameter %O on page 21 and configuration parameter LogFarmInstance on page 43.

CAUTION

This parameter, and the use of server farms is not compatible with many EAP protocols, such as EAP-TLS, EAP-TTLS, PEAP etc. This is because such protocols rely in authentication state being held within each server process, and it is necessary for all the requests for such protocols to go to the same Radiator instance.

3.7.53. LogFarmInstance

LogFarmInstance flag parameter allows logging server farm instance number of a worker. *LogFarmInstance* can be configured for global level and Log clause level. *LogFarmInstance* enables prepending server farm instance number to the messages logged to STDOUT, when *LogStdout* is enabled, and to log file configured with <*Log FILE*> and <*Log SYSLOG*>. For more information, see Section 3.7.2. LogStdout on page 30.

Tip

This parameter is typically enabled when *FarmSize* parameter is set. When *FarmSize* is set, the farm supervisor logs with instance number 0. When *FarmSize* is not set, instance number is always 0.

```
# Enable server farm and prepend farm instance number to log messages
FarmSize 5
LogFarmInstance
```

3.7.54. FarmChildHook

Perl hook that is run in each child when FarmSize is used. The hook is run when the child is started or restarted.

3.7.55. DupCache

This optional parameter defines if duplicate cache is implemented locally within the process, shared between processes with mmapped memory or if a Gossip cluster will provide a global shared cache for detecting duplicate requests. When FarmSize on page 42 is configured, using a shared duplicate cache is strongly recommended. Defaults to **`local`**.

For more about Gossip, see Section 3.134. <GossipRedis> on page 439 and the configuration sample `goodies/farmsize.cfg`.

Possible values are:

- **`local`**: Local to each process. The duplicate cache is not shared

- **`shared`**: Shared between processes on the same server. Requires `Cache::FastMmap` Perl module.
- **`global`**: Shared between processes using the Gossip framework

```
# See DupCacheFile for the shared file location
DupCache shared
```

3.7.56. DupCacheFile

DupCacheFile sets the shared memory based cache file name. Special characters are allowed. %0 is replaced by `-pid-timestamp-random` where the actual values are generated dynamically. Perl module `Cache-FastMmap` is required.

Defaults to `/tmp/radiator-dupcache-sharefile%0` or `C:\radiator-dupcache-sharefile%0` on Windows.

CAUTION

DupCacheFile should be readable only by Radiator process. It contains the sent RADIUS reply messages which may include sensitive information.

```
# For example: /var/run/radiator/hashbalance-dupcache-38479-1473082581-9264
DupCacheFile /var/run/radiator/hashbalance-dupcache%0
```

3.7.57. EAP_UseState

`EAP_UseState`, currently set to off by default, enables or disables the use of *`State`* attribute with EAP for the whole server.

If your configuration uses *`<AuthBy EAPBALANCE>`*, consider switching to another AuthBy, such as *`<AuthBy HASHALANCE>`*, to avoid adding a second *`State`* in the proxied requests.

Here is an example of using *`EAP_UseState`*:

```
# Use State attribute for identifying all EAP authentication conversations
EAP_UseState
```

3.7.58. Identifier

This allows you to assign a symbolic name to a *`radiusd`* server instance, similar to Identifier for an AuthBy, Handler, and any other clause. This allows hooks, logging, Gossip framework, and other code to use a name that uniquely identifies a *`radiusd`* instance. The global Instance parameter does not have a value by default. Special formatting characters are supported.

```
# Set identifier based on the hostname and server farm instance number
Identifier Radiator-%h-%O
```

Note

A 32 hex character long hash is calculated from the formatted Identifier for possible later use. This fixed length value can be accessed from hooks and other custom code.

3.7.59. DisableMTUDiscovery

If this optional parameter is set, it disables MTU discovery on platforms that support that behaviour (currently Linux only). This can be used to prevent discarding of certain large RADIUS packet fragments on supporting operating systems.

3.7.60. PacketDumpOmitAttributes

This optional parameter specifies a comma separated list of RADIUS attribute names which will be omitted from RADIUS packet dumps in logs.

```
PacketDumpOmitAttributes EAP-Message,User-Password
```

3.7.61. StatusServer

Normally, when a Status-Server request is received, Radiator replies with some statistics including the total number of requests handled, the current request rate and so on. You can control Status-Server response by setting *`StatusServer`* to one of the following values:

- **`off`**

  Status-Server requests are ignored.

- **`minimal`**

  Reply without any attributes.

- **`uptime`**

  Reply without server uptime and, in case of RADIUS/TCP and RADIUS/TLS (RadSec), connection uptime .

- **`default`**

  Reply with statistics.

3.7.62. DisabledRuntimeChecks

Radiator tries to check for commonly required but missing modules, some known security vulnerabilities and possible other runtime parameters when it starts up. Any Hooks may also call the runtime check module functions, as required by the Hook authors. Special formatting characters are supported.

Any checks that do not pass are logged but no other action is taken.

The currently recognised built-in checks are:

- *`CVE-2014-0160`* - the OpenSSL vulnerability commonly called Heartbleed
- *`Digest::MD4`* - MD4 is required by MSCHAP and MSCHAP-v2 and their derivatives

The optional DisabledRuntimeChecks parameter allows you to define the checks that should not be run.

Check for CVE-2014-0160 is done by trying to load Net::SSLeay and using the functions it provides to check for vulnerable OpenSSL versions. Many vendors have patched their OpenSSL for CVE-2014-0160 without changing the OpenSSL version number. For this reason the check may report your OpenSSL as vulnerable. The Net::SSLeay functions for reporting OpenSSL version are only present in recent Net::SSLeay versions. For this reason Radiator may log a message about version check not being able to determine OpenSSL version.

Digest::MD4 is required by MSCHAP, MSCHAP-V2 and their derivatives such as EAP-MSHCHAP-V2. We recommend having Radiator Radius::UtilXS or Digest::MD4 installed unless you are sure you will never need to support these authentication protocols. See Section 2.1.9. Radiator Radius::UtilXS on page 6 and Section 2.1.5. MD4 digest for MSCHAP and MSCHAPv2 on page 5 for more information.

```
# Our OpenSSL is patched but still reports vulnerable version
DisabledRuntimeChecks CVE-2014-0160
```

3.7.63. PBKDF2_MinRounds

This is an integer that defines the minimum number of rounds PBKDF2 algorithm. If the stored user password uses less rounds than this parameter specifies, the password check fails. The default value is **100**.

3.7.64. PBKDF2_MaxRounds

This is an integer that defines the maximum number of rounds PBKDF2 algorithm. If the stored user password uses more rounds than this parameter specifies, the password check fails. The default value is **200000**.

3.8. SQL configuration

SQL clauses use the Perl DBI/DBD interface to connect to your database. You can therefore use SQL clauses with a large number of commercial and free SQL database systems. In order to use SQL, the minimum installation to get your SQL system to work is:

- Database software
- Matching Perl DBD module
- Perl DBI module

The SQL parameters *DBSource*, *DBUsername*, and *DBAuth* are passed to DBI like this:

```
DBI->connect(DBSource, DBUsername, DBAuth)
```

DBSource is a specification, which usually starts with *dbi:drivername:...*, but the exact meaning of these variables depends on the Perl DBD driver you use. For more information about the syntax of *DBSource*, *DBUsername*, and *DBAuth* for different database vendors, see Section 19. Using SQL with various database vendors on page 527.

You can specify multiple databases by using multiple *DBSource*, *DBUsername*, and *DBAuth* parameters. Whenever Radiator tries to connect to a database, SQL tries to connect to the first *DBSource* listed, using the first *DBUsername* and *DBAuth* parameters. If that connection fails, it tries the second, third and so on, until all the databases are tried, and finally gives up without replying to the NAS. This gives your NAS the opportunity to fall back to another RADIUS server if all your SQL databases are down. You can change this default behaviour with the *RoundRobinOnFailure* configuration parameter. For more information, see Section 3.8.8. RoundRobinOnFailure on page 50.

SQL clauses are tolerant of database failures. If the database server goes down, Radiator tries to reconnect to a database as described above, starting again at the first database or using round robin, depending on the configuration. Whichever database Radiator connects to, it stays connected to it until that database becomes unreachable, at which time it searches again for a database, starting at the first. If Radiator is not able to connect to any SQL server, it returns an IGNORE, which causes Radiator to ignore the request. This causes most NASs to fall back to a secondary RADIUS server.

Tip

Remember the SQL database needs maintenance. Make sure you have someone who is knowledgeable about installing, configuring, maintaining, and backing up your SQL database.

Tip

In some cases SQL database server use a significant number of CPU cycles and become the performance bottleneck. For example, doing a simple lookup on a user/password database with a simple index on the user name is usually very quick, in the order of milliseconds per lookup, even if the table has millions of rows. However, an insert into a large accounting table with a complicated index can be very slow. Make sure you understand how to design and tune your database tables, otherwise it could have a significant effect on SQL performance.

The following clauses utilise SQL:

- <AcctLog SQL> on page 380
- <AddressAllocator SQL> on page 383
- <AuthBy PRESENCESQL> on page 337
- <AuthBy RADMIN> on page 221
- <AuthBy SQL> on page 190
- <AuthBy SQLAUTHBY> on page 348
- <AuthBy SQLFAILUREPOLICY> on page 368
- <AuthBy SQLHOTP> on page 344
- <AuthBy SQLRADIUS> on page 249
- <AuthBy SQLTOTP> on page 346
- <AuthBy SQLYUBIKEY> on page 339
- <AuthLog SQL> on page 372
- <ClientList SQL> on page 118
- <Log SQL> on page 142
- <SessionDatabase SQL> on page 122
- <ServiceDatabase SQL> on page 133
- <StatsLog SQL> on page 431

The following clauses utilise AuthBy SQL and thus support all the common SQL parameters:

- <AuthBy SQLDIGIPASS> on page 271
- <AuthBy EMERALD4> on page 224
- <AuthBy FREERADIUSSQL> on page 312
- <AuthBy RADMIN> on page 221
- <AuthBy WIMAX> on page 337

3.8.1. SQL bind variables

Most SQL servers support the use of bind variables. Bind variables are used by the SQL server to do a dynamic replacement of variables in an SQL statement. In some SQL servers this can increase query performance by allowing the server to compile and reuse the SQL compiled SQL query many times.

With Radiator SQL clause query parameters that support bind variables, you can specify the query separately from the values for the bind variables. With most SQL servers, the position of each bound variable is marked by a question mark ('?') character. The bound variables will be replaced (after special characters are replaced) at run-time one by one in the order of the question marks in the query, and in the order of the bound variable specifications.

For example, <AuthBy SQL> supports bound variables with the AuthSelect query parameter. In this sample configuration fragment:

```
AuthSelect select PASSWORD from SUBSCRIBERS where USERNAME=? and CLIENT=?
AuthSelectParam %0
AuthSelectParam %N
```

%0 (user name) will be used to replace the first ? (the one for the *USERNAME* column), and *%N* (NAS id) will be used to replace the second (the one for the *CLIENT* column).

Some SQL queries are cached when configured with bind variables. By default as much as 32 queries can be cached. This can be changed in SqlDb.pm if required.

Note

Most SQL servers and their Perl DBD modules support bound variables. Check the documentation for your SQL server, and the Perl DBD module for your server.

3.8.2. DBSource

This parameter is used by Perl DBI to specify the database driver and database system to connect to. It usually begins with *dbi:drivername:*. There is no standard for the text following the driver name, consult the details for your DBD (database driver) documentation. You can use any of the special characters described in Section 3.3. Special formatters on page 21. Here are some examples.

```
# Connect to MySQL database called radius. Typically defaults to Unix socket
DBSource dbi:mysql:database=radius

# Or... Use SQLIte file called users.db located in the DbDir directory
DBSource dbi:SQLite:%D/users.db

# Or... Connect to the Oracle SID called users
DBSource dbi:Oracle:users

# Or... Connect to PostgreSQL database called radius on localhost, default port
DBSource dbi:Pg:dbname=radius;host=127.0.0.1
```

Tip

For some applications, it is useful to use a *GlobalVar* to specify the name of the SQL database. That way your Radiator can be parameterised from the command line:

```
DBSource dbi:mysql:%{GlobalVar:databasename}
radiusd -config_file xxxxxx.cfg databasename=radius
```

3.8.3. DBUsername

For most database types, this specifies the user name to log in to the database. For some databases, this has a different meaning. For example, for SQLite user name is meaningless. You can use any of the special characters described in Section 3.3. Special formatters on page 21.

```
# For SQLite, it is ignored
DBUsername ignored

# For Oracle, it is the name of the Oracle user to log in as
DBUsername scott
```

3.8.4. DBAuth

This parameter is usually used by Perl DBI to specify the password for the user specified in *DBUsername*. For some databases, this has a different meaning. For example, for SQLite it is meaningless and can be ignored. You can use any of the special characters described in Section 3.3. Special formatters on page 21.

```
# For SQLite, it is ignored
DBAuth ignored

# For Oracle, it is Oracle password for DBUsername
DBAuth tiger
```

3.8.5. Timeout

This optional parameter specifies a timeout interval in seconds that Radiator waits for SQL queries. When a query times out, it is retried until *SQLRetries* limit is reached. For more information about *SQLRetries*, see Section 3.8.7. SQLRetries on page 50.

If *ConnectTimeout* is not defined, *Timeout* is also be used when trying to connect or disconnect an SQL database specified by *DBSource*.

If the server does not respond within the *Timeout* period, Radiator considers the SQL server to be failed, and stops trying to contact the SQL server until the *FailureBackoffTime* is expired. The default value for *Timeout* is **60** seconds. If you set Timeout to **0**, no timeouts are implemented, and Radiator relies on the underlying implementation to timeout any SQL operations.

```
# Set the timeout to two seconds
Timeout 2
```

Note

Timeout is not supported on Perl for Windows. On Windows platforms, the timeout usually is determined by the TCP timeouts built in to your Windows TCP stack.

Note

When DBD supports defining connection or query timeouts, those driver-specific timeout options must be used in *DBSource*. In this case, the value of *Timeout* must be larger than the timeout value in *DBSource*.

Tip

Set *`Timeout`* to **`0`** If you are using Sybase ODBC libraries.

3.8.6. FailureBackoffTime

If Radiator detects an SQL server failure, it waits for this number of seconds before trying to contact the SQL server again. The default value is **`600`** (10 minutes).

```
# Try again after 3 minutes
FailureBackoffTime 180
```

3.8.7. SQLRetries

If Radiator detects certain SQL errors while running a query, it will reconnect and retry until it has tried SQLRetries times, then declares an SQL server failure. The default value is 2. Applies to SQL errors other than primary key violations, or Oracle error 'ORA-00001'.

CAUTION

If SQLRetries is set to 0, no connection is made and no queries are executed.

3.8.8. RoundRobinOnFailure

This optional flag helps with some types of overloaded database that can be connected but then time out when a query is sent. It causes the next database in the *`DBSource`* list to be tried next instead of the first one.

```
# Try to skip databases that have become slow to respond
RoundRobinOnFailure
```

3.8.9. ConnectTimeout

This optional parameter specifies a timeout interval in seconds that Radiator waits for when trying to connect or disconnect an SQL server specified by *`DBSource`*. If this parameter is not set, value of *`Timeout`* is used for connection handling. For more information about *`Timeout`*, see Section 3.8.5. Timeout on page 49.

Some databases may leak resources, such as file descriptors, when Radiator times out a connection before the DB driver does. With *`ConnectTimeout`*, SQL connection timeout can different than *`Timeout`* that is used for SQL queries.

```
# DB driver connect timeout is shorter than 20 seconds
# but we want 2 second query timeout
ConnectTimeout 20
Timeout 2
```

3.8.10. SQLRecoveryFile

Note

This feature is known not to work as expected with some types of database. Its use is deprecated: you are strongly discouraged from using this feature. Support for it may be removed in future versions.

This optional parameter specifies the name of a file where any failed SQL do queries aree logged, perhaps for later recovery. The default is no logging. The SQLRecovery-File file is always opened written and closed for each failed SQL do query, so you can safely rotate it at any time.

Tip

Make the file name depend on the date. This way, all the missed accounting records per day are located in a single file.

Here is an example of using *`SQLRecoveryFile`* and logging all failed queries of one day into a one file:

```
# Log all failed SQL queries to a log file per day
SQLRecoveryFile %L/sqlfailures-%Y-%m-%d
```

3.8.11. ConnectionHook

This optional parameter specifies a Hook that is run every time this clause connects or reconnects to the SQL database. This is most useful for executing *`func()`* to configure the database connection in customised ways. The hook is called with 2 arguments. The first is a reference to the clause object that is making the connection. The second argument is the DBH handle to the newly connected database.

In the following example, the hook calls DBI func() to configure an Interbase database connection for custom requirements:

```
ConnectionHook sub {$_[1]->func(-access_mode => 'read_write',\
      -isolation_level => 'read_committed',\
      -lock_resolution => 'wait',\
      'ib_set_tx_param')}
```

3.8.12. ConnectionAttemptFailedHook

You can run this hook whenever Radiator attempts to connect to an SQL database and fails to connect. The default is to log the failure. The hook is called with 4 arguments: *`$object`*, *`$dbsource`*, *`$dbusername`*, *`$dbauth`*. *`$object`* is the SqlDb object trying to connect. The other parameters are the currently used values for *`DBSource`*, *`DBUsername`*, and *`DBAuth`*.

In the following example the default hook is replaced with a hook that logs unobscured password.

```
ConnectionAttemptFailedHook sub { \
 my $self = $_[0]; my $dbsource = $_[1]; \
 my $dbusername = $_[2]; my $dbauth = $_[3]; \
 $self->log($main::LOG_ERR, "Could not connect to SQL database with DBI->connect \
            $dbsource, $dbusername, $dbauth: $@ $DBI::errstr"); }
```

3.8.13. NoConnectionsHook

You can run this hook whenever Radiator fails connect to any SQL server. The default is to log the failure. The hook is called with 1 argument: *`$object`*. *`$object`* is the SqlDb object that was trying to connect.

In the following example the default hook is replaced with a hook that logs a very short message.

```
NoConnectionsHook sub { \
  my $self = $_[0]; \
  $self->log($main::LOG_ERR, "Could not connect to any SQL database"); }
```

3.8.14. ConnectSQLAtStartup

This optional flag parameter causes radiusd to connect to SQL database immediately when Radiator starts. This flag is not set by default and the connection to SQL is first made when the first SQL query is done.

See AsynchronousSQL on page 52 and `goodies/addressallocator.sql` for a configuration sample.

3.8.15. AsynchronousSQL

This optional flag parameter tells *`radiusd`* to use asynchronous SQL queries. This flag is not set by default and the queries are synchronous which means no processing is done before the reply, or timeout, is received from the SQL database. With asynchronous queries, *`radiusd`* can do other processing while the query is being processed by the database. When the the database result is ready, *`radiusd`* continues from the point where the asynchronous query was started.

Starting with Radiator 4.18, asynchronous queries are supported with MySQL, MariaDB and PostgreSQL. Testing was mainly done with DBD::mysql 4.035 with MariaDB 10.1.13.

Note

Asynchronous SQL queries were introduced with Radiator 4.18 and are only supported by AddressAllocator SQL on page 383 clause.

Here is an example of *`AsynchronousSQL`* and the parameters typically used with it:

```
# See goodies/addressallocator.cfg for full example
# Run address allocator in asynchronous mode
<AddressAllocator SQL>
    # Other parameters
    AsynchronousSQL
    AsynchronousSQLConnections 10
    ConnectSQLAtStartup
    RoundRobinQueries
</AddressAllocator>
```

3.8.16. AsynchronousSQLConnections

This optional parameter defines the size of connection pool used when AsynchronousSQL on page 52 is enabled. The default value is 1 which means that only a single connection is established with the database.

See AsynchronousSQL on page 52 and `goodies/addressallocator.sql` for a configuration sample.

3.8.17. RoundRobinQueries

This optional flag parameter enables load balancing SQL queries to all defined DBSources on page 48 when AsynchronousSQL on page 52 is enabled. This flag is not set by default and only one of the defined DBSources is used at the time.

See AsynchronousSQL on page 52 and `goodies/addressallocator.sql` for a configuration sample.

3.9. LDAP configuration

Radiator's LDAP support requires Perl `Net::LDAP` module version 0.32 or later. Operating system vendors and Windows Perl distributions typically include `Net::LDAP`. If it is not present in your Perl distribution, see Section 2.1.2. CPAN on page 3 for how to obtain and install it. `Net::LDAP` works with Microsoft Active Directory, Novell/NetIQ eDirectory, OpenLDAP, 389 Directory Server/FreeIPA, and other LDAP servers.

When an LDAP clause needs to fetch information from the LDAP server, it connects to the LDAP server specified by *`Host`*. Optionally, you can authenticate Radiator as a valid user of the LDAP server by specifying *`AuthDN`* and *`AuthPassword`*. This is not the same thing as authenticating a user. It happens before querying the LDAP server, and proves that this *`radiusd`* is allowed to talk to the LDAP database.

LDAPS and StartTLS connection methods are supported. A number of TLS specific parameters control verification of server certificates. TLS client authentication with certificate is also supported.

Important

System certificate store is not used. The TLS configuration parameters must cover all CAs and other settings that are needed.

At present, LDAP clauses do synchronous connections and searches. This can mean significant delays if your LDAP server is reached by a slow network connection, or your LDAP server is slow. If this is the case, consider putting the LDAP server in a sub-server and having your main Radiator forward requests for that realm to the RADIUS sub-server.

The following configuration clauses utilise LDAP:

- <ClientList LDAP> on page 115
- <AuthBy LDAP2> on page 226
- <AuthBy LDAPRADIUS> on page 247
- <AuthBy LDAPDIGIPASS> on page 276
- <AuthBy LDAP_APS> on page 278

SASL Authentication of the LDAP connection

LDAP clauses support SASL authentication of the connection to the LDAP server. If SASL authentication is specified, the LDAP server uses SASL to authenticate the SASL user credentials specified by *`SASLUser`* and *`SASLPassword`*. You must configure your LDAP server to enable SASL authentication, and to map SASL user names to LDAP server administrator names. For example, when using OpenLDAP see their SASL configuration guide for the details.

3.9.1. BaseDN

This is the base DN relative to which the searches are made.

Special formatting characters are permitted. See the specific LDAP configuration clauses for more information about the clause-specific formatting characters.

Here is an example of using *BaseDN*:

```
# Start looking here
BaseDN o=University of Michigan, c=US
```

3.9.2. SearchFilter

Each Radiator LDAP clause defines a default *SearchFilter* that sets the conditions an entry in the directory must meet to be returned by the search.

Special formatting characters are permitted. See the specific LDAP configuration clauses for more information about the clause specific formatting characters. For example, the default search filter for *<ClientList LDAP>* is:

```
SearchFilter (objectclass=oscRadiusClient)
```

3.9.3. Scope

This optional parameter allows you to control the search scope used to find the user. The default value is **sub**.

The permitted options are:

- **base**
- **one**
- **sub** or **subtree**
- **children** Requires Net::LDAP 0.48 or better and LDAPv3 subordinate feature extension

Here is an example of using *Scope*:

```
# We know where the entry should be
Scope base
```

3.9.4. AuthDN

This is the optional name to use to authenticate this Radiator server to the LDAP server. You only need to specify this if the LDAP server requires authentication from its clients. *AuthBy LDAP2* supports **%0** as a special for DN escaped currently authenticated user name.

Here is an example of using *AuthDN*:

```
# Log in to LDAP as admin. You man need a DN formatted name.
# AuthDN admin
AuthDN cn=admin,dc=example,dc=com
```

3.9.5. AuthPassword

This is the optional password to use to authenticate this Radiator server to the LDAP server. You only need to specify this if the LDAP server requires authentication from its clients, and you specify *AuthDN*.

Here is an example of using *AuthPassword*:

```
# log in to LDAP with password adminpassword
AuthPassword adminpassword
```

3.9.6. Host

This specifies the LDAP host names to connect to. The default value is *localhost*. Special formatting characters are permitted. Multiple host names can be specified, and in this case Radiator tries connecting each configured host individually until it succeeds. If a *Host* name begins with *ipv6:* the subsequent host names are interpreted as IPv6 addresses where possible, and `Net::LDAP` uses IPv6 to connect to the LDAP server. IPv6 may require `IO::Socket::INET6`, `Socket6` or other IPv6 specific Perl modules depending on your Perl version.

Here are examples of using *Host*:

```
Host ldaphost.example.com
```

Same as above, but resolve the name to individual addresses before connecting:

```
# May resolve to multiple addresses.
# Implicitly sets SSLExpectedServerName to ldaphost.example.com for each address.
Host ldaphost.example.com
ResolveHost
```

When there are multiple Host values, connect to the first server that works:

```
# Connect to first available server
Host ldaphost1.example.org
Host ldaphost2.example.org
Host ldaphost3.example.org
```

IPv4 and IPv6 address can also be used. When TLS is enabled, we need to tell TLS library what is the expected name in each Host's server certificate:

```
# Server 10.20.30.11 has name ldaphost1.example.org in its certificate.
# Server 10.20.30.22 is named ldaphost2.example.org, respectively.
Host 10.20.30.11
Host 10.20.30.22

SSLExpectedServerName ldaphost1.example.org
SSLExpectedServerName ldaphost2.example.org
```

3.9.7. Port

This parameter specifies the port to connect to the LDAP host. The default value is **389**, the standard port for unencrypted LDAP. If *UseSSL* is specified, the default value is **636**, the standard port for encrypted LDAP. *Port* can be a numeric port number or a symbolic service name from `etc/` services or its equivalent on your system. Usually, there is no need to override the defaults. *Port* can contain special formatting characters. A typical use of special formatting characters is with *GlobalVar* and command line arguments.

Here is an example of using *Port*:

```
# Connect using the SSL encrypted port
Port 636
```

3.9.8. ResolveHost

ResolveHost flag parameter causes Radiator to resolve LDAP server name to addresses instead of passing the name directly to the LDAP library. This allows Radiator to have separate connection and failure backoff parameters for each address. To refresh resolved addresses, name resolution is done periodically and after

LDAP connection, bind and other failures. Periodic refresh is currently done after one hour has elapsed from the previous refresh. This allows adding and withdrawing server addresses from name service without restarting Radiator. For more examples, see Section 3.9.6. Host on page 55.

```
# Name ldaphost.example.com resolves to multiple addresses.
# Implicitly sets SSLExpectedServerName to ldaphost.example.com for each address.
Host ldaphost.example.com
ResolveHost
```

3.9.9. UseSSL

This optional parameter specifies to use direct TLS, often called LDAPS, to connect to the LDAP server. This is an alternative for upgrading to TLS with StartTLS operation. For more about StartTLS support, see Section 3.9.10. UseTLS on page 56.

Here is an example of using *`UseSSL`*:

```
# Enable direct SSL/TLS (LDAPS)
UseSSL
```

A full SSL/TLS configuration requires setting the certificate locations and possible other parameters.

```
# Enable direct SSL/TLS (LDAPS) and tell it where to find certificates
UseSSL

# Name of the client certificate file:
SSLCAClientCert %D/certificates/cert-clt.pem
# Name of the file containing the client private key
SSLCAClientKey %D/certificates/cert-clt.pem

# Only need to set one of the following
#SSLCAPath %D/cadirectory
SSLCAFile %D/certificates/demoCA/cacert.pem
```

Tip

All certificates are required to be in PEM format.

Tip

If both *`UseSSL`* and *`UseTLS`* are specified, *`UseSSL`* is prioritised.

3.9.10. UseTLS

This optional parameter is used in a similar way as *`UseSSL`*. For more information, see Section 3.9.9. UseSSL on page 56. *`UseTLS`* enables StartTLS LDAP operation to upgrade the LDAP connection to use TLS authentication and encryption. *`UseTLS`* takes the same parameters as *`UseSSL`*, including *`SSLVerify`*, *`SSLCiphers`*, *`SSLCAPath`*, *`SSLCAFile`*, *`SSLCAClientCert`*, and *`SSLCAClientKey`*.

```
# Use StartTLS with this LDAP server
UseTLS
```

Tip

If both *UseSSL* and *UseTLS* are specified, *UseSSL* is prioritised.

Note

Net::LDAP 0.57 and earlier had a bug where LDAP + StartTLS followed by LDAPS failed. This happens when a clause with *UseTLS* is followed by another clause with *UseSSL*.

3.9.11. Debug

This parameter enables debugging of the Net::LDAP module. This sets the Net::LDAP debug parameter for the connection, which prints to `stdout`. See Debug_TLS on page 57 for how to view messages written to `stdout`.

You can use the following debugging options by adding together the relevant bits. The available values are shown below:

- 1

  Show outgoing packets (using asn_hexdump)

- 2

  Show incoming packets (using asn_hexdump)

- 4

  Show outgoing packets (using asn_dump)

- 8

  Show incoming packets (using asn_dump)

Here is an example of enabling *Debug* messages for outgoing and incoming packet using asn_dump:

```
Debug 12
```

3.9.12. DebugTLS

This integer parameter enables TLS debugging in `IO::Socket::SSL` for all TLS based LDAP connections. The debug messages are printed to `stderr`. The available values as listed by `IO::Socket::SSL` version 2.0.69 are shown below:

- 1

  Print out errors from `IO::Socket::SSL` and ciphers from `Net::SSLeay`.

- 2

  Print also information about call flow from `IO::Socket::SSL` and progress information from `Net::SSLeay`.

- 3

  Print also some data dumps from `IO::Socket::SSL` and from `Net::SSLeay`.

The following Radiator configuration file example enables all available debugging:

```
DebugTLS 3
```

When Radiator is run directly on console, LDAP TLS debug messages are directly visible. When Radiator is started with `systemd` utilities, use its utilities to view `stderr` messages. For example, using Linux command line:

```
% sudo journalctl -u radiator
% less /var/log/messages
```

Another option is to add a local `systemd` configuration change to redirect `stdout` and `stderr` to a file. For the details, see Section 5.1. Systemd service unit file on page 444

3.9.13. Timeout

This optional parameter sets the timeout period in seconds for the connection to the LDAP server. The default value is **`10`** seconds. If this is set to **`0`**, Radiator waits forever for LDAP connections and transactions.

If the connection times out :

- LDAP authentication request are IGNOREd
- LDAP Client list refresh is not done and Radiator continues to use the current client list

Here is an example of using *`Timeout`*:

```
# Make timeout really short, 2 seconds
Timeout 2
```

3.9.14. FailureBackoffTime

This optional parameter sets the period of time in seconds that this clause (*`AuthBy`*, *`<ClientListLDAP>`*, or other) stops trying to connect to its LDAP server after a connection failure. The default value is **`600`** seconds (10 minutes). This is intended to give the LDAP server time to recover after a failure. During the failure back-off interval, all authentication requests are IGNOREd.

3.9.15. BindFailedHook

This optional parameter allows you to define a Perl function that is run after an LDAP bind with *`AuthDN`* fails.

`BindFailedHook` is called with the following arguments:

1. A handle to the current `Radius::Ldap` object
2. *`AuthDN`* that failed LDAP bind
3. *`AuthPassword`* that was used with the failed LDAP bind
4. Error string from LDAP library
5. String with LDAP server and port the connection was made to
6. Possible SASL error string

The default is to log the error with the password set to `**obscured**`. If password debugging is required, redefine the hook or see console output when *`Debug`* flag parameter is enabled for the LDAP clause. For more information about Debug, see Section 3.9.11. Debug on page 57.

3.9.16. HoldServerConnection

By default the LDAP clauses, except *`<AuthBy LDAPRADIUS>`*, disconnect from the LDAP server after each authentication. This is because not all LDAP servers permit multiple searches from the same LDAP connection.

HoldServerConnection forces holding the connection to the LDAP server up for as long as possible. It is an optional parameter and available for *<AuthBy LDAP2>* and *<AuthBy LDAPDIGIPASS>*.

Most of the LDAP servers support this behaviour and it can significantly improve performance, especially where *UseTLS* or *UseSSL* is enabled. If you enable this parameter and get unwanted behaviour, you are probably using an unsupported LDAP server. In this case, remove this parameter.

Here is an example of using *HoldServerConnection*:

```
# Our server supports multiple searches
HoldServerConnection
```

Note

In some cases, using *HoldServerConnection* with *ServerChecksPassword* of *<AuthBy LDAP2>* may cause failure situations. This is due to some LDAP servers' behaviour when the password check fails but the connection is not closed. A failure situation may also occur when the password check succeeds but the user is not allowed to perform searches in the server. If your users experience unexpected authentication failures, try testing your system without using these 2 parameters together.

3.9.17. NoBindBeforeOp

This optional parameter prevents Net::LDAP from binding with the *AuthDN* and password prior to a search operation.

Note

This is an advanced parameter. Most installations do not need to use it.

3.9.18. SSLVerify

This parameter can be used with *UseSSL* or *UseTLS* parameters to control how LDAP server's certificate is verified. The options are:

- **none**

  No server certificate is required, and if the server supplies a certificate it is not checked.

- **optional**

  Verify if the server offers a certificate.

- **require**

  The server must provide a certificate, and it must be valid.

The default value is **require**. Format specifiers, such as *%{GlobalVar:name}*, are evaluated when the configuration is loaded.

Tip

require is the most secure option.

3.9.19. SSLCiphers

This optional parameter specifies which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value is **`DEFAULT:!EXPORT:!LOW`**.

Here is an example of using *`SSLCiphers`*:

```
# Exclude cipher suites using RC4 too
SSLCiphers DEFAULT:!EXPORT:!LOW:!RC4
```

3.9.20. SSLCAPath

`SSLCAPath` parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS client certificates. Radiator looks for root certificates first in *`SSLCAFile`*, then in *`SSLCAPath`*, so there usually is no need to set both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files.

Special characters are supported.The certificates and CRLs must be in PEM format, one per file. The file name has a special format. Setting up this directory is described in Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81.

Here is an example of using *`SSLCAPath`*:

```
SSLCAPath %D/cadirectory
```

3.9.21. SSLCAFile

Use this option to locate the file containing the certificates of the trusted certificate authorities. Thus, you can verify that the server certificate has been signed by a reputable certificate authority. Special characters are permitted.

Here is an example of using *`SSLCAFile`*:

```
SSLCAFile %D/certificates/demoCA/cacert.pem
```

3.9.22. SSLCAClientCert

This optional parameter specifies the location of the SSL client certificate that this LDAP connection uses to verify itself with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

Here is an example of using *`SSLCAClientCert`*:

```
SSLCAClientCert %D/certificates/cert-clt.pem
```

3.9.23. SSLCAClientKey

This optional parameter specifies the location of the SSL private key that this connection uses to communicate with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

It is common for the SSL client private key to be in the same file as the client certificate. In that case, both *`SSLCAClientCert`* and *`SSLCAClientKey`* refer to the same file.

If *`SSLCAClientKey`* contains a private key in encrypted format, you need to specify the decryption password in *`SSLCAClientKeyPassword`*.

Here is an example of using *`SSLCAClientKey`*:

```
SSLCAClientKey %D/certificates/cert-clt.pem
```

3.9.24. SSLCAClientKeyPassword

This optional parameter specifies the passphrase to decrypt *`SSLCAClientKey`*. This parameter is only required when SSL client verification is required and the key file configured with *`SSLCAClientKey`* is encrypted. Special characters are permitted.

```
SSLCAClientKeyPassword whatever
```

3.9.25. SSLExpectedServerName

This optional parameter specifies the name to use when verifying LDAP server certificate. Useful, for example, when *`Host`* is configured with an IP address. Special characters are permitted. For an example, see Section 3.9.6. Host on page 55.

```
# Certificate does not have a fully qualified name
SSLExpectedServerName myserver
```

3.9.26. Version

This optional parameter sets the LDAP version number to use. Currently supported values are **2** and **3**. The default value is **3**.

Here is an example of using *`Version`*:

```
# Support LDAP protocol version 2 as
# required by very old servers
Version 2
```

3.9.27. Deref

By default, aliases are dereferenced to locate the base object for the search, but not when searching subordinates of the base object. Change this by specifying *`Deref`* with one of the following case-sensitive values:

- **`never`**

  Do not dereference aliases in searching or in locating the base object of the search.

- **`search`**

  Dereference aliases in subordinates of the base object in searching, but not in locating the base object of the search.

- **`find`**

  Dereference aliases in locating the base object of the search, but not when searching subordinates of the base object. This is the default.

- **`always`**

  Dereference aliases both in searching and in locating the base object of the search.

Note

Usually, there is no need to use this parameter.

3.9.28. UseSASL

This optional parameter tells Radiator to request SASL authentication of the connection to the LDAP server instead of the default simple authentication. *AuthDN* and *AuthPassword* are used as the SASL credentials. Only if *AuthDN* and *AuthPassword* are not defined, *SASLUser* and *SASLPassword* are used.

Using this parameter requires the Authen-SASL-2.09 module or later. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3.

3.9.29. SASLUser

Note

This parameter is deprecated.

This is a string that defines the user name, which is used for SASL authentication during the connection to the LDAP server. However, if *AuthDN* is defined, it overrides *SASLUser* and is used instead. Only if *AuthDN* is not defined and *SASLUser* is defined, *SASLUser* is used for authentication. For more information, see Section 3.9.4. AuthDN on page 54.

3.9.30. SASLPassword

Note

This parameter is deprecated.

This is a string that defines the password, which is used for SASL authentication during the connection to the LDAP server. However, if *AuthPassword* is defined, it overrides *SASLPassword* and is used instead. Only if *AuthPassword* is not defined and *SASLPassword* is, *SASLPassword* is used for authentication. For more information, see Section 3.9.5. AuthPassword on page 54.

3.9.31. SASLMechanism

If *UseSASL* is enabled, this optional parameter specifies what SASL mechanisms are to be used to authenticate the connection to the LDAP server. *SASLMechanism* is a space-separated list of mechanism names supported by Authen::SASL, such as

- **ANONYMOUS**
- **CRAM-MD5**
- **DIGEST-MD5**
- **EXTERNAL**
- **LOGIN**
- **PLAIN**

The default value is **DIGEST-MD5**. If you change this, it is possible that you need also to change your SASL to LDAP user mapping. See your SASL system documentation for details on what mechanisms are supported.

Here is an example of using *SASLMechanism*:

```
SASLMechanism DIGEST-MD5,PLAIN
```

3.9.32. BindAddress

You can specify the local address for the client side of the LDAP connection with *BindAddress*, in the form *hostname[:port]*. Special formatting characters are permitted.

Use this parameter to set *LocalAddr* parameter of the underlying IO::Socket used as the LDAP client. This is usually useful only on multi-homed hosts, where you need to control the source address of the LDAP connection, perhaps for firewall rules. There is no default and the system chooses the address.

Tip

This parameter is separate from the global *BindAddress*, see Section 3.7.9. BindAddress on page 33.

3.9.33. MultiHomed

This optional flag enables the *MultiHomed* option in Net::LDAP and IO::Socket for this LDAP connection. If this is set then Net::LDAP tries all addresses for a multi-homed LDAP host until one is successful. The default value is true.

3.10. EAP configuration

This section introduces the parameters used for EAP configuration. EAP parameters are available for several AuthBys. For more information about supported EAP methods, see Section 17. Extensible Authentication Protocol (EAP) on page 518.

Important

System certificate store is not used. The configuration parameters for TLS based EAP methods must cover all CAs, CRLs and other settings that are needed.

3.10.1. EAPType

This optional parameter specifies which EAP authentication methods are permitted when EAP authentication is requested by the NAS. See RFCs 3748, 5216, and 2433 for more details. When an EAP identity request is received, Radiator replies with the first EAP type given. If the NAS requests another type, it is permitted only if it appears in *EAPType* list. It is ignored and has no effect unless EAP authentication is requested. This parameter is not set by default, which means that Radiator does not perform EAP authentication by default. The allowed values for *EAPType* are given by the following table.

Table 6. Allowed values for EAPType

EAPType	Explanation
MD5	This is the default value. Use **MD5-Challenge** as per RFC 3748. This can be used with any authentication method that provides a plaintext password, such as *<AuthBy FILE>*, *<AuthBy SQL>*, and *<AuthBy LDAP2>*. See goodies/eap_md5.cfg for example configuration. **MD5-Challenge** is an old alias for **MD5**.
OTP	Use One-time-password authentication as per RFC 3748. This requires a one-time password authenticator such as *<AuthBy OTP>*. **One-Time-Password** is an old alias for **OTP**.

EAPType	Explanation
GTC	Use Generic Token authentication as per RFC 3748. This requires a token-based authenticator such as *<AuthBy OTP>*, *<AuthBy ACE>*, or *<AuthBy RSAAM>*. **GenericToken** is an old alias for **GTC**.
TLS	Use Transport Layer Security (TLS). This can be used with any authentication method. TLS checks that the client certificate is valid and has a short enough certificate chain to the root certificate. It requires an SSL certificate for the server and one on each client requiring authentication. See `goodies/eap_tls.cfg` for example configuration.
TTLS	Use Tunnelled TLS. This can be used with any authentication method. TTLS does not usually involve a client certificate, but the client may be configured to check the server's SSL certificate. See `goodies/eap_ttls.cfg` for example configuration.
PEAP	Use PEAP tunnel as used by Windows XP and others. This can be used with any authentication method. See `goodies/eap_peap.cfg` for example configuration.
LEAP	This is compatible with Cisco LEAP authentication, a proprietary authentication protocol. LEAP requires an authenticator that supplies plaintext passwords, such as *<AuthBy FILE>*, *<AuthBy SQL>*, or *<AuthBy LDAP2>*, or MSCHAPV2, such as *<AuthBy LSA>*.
SIM	Use EAP-SIM which authenticates against SIM cards. This requires the additional EAP-SIM bundle from Radiator Software.
AKA	Use EAP-AKA. This requires the additional EAP-SIM bundle from Radiator Software, which contains support for EAP-AKA.
AKA-PRIME	Use EAP-AKA'. This requires the additional EAP-SIM bundle from Radiator Software, which contains support for EAP-AKA'.
MSCHAP-V2	Use EAP-MSCHAPV2, which is commonly tunneled inside PEAP
TNC	Support EAP-TNC, a protocol for assessing the security posture of end points.
FAST	Use EAP-FAST, a rarely-used protocol from Cisco.
PAX	Use EAP-PAX (Password Authenticated Exchange)
PSK	Use EAP-PSK (Pre-Shared Key).
PWD	Use EAP-pwd, a method which uses a shared password for authentication.

3.10.2. EAPContextTimeout

This optional parameter specifies the maximum time period in seconds an EAP context is retained. The default value is 120 seconds. Usually there is no need to change this.

3.10.3. EAPAnonymous

For tunnelling EAP types, such as TTLS and PEAP, this optional parameter specifies the User-Name that is used in the RADIUS request resulting from the EAP inner request. The default value is **anonymous**. Special characters can be used. %0 is replaced by the EAP identity of the inner EAP request.

This parameter is useful when proxying the inner authentication. If there is a realm in the *EAPAnonymous* name, it is used to choose a local Realm to handle the inner authentication.

3.10.4. EAPTLS_CAFile

For TLS based EAP types such as TLS, TTLS and PEAP, this parameter specifies the name of a file containing Certificate Authority (CA) root certificates that may be required to validate TLS client certificates. The certificates must be in PEM format. The file can contain several root certificates for one or more CA's. Radiator looks for root certificates first in *EAPTLS_CAFile*, then in EAPTLS_CAPath, so there usually is no need to set both.

EAPTLS_CAFile is expected to contain a stack of CA one or more CA certificates that will be used to validate client certificates. The list of CA issuers in that is also sent to the client during handshaking to tell the client which certificates Radiator accepts.

Special characters are supported.

3.10.5. EAPTLS_CAPath

For TLS based EAP types such as TLS, TTLS and PEAP, this parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS client certificates. Radiator looks for root certificates first in *EAPTLS_CAFile*, then in *EAPTLS_CAPath*, so there usually is no need to set both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files. Special characters are supported.

The certificates and CRLs must be in PEM format, one per file. The file name has a special format. Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81.

3.10.6. EAPTLS_CertificateFile

For TLS-based EAP types, such as TLS, TTLS and PEAP, this parameter specifies the name of a file containing the RADIUS server certificate. The server certificate is sent to the EAP client and validated by the client during EAP authentication. The certificate file may be in PEM or ASN1 format, depending on the setting of the *EAPTLS_CertificateType* parameter. The certificate file can also contain the server's TLS private key if the *EAPTLS_PrivateKeyFile* parameter specifies the same file. If the server certificate is a chain of certificates, use *EAPTLS_CertificateChainFile* instead.

Special characters are supported.

Important

System certificate store is not used. The TLS configuration parameters must cover all CAs, CRLs and other settings that are needed.

3.10.7. EAPTLS_CertificateChainFile

For TLS-based EAP types, such as TLS, TTLS and PEAP, this parameter specifies the name of a file containing a certificate chain for the Radius server certificate. The server certificate chain is sent to the EAP client and validated by the client during EAP authentication. The certificate chain must be in PEM format. *EAPTLS_CertificateChainFile* can be used instead of *EAPTLS_CertificateFile* for explicitly constructing the server certificate chain which is sent to the client.

Use *EAPTLS_CertificateChainFile* to specify a chain of certificates that the server uses to identify itself to the client. If there is only one server certificate, and not a chain, you can use *EAPTLS_CertificateFile* instead.

Special characters are supported.

3.10.8. EAPTLS_CertificateType

For TLS-based EAP types, such as TLS, TTLS and PEAP, this optional parameter specifies the format of the *EAPTLS_CertificateFile* and *EAPTLS_PrivateKeyFile*. The options are:

- **PEM**
- **ASN1**

The default value is **ASN1** which means binary DER format. Special formatting characters are supported.

3.10.9. EAPTLS_PrivateKeyFile

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter specifies the name of the file containing the server's private key.

The private key file must be in PEM or ASN1 format, depending on the setting of the *EAPTLS_CertificateType* parameter.

It can be located in the same file as the server certificate (*EAPTLS_CertificateFile*). If the private key is encrypted, which is usually the case, *EAPTLS_PrivateKeyPassword* is the key to decrypt it.

Special characters are supported.

3.10.10. EAPTLS_PrivateKeyPassword

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter specifies the password that is used to decrypt the *EAPTLS_PrivateKeyFile*.

Special characters are supported.

3.10.11. EAPTLS_Protocols

This specifies the comma-separated list of TLS protocols that are permissible for TLS-based EAP types. Currently supported protocols are:

- TLSv1
- TLSv1.1
- TLSv1.2
- TLSv1.3

> **CAUTION**
>
> TLSv1.3 is not enabled by default with Radiator yet. Testing reports are welcome. Net::SSLeay 1.83 or later is required if you use Radiator with SSL/TLS library that has TLSv1.3 enabled. Net::SSLeay 1.92 or later is recommended.

There is no default value. If this parameter is not set, all protocols, which are supported by the SSL/TLS library within Radiator, except of TLSv1.3, are permitted.

The available TLS protocols depend on your system's TLS library. See Radiator debug log for details when the first EAP message is processed by this AuthBy.

```
# Do not allow TLSv1
EAPTLS_Protocols TLSv1.1, TLSv1.2
```

3.10.12. EAPTLS_Ciphers

This parameter specifies which subset of cipher suites is permissible for TLS-based EAP types. It uses the standard OpenSSL string format. The default is **`DEFAULT:!EXPORT:!LOW`**.

```
# Exclude cipher suites using RC4 too
EAPTLS_Ciphers DEFAULT:!EXPORT:!LOW:!RC4
```

When SSL library supports security levels, a level can be set for each configuration clause together with cipher string. For more information about security levels, see Section 3.10.13. EAPTLS_SecurityLevel on page 67

```
# System default level 2 is too strict for this module
EAPTLS_Ciphers DEFAULT:!EXPORT:!LOW@SECLEVEL=1
```

3.10.13. EAPTLS_SecurityLevel

`EAPTLS_SecurityLevel` is an integer parameter that sets the SSL library security level for the enclosing clause. Security levels were added in OpenSSL 1.1.0 and are detailed in OpenSSL manual page `SSL_CTX_set_security_level`. There is no default and the system default is used. This parameter may be needed if you see unexpected TLS errors with older clients.

Security level can also set together with ciphers in which case *`EAPTLS_SecurityLevel`* is not needed. For more information, see Section 3.10.12. EAPTLS_Ciphers on page 67

```
# The default on this system, 2, is too strict for our clients,
# lower it for this Radiator module
EAPTLS_SecurityLevel 1
```

3.10.14. EAPTLS_RandomFile

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter specifies the name of a file containing randomness. Usually there is no need to set this parameter.

Special characters are supported.

3.10.15. EAPTLS_DHFile

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter specifies the name of the DH group. Usually there is no need to set this parameter. It is required if you use ephemeral DH keys.

Special characters are supported.

3.10.16. EAPTLS_ECDH_Curve

For TLS-based EAP types, this optional parameter enables ephemeral EC keying by specifying the name of the elliptic curve to use. The default is to not enable ephemeral EC keying.

```
# Curves often have multiple aliases. Need OpenSSL name here.
# openssl ecparam -list_curves
EAPTLS_ECDH_Curve prime256v1
```

3.10.17. EAPTLS_AllowUnsafeLegacyRenegotiation

This optional parameter enables legacy insecure renegotiation between OpenSSL and unpatched clients or servers. It is used with TLS-based EAP types, such as TLS, TTLS, and PEAP, and with OpenSSL version 0.9.8m or later. OpenSSL 0.9.8m and later always attempts to use secure renegotiation as described in RFC5746. This counters the prefix attack described in CVE-2009-3555 and elsewhere.

3.10.18. EAPTLS_MaxFragmentSize

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter specifies the maximum size in octets permitted for each TLS message fragment. The default value is **2048**, but many EAP clients, routers, and wireless Access Points have limitations that require *EAPTLS_MaxFragmentSize* to be set as low as 1000 or less. Setting this number too small can result in excessive RADIUS request round trips during EAP TLS authentication. This slows down the authentication process. Setting this number too large can result in failure to complete TLS authentication for some types of clients and devices. Many customers find that 1300 is a good compromise.

The EAP packet that is encapsulated inside EAP-Message and all other radius attributes must not exceed one Ethernet frame because EAP does not support fragmentation.

Depending on the number of other RADIUS attributes your switches or WLAN controllers send to the RADIUS servers, you can increase *EAPTLS_MaxFragmentSize*, which may result in fewer RADIUS requests in the EAP conversation which reduces the authentication time and lowers to load on both the RADIUS client (switch, WLAN controller) and RADIUS server.

If incoming RADIUS requests have Framed-MTU that is less than *EAPTLS_MaxFragmentSize*, then Radiator uses the reported Framed-MTU to limit fragment size when doing TLS, TTLS, PEAP, and PSK.

Special characters are supported.

3.10.19. EAPTLS_CRLCheck

This optional flag parameter specifies if Certificate Revocation List must be checked for revoked certificates. It is used with TLS-based EAP types, such as TLS, TTLS, and PEAP, that have been configured to check client certificates.

3.10.20. EAPTLS_CRLCheckUseDeltas

This optional flag parameter specifies if Delta Certificate Revocation List must be checked for revoked certificates in addition to base CRL. It is used with TLS-based EAP types, such as TLS, TTLS, and PEAP, that have been configured to check client certificates. Currently delta CRL files are loaded with *EAPTLS_CRLFile* parameter, similar to base CRL files.

CAUTION

EAPTLS_CRLCheckUseDeltas is currently experimental.

Before enabling *EAPTLS_CRLCheckUseDeltas*, note the following requirements and restrictions:

- *EAPTLS_CRLCheck* must be enabled in Radiator configuration
- Both base and delta CRLs must use CRL v2 format
- Do not use delta CRL files without enabling *EAPTLS_CRLCheckUseDeltas*
- OpenSSL indicates only one delta CRL file can be used
- Review OpenSSL notes about delta CRLs on OpenSSL manual page for X509_VERIFY_PARAM_set_flags. [https://www.openssl.org/docs/manmaster/man3/X509_VERIFY_PARAM_set_flags.html]
- Test that your base and delta CRL work when CRL files are updated or refreshed

Please contact Radiator support about success or possible problems there might be with delta CRLs.

3.10.21. EAPTLS_CRLCheckAll

For TLS-based EAP types, this optional flag parameter enables CRL checks for the entire certificate chain. *EAPTLS_CRLCheckAll* is not enabled by default. The CRL files for the intermediate CAs must be found, otherwise the certificate check fails. See Section 3.10.22. EAPTLS_CRLFile on page 69 for the details.

Note

EAPTLS_CRLCheck must be enabled for any certificate checks to happen.

3.10.22. EAPTLS_CRLFile

For TLS-based EAP types, such as TLS, TTLS, and PEAP, and where CRL checking has been enabled with *EAPTLS_CRLCheck*, this optional parameter specifies one or more CRL files that are used to check client certificates for revocation. These files are also used when *EAPTLS_CRLCheckAll* is enabled. Special characters are supported.

If a CRL file is not found, or if the CRL says the certificate has been revoked, TLS authentication will fail with an error:

```
SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
```

To ease automation, CRLs may follow a file naming convention where each CRL file uses a special file name in *EAPTLS_CAPath* directory. Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81. In this case you do not need to configure *EAPTLS_CRLFile*.

If CRLs are not stored in the CAPath directory, one or more CRLs can be named with multiple *EAPTLS_CRLFile* parameters. The intended way CRL reloading works is this: Each CRL file named with *EAPTLS_CRLFile* will be automatically reloaded and reread at the start of each new EAP-TLS, EAP-TTLS or PEAP session if the modification date of the named CRL file has changed since the last time it was loaded. If the CRL for a particular issuer changes, it is sufficient to replace the existing CRL file with the newer version and Radiator will reload the new CRL when required.

Tip

Operating system wildcards are supported, so you can name multiple CRLs with a single wildcard like:

```
EAPTLS_CRLFile %D/crls/revocations-*.pem
```

3.10.23. EAPTLS_SessionResumption

For TLS-based EAP types such as TLS, TTLS, and PEAP, this optional parameter enables or disables support for TTLS Session Resumption and PEAP Fast Reconnect. It is enabled by default.

```
# Disable session resumption
EAPTLS_SessionResumption 0
```

Special characters are supported.

3.10.24. EAPTLS_SessionResumptionLimit

For TLS-based EAP types such as TLS, TTLS, and PEAP, this optional parameter allows you to limit how long after the initial session that a session can be resumed. The time is given in seconds and the default value is 43200 seconds (12 hours).

Special characters are supported.

3.10.25. EAPTLS_SessionContextId

For TLS-based EAP types, such as TLS, TTLS, and PEAP, this optional parameter allows you to set the context within which the TLS session resumption is allowed. The default value is **`%1:%3:%n`**, which means that TLS session resumption is allowed if the resumed and the full authentication are processed by the same Handler, EAP Type, and original user name.

Special characters are supported. **`%0`** is replaced by value referring to Client, **`%1`** is replaced by value referring to Handler, **`%2`** is replaced by value referring to AuthBy, and **`%3`** with the current EAP type number.

Here is an example of using *`EAPTLS_SessionContextId`*:

```
# Allow resumption when using the same Client and Handler
EAPTLS_SessionContextId %0%1
```

3.10.26. EAPTLSRewriteCertificateCommonName

For TLS-based EAP types such as TLS, TTLS, and PEAP, this optional parameter allows you to rewrite the Common Name in the client's TLS certificate before using it to find the user name in the Radiator database:

```
EAPTLSRewriteCertificateCommonName s/testUser/mikem/
```

3.10.27. EAPTLS_PEAPVersion

For PEAP, this optional parameter allows you to control which version of the draft PEAP protocol to honour. The default value is **`0`**. Set it to **`1`** for unusual clients.

3.10.28. EAPTLS_PEAPBrokenV1Label

This optional parameter makes PEAP Version 1 support compatible with non-standard PEAP V1 clients that use the old TLS encryption labels. These labels appear to be used frequently, due to Microsoft's use of the incorrect label in its V0 client.

3.10.29. EAP_PEAP_MSCHAP_Convert

For EAP-PEAP MSCHAPV2 authentication, this optional parameter tells Radiator to convert the inner EAP-MSCHAPV2 request into a conventional RADIUS-MSCHAPV2 request. The new RADIUS-MSCHAPV2 request is re-despatched to the Handlers, where it is detected and handled with *`<Handler ConvertedFromEAPMSCHAPV2= 1>`*.

This is useful when proxying PEAP-MSCHAPV2 inner requests to a remote RADIUS server that can handle ordinary RADIUS-MSCHAPV2, but not EAP authentication. See `goodies/eap_peap_mschap_proxy.cfg` in your distribution for an example of how to configure convert and proxy system.

3.10.30. EAPTLS_RequireClientCert

This optional parameter forces TTLS and PEAP to require a valid client certificate to be presented by the client. TLS always requires a valid client certificate, regardless of the setting of this parameter. In this case, valid means that the certificate is within its validity dates and has been signed by a CA for which Radiator has a root certificate.

3.10.31. EAPTLS_PolicyOID

For TLS based EAP types such as TLS, TTLS, and PEAP, when the client presents a certificate, this optional parameter enables certificate policy checking. It also specifies one or more policy OIDs that must be present

in the certificate path. It sets the 'require explicit policy' flag as defined in RFC3280. Using this requires Perl `Net::SSLeay` module 1.37 or later.

When multiple EAPTLS_PolicyOID parameters are configured, the peer certificate needs to match only one of the configured OIDs, not all of them.

```
# Require just one policy
EAPTLS_PolicyOID 1.3.6.1.4.1.9048.33.2
```

3.10.32. EAP_LEAP_MSCHAP_Convert

For LEAP authentication, this optional parameter tells Radiator to convert the LEAP request into a conventional RADIUS-MSCHAP request. The new RADIUS-MSCHAP request is re-despatched to the Handlers, where it is detected and handled with <Handler ConvertedFromLEAP=1>.

This is useful for proxying LEAP requests to a remote RADIUS server that cannot handle LEAP, but which can handle RADIUS-MSCHAP. See `goodies/eap_leap_proxy.cfg` in your distribution for an example of how to configure such a convert and proxy system.

3.10.33. EAP_GTC_MaxLength

This parameter specifies the maximum length of EAP GTC token that is accepted from the client. The default value is **253**, which is compatible with RADIUS protocol.

3.10.34. EAP_GTC_PAP_Convert

For EAP-GTC authentication, this optional parameter tells Radiator to convert the EAP-GTC request into a conventional RADIUS-PAP request. The new RADIUS-PAP request is re-despatched to the Handlers, where it is detected and handled with <Handler ConvertedFromGTC=1>.

3.10.35. EAP_MSCHAPv2_UseMultipleAuthBys

> **Note**
>
> Avoid using this parameter. It will probably be deprecated if additional features, such as support for password change, are implemented in EAP MSCHAP-V2. If you need support for multiple user database lookups, use *`EAP_PEAP_MSCHAP_Convert`* instead. For more information, see Section 3.10.29. EAP_PEAP_MSCHAP_Convert on page 70.

This optional parameter is used with EAP MSCHAP-V2 authentication. It allows using multiple AuthBys within one Handler or AuthBy GROUP. If this parameter is not set, using multiple EAP MSCHAP-V2 causes an authentication failure. To avoid this, define *`EAP_MSCHAPv2_UseMultipleAuthBys`* in all AuthBys within Handler or AuthBy GROUP. This flag is not set by default.

Here is an example of using *`EAP_MSCHAPv2_UseMultipleAuthBys`*:

```
<Handler>
 AuthByPolicy ContinueUntilAcceptOrChallenge
  <AuthBy SQL>
    EAPType MSCHAP-V2
    EAP_MSCHAPv2_UseMultipleAuthBys
  </AuthBy>
   <AuthBy SQL>
```

```
    EAPType MSCHAP-V2
    EAP_MSCHAPv2_UseMultipleAuthBys
  </AuthBy>
</Handler>
```

3.10.36. EAPTLS_NoCheckId

For EAP-TLS authentication, this optional parameter prevents matching the *User-Name* to certificate *CN* or *subjectAltName* and then using the matched value to fetch the user from the user database. *EAPTLS_CommonNameHook* on page 73 and *EAPTLSRewriteCertificateCommonName* on page 70 are not run and the user's check and reply attributes are not applied because no user lookup is done. This allows Radiator to mimic the behaviour of some other RADIUS servers.

The certificate will be accepted based only on the validity dates and the verification chain to the root certificate. *EAPTLS_OCSPCheck* on page 75 is allowed and *EAPTLS_CertificateVerifyHook* on page 72 is run.

3.10.37. EAPTLS_CertificateVerifyHook

For EAP-TLS authentication, this optional parameter specifies a Perl function that is called after the request user name or identity has been matched with the certificate CN. It passes the certificate and various other details, and returns a different user name which is used to do the user database lookup.

The function is passed the following arguments:

- *$_[0]*: *$matchedcn*, the CN that matched the user name or identity with or without the domain name. *$matchedcn* is the CN in the certificate that was matched against either the user name or EAP identity. It is normally used as the user name to do the user database lookup, but you can return a new name from this function.
- *$_[1]*: *$x509_store_ctx*, the EAP SSLEAY store context (you can pass this to Net::SSLeay::X509_STORE_CTX_get_current_cert)
- *$_[2]*: *$cert*, the current certificate, result of Net::SSLeay::X509_STORE_CTX-_get_current_cert($x509_store_ctx)
- *$_[3]*: *$subject_name*, the certificates subject name, result of &Net::SSLeay::X509_get_subject_name($cert)
- *$_[4]*: *$subject*, the certificate subject, result of &Net::SSLeay::X509_NAME_oneline($subject_name)
- *$_[5]*: *$p*, the current Radius::Radius request

The function is expected to return a new value for *$matchedcn*, which is used to do the user database lookup. If it returns undef, the certificate verification is deemed to fail with the OpenSSL error X509_V_ERR_APPLICATION_VERIFICATION.

3.10.38. EAPTLS_CertificateVerifyFailedHook

EAPTLS_CertificateVerifyFailedHook specifies a Perl function that is called if the certificate cannot be verified. It is an optional parameter and is used with EAP-TLS authentication. It is passed the certificate (if present), and various other details.

The peer certificate *$cert* is not always present. An example of such case is a policy OID mismatch .

EAPTLS_CertificateVerifyFailedHook is passed the following arguments:

- *$_[0]: $verify_error*

  This is the EAP SSLEAY store context verification code.

- *$_[1]: $x509_store_ctx*

  This is the EAP SSLEAY store context.

- *$_[2]: $cert*

  This is the current certificate. May be undefined.

- *$_[3]: $subject_name*

  This is the certificate's subject name. Undefined when *$cert* is undefined.

- *$_[4]: $subject*

  This is the certificate subject. Undefined when *$cert* is undefined.

- *$_[5]: $p*

  This is the current Radius::Radius request.

EAPTLS_CertificateVerifyFailedHook must return a single value. This value is used as an OpenSSL error code to set the verify result code as follows:

- > 0: Non-zero error code

  This is a new verification result code.

- 0

  This changes verification failure to verification success.

- < 0

  The verification process is immediately stopped with "verification failed" state.

- Undefined

  This is handled as an OpenSSL error *X509_V_ERR_APPLICATION_VERIFICATION*.

Here is a example of using *EAPTLS_CertificateVerifyFailedHook*. This configuration accepts all certificates. Any additional authorisation must be done later.

```
EAPTLS_CertificateVerifyFailedHook sub { return 0; }
```

The following example allows expired certificates. **10** is *X509_V_ERR_CERT_HAS_EXPIRED*.

```
EAPTLS_CertificateVerifyFailedHook sub { \
                if ($_[0] == 10) { return 0; } else { return $_[0]; } }
```

Note

This parameter may cause security issues if not used properly. Use it only in special cases.

3.10.39. EAPTLS_CommonNameHook

This optional parameter specifies a Perl hook that is used to choose the authenticated CN from the client certificate during EAP-TLS authentication. Normally, EAP-TLS attempts to match each CN in the client certificate (after *EAPTLSRewriteCertificateCommonName* is executed) against the User-Name (with and without any trailing @domain) and the EAP identity (with and without any trailing @domain). If a match is found, that is the authenticated CN, and it is the name that is be used to look up the user name in the user database.

If *EAPTLS_CommonNameHook* is defined, it returns the user name that matches with the CN.

It is called for each CN in the client certificate with the following arguments:

- *$_[0]*: the CN
- *$_[1]*: the User-Name from the incoming request
- *$_[2]*: the EAP Identity of the TLS handshake
- *$_[3]*: $p, reference to the current request

It is expected to return the matched CN or `undef` if no match is found. When `undef` is returned, the name checks continue similarly as when the hook is not configured at all.

3.10.40. EAPTLS_TraceState

EAPTLS_TraceState is a flag parameter that turns on TLS state tracing for TLS-based EAP modules such as EAP-TLS, EAP-TTLS and PEAP. This may be useful for additional debugging of TLS handshake processing. *EAPTLS_TraceState* is not set by default.

Radiator 4.27 with `Net::SSLeay` 1.92 and later automatically enables TLS handshake message logging when log level is set to DEBUG or higher with Trace on page 136 or PacketTrace on page 105 parameter.
When TLS message logging is enabled, TLS state tracing can also be enabled by setting *EAPTLS_TraceState*. TLS state and messaging logging contain similar information and in most cases it's not useful have them both enabled.

Here is an example of using *EAPTLS_TraceState*:

```
# Turn on TLS state tracing for this AuthBy
EAPTLS_TraceState
```

3.10.41. EAPTLS_CopyToInnerRequest

This parameter overrides the default list of attributes that is copied from an outer EAP request, such as PEAP, EAP-TTLS, or EAP-FAST, to an inner request. The default list of attributes for PEAP and EAP-FAST is **`NAS-IP-Address NAS-Identifier NAS-Port Calling-Station-Id`**. The default list is empty for EAP-TTLS.

Here is an example of using *EAPTLS_CopyToInnerRequest*:

```
EAPTLS_CopyToInnerRequest Calling-Station-Id, Called-Station-Id
```

3.10.42. EAPTLS_CAPartialChain

This is a flag parameter. It is used with TLS-based EAP types, such as TLS, TTLS, and PEAP. When it is enabled, only a partial CA certificate chain is required to validate TLS client certificates. This is not set by default.

3.10.43. EAPTLS_UseCADefaultLocations

This is a flag parameter. It is used with TLS-based EAP types, such as TLS, TTLS, and PEAP. When set, the default locations are used for loading CA certificates. This is not set by default.

The default CA certificate directory is `certs/` in the default OpenSSL directory. Use *SSL_CERT_DIR* environment variable to override the default location. The default CA certificate file is `cert.pm` in the default OpenSSL directory. Use *SSL_CERT_FILE* environment variable to override the default file name.

3.10.44. EAPTLS_NoClientCert

This is a flag parameter. It is used with TLS-based EAP types, such as TTLS and PEAP. When set, it disables loading of any CA certificates for client certificate verification. This allows you to simplify PEAP and EAP-TTLS configuration when client certificates are not requested with *`EAPTLS_RequireClientCert`*. Do not set this if EAP-TLS support is needed.

3.10.45. EAPTLS_OCSPCheck

This optional flag parameter is alternative to *`EAPTLS_OCSPAsyncCheck`*. This defines if Online Certificate Status must be checked for revoked certificates. It is used only with TLS-based EAP types, such as TLS, TTLS, and PEAP, that are configured to check client certificates. Using this parameter requires `LWP::UserAgent` and `HTTP::Request` Perl modules.

Here are the requirements for using OCSP parameters:

- OpenSSL 1.0.0 or later
- Net::SSLeay 1.59 or later. Net::SSLeay 1.82 or later is recommended.

  If you use *`EAPTLS_OCSPCheck`* or *`EAPTLS_OCSPStapling`*, Net::SSLeay 1.82 or later is required.

3.10.46. EAPTLS_OCSPAsyncCheck

This optional flag parameter defines if Online Certificate Status must be checked asynchronously for revoked certificates. This is used only with TLS-based EAP types, such as TLS, TTLS, and PEAP, that are configured to check client certificates. Using this parameter requires `HTTP::Async` and `HTTP::Request` Perl modules.

`EAPTLS_OCSPAsyncCheck` is alternative to *`EAPTLS_OCSPCheck`*, the same requirements apply also with *`EAPTLS_OCSPAsyncCheck`*. For more information, see Section 3.10.45. EAPTLS_OCSPCheck on page 75.

3.10.47. EAPTLS_OCSPStapling

This optional flag parameter defines if Radiator provides OCSP staple with its server certificate. This is used only with TLS-based EAP types, such as TLS, TTLS, and PEAP. Using this parameter requires `LWP::UserAgent` and `HTTP::Request` Perl modules.

Using *`EAPTLS_OCSPStapling`* does not require that *`EAPTSL_OCSPCheck`* or *`EAPTSL_OCSPAsyncCheck`* is set, but the same requirements apply when you use only *`TLS_OCSPStapling`*. See the requirements list in Section 3.10.45. EAPTLS_OCSPCheck on page 75.

3.10.48. EAPTLS_OCSPURI

This string defines the OCSP URI that is used for all OCSP queries.

3.10.49. EAPTLS_OCSPFailureBackoffTime

If the OCSP query fails, the query is not sent again to the failed OCSP URI. This integer defines the time for how long the failed OCSP URI is not contacted. The unit is seconds.

3.10.50. EAPTLS_OCSPStrict

This optional flag parameter defines if Online Certificate Status must be checked successfully. This is used only with TLS-based EAP types, such as TLS, TTLS, and PEAP, that are configured to check client certificates.

When *`EAPTLS_OCSPStrict`* is set, the certificate check must be successful in all phases. Here are some examples when the certificate check fails if *`EAPTLS_OCSPStrict`* is set:

- *`EAPTLS_OCSPURI`* is not set and the certificate does not have OCSP URI.

- OCSP responder is not responding.
- OCSP responder cannot be reached.

3.10.51. EAPTLS_OCSPCacheTime

This integer defines the time for how long OCSP replies are save in the cache. The unit is seconds. The default value is **`1200`**, that is 20 minutes.

3.10.52. EAPTLS_OCSPCacheSize

This integer defines the maximum number of OCSP cache entries. The default value is **`1000`**.

3.10.53. EAPFAST_PAC_Lifetime

For EAP-FAST, this parameter specifies the maximum lifetime for each PAC.

3.10.54. EAPFAST_PAC_Reprovision

For EAP-FAST, this parameter specifies the time after which a PAC is re-provisioned.

3.10.55. EAP_TTLS_AllowInRequest

For EAP-TTLS authentication, this optional parameter tells Radiator to allow only the specified attributes in requests from EAP-TTLS clients. Attributes that are not allowed are ignored and logged on debug level.

By default, the following attributes are allowed in requests.

- *`User-Name`*
- *`User-Password`*
- *`CHAP-Password`*
- *`CHAP-Challenge`*
- *`EAP-Message`*
- *`MS-CHAP-Response`*
- *`MS-CHAP-Challenge`*
- *`MS-CHAP2-Response`*

These are the attributes from EAP-TTLS RFC 5281 except of the password change related attributes, which are currently not allowed by default.

Here is an example of using *`EAP_TTLS_AllowInRequest`*:

```
# Also allow our vendor specific attribute in EAP-TTLS requests
EAP_TTLS_AllowInRequest OSC-AVPAIR, User-Name, User-Password, \
         CHAP-Password, CHAP-Challenge, EAP-Message, \
         MS-CHAP-Response, MS-CHAP-Challenge, MS-CHAP2-Response
```

3.10.56. EAP_TTLS_AllowInReply

For EAP-TTLS authentication, this optional parameter tells Radiator to allow only the specified attributes in replies to EAP-TTLS clients. Attributes that are not allowed are silently ignored.

By default, the following attributes are allowed in requests:

- *`EAP-Message`*

- *MS-CHAP2-Success*

These are the attributes from EAP-TTLS RFC 5281 except of the password change related attributes, which are currently not allowed by default.

Here is an example of using *EAP_TTLS_AllowInReply*:

```
# Also allow our vendor specific attribute in EAP-TTLS replies
EAP_TTLS_AllowInReply OSC-AVPAIR, EAP-Message, MS-CHAP2-Success
```

3.10.57. EAP_Identity_MaxLength

This is an integer that specifies the maximum length of EAP identity. The default value is **253**.

3.10.58. EAP_PWD_PrepMethod

This parameter specifies a password preparation method to be used in EAP-pwd authentication. RFC 5931, that defines EAP-pwd, specifies three password pre-processing methods. RFC 8146 specifies additional methods which are not implemented by Radiator yet. Preparation methods are configured with an optional parameter *EAP_PWD_PrepMethod*. The default value is **None**. The currently available methods are shown in the table below.

Table 7. Allowed values for EAP_PWD_PrepMethod

Preparation method	Explanation
None	Password is used as is. No additional preparation is done. The password must be stored in plain text, including rcrypt, format.
NtHash	Password is processed to produce the output PasswordHashHash, as defined in RFC 2759. The password must be stored in plain text, including rcrypt, or NT hashed format. This requires `Digest::MD4` Perl module.
SASLprep	Password is processed according to RFC 5931 SASLprep specification. The password must be stored in plain text, including rcrypt, format. This requires `Authen::SASL::SASLprep` version 1.100 or later.

CAUTION

EAP-pwd clients may not support other methods than **None**. For example, wpa_supplicant 2.6+fixes is needed for the **NtHash** method to work.

Here is an example of using *EAP_PWD_PrepMethod*:

```
# Our passwords are stored in {nthash} prefixed format
EAP_PWD_PrepMethod NtHash
```

3.10.59. UsernameCharset

This optional parameter checks that every user name consists only of the characters in the specified character set. This can be useful to reject requests that have user names that cannot be valid. The value of the parameter is a Perl character set specification. See your Perl reference manual for details about how to construct Perl character set specifications. Note that the some special characters must be escaped with a backslash. This parameter is not set by default and no character set check is done.

UsernameCharset is available as a global and Handler level parameter. The character set checks are done for both User-Name attribute and EAP identity.

When a request is processed by a Handler, User-Name attribute must pass both global and per Handler *UsernameCharset* checks. When an EAP-Response/Identity message is handled by an AuthBy, the EAP identity must pass both global and per Handler *UsernameCharset* checks. The Handler is the last Handler that processed the request before it was passed to the AuthBy.

This example permits only alphanumeric, period, underscore, the @-sign, and dash. Note that a dash at the end of character class needs not to be escaped with a backslash:

```
UsernameCharset a-zA-Z0-9._@-
```

3.10.60. NoEAP

This optional parameter disables EAP authentication in the specific AuthBy. If the AuthBy would otherwise do EAP authentication, this parameter forces it to do conventional authentication. This is useful for performing additional checks besides EAP authentication, for example, when doing MAC Address whitelist checking as well as EAP authentication.

```
AuthByPolicy ContinueWhileAccept
# Check the MAC address is valid..
<AuthBy FILE>
      NoEAP
      AuthenticateAttribute Calling-Station-Id
      Filename ...
      ...
</AuthBy>
# then check the username and password, perhaps by EAP
<AuthBy LDAP2>
      EAPType ....
      ....
</AuthBy>
```

3.10.61. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  PreHandlerHook is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

    PreHandlerHook specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

    If *EAP_LEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-MSCHAPv2

If *EAP_PEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

– EAP-GTC

If *EAP_GTC_PAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  PreHandlerHook is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  PreHandlerHook is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  PreHandlerHook is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  PreHandlerHook is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's *PrehandlerHook* is run before ServerTACASPLUS's *PreHandlerHook*. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

PreHandlerHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *PreHandlerHook*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.10.62. EAPTLS_KeylogFilename

Sets the name of file Radiator uses for logging TLS key material. TLS key log allows fully decrypting EAP and Stream SSL/TLS sessions, including those that have forward security enabled. TLS keylog should only be used for debugging to avoid security issues.

The keylog file is written in NSS Key Log Format, also known as SSLKEYLOGFILE. Tools, such as Wireshark, can read this file and fully decrypt TLS, including TLS sessions that have forward security enabled.

```
# Enable when debugging, remove when in production
EAPTLS_KeylogFilename %L/eap-keylog
```

DANGER

Keylog should only be used for debugging to avoid security issues.

3.11. TLS configuration

Multiple Radiator configuration clauses are built on top of common server and client stream modules. These modules support TCP and SCTP stream transport protocols and can optionally be configured to use TLS for transport security.

The exact behaviour of many of the configuration parameters, such as the search order between *TLS_CAFile* and *TLS_CAPath*, depends on the TLS library Radiator uses. This is typically OpenSSL but can be LibreSSL or any other TLS library that works with Net::SSLeay Perl module. For more information about Net::SSLeay and OpenSSL, see Section 2.1.6. Net::SSLeay and OpenSSL on page 5.

Important

System certificate store is not used. The TLS configuration parameters must cover all CAs, CRLs and other settings that are needed.

The default values for the TLS configuration parameters are the same for all configuration clauses unless otherwise stated.

3.11.1. TLS_Protocols

TLS_Protocols forces SSL or TLS for the configuration clause and specifies a comma-separated list of SSL and TLS protocols that are permissible for SSL and TLS connections. Currently, the supported SSL and TLS protocols are:

- SSLv3
- TLSv1
- TLSv1.1
- TLSv1.2
- TLSv1.3

CAUTION

TLSv1.3 is not enabled by default with Radiator yet. Testing reports are welcome. Net::SSLeay 1.83 or later is required if you use Radiator with SSL/TLS library that has TLSv1.3 enabled. Net::SSLeay 1.92 or later is recommended.

When set, *TLS_Protocols* overrides *UseSSL* and *UseTLS*. Otherwise *UseSSL* and *UseTLS* control the allowed protocols, if set. *TLS_Protocols* is not set by default.

Here is an example of using *TLS_Protocols*:

```
# Allow connections with these TLS versions only
TLS_Protocols TLSv1.1, TLSv1.2
```

Note

SSLv3 is obsolete and may not be supported by the TLS library Radiator uses. Use of SSLv3 is strongly discouraged but it is available for communicating with legacy SSLv3 peers.

3.11.2. TLS_CAFile

When TLS is enabled, this parameter specifies the name of a file containing Certificate Authority (CA) root certificates that may be required to validate TLS peer certificates. The certificates must be in PEM format. The file can contain several root certificates for one or more Certificate Authorities. Radiator looks for root certificates for RadSec connections in *`TLS_CAFile`* then in *`TLS_CAPath`*, so there usually is no need to set both.

3.11.3. TLS_CAPath

When TLS is enabled, this parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS peer certificates. Radiator looks for root certificates in *`TLS_CAFile`* then in *`TLS_CAPath`*, so there usually is no need to set the both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files.

Setting up CAPath directory for certificates and CRLs

The CA certificates must be in PEM format, one per file. When a certificate is needed, it is looked up using a special file name. Similarly lookup is done for Certificate Revocation List (CRL) files when certificate revocation lists are enabled. CRL files must also be in PEM format. Recent OpenSSL versions have a built-in command **`rehash`** that creates a symbolic link or a copy of file with the special file name. OpenSSL also comes with an separate tool called **`c_rehash`** that does the similar job. LibreSSL has a built-in command **`certhash`** to create symbolic links.

Here's an example of a directory with a file containing two CA certificates, a CRL file and a file with a single CA certificate. Note that no symbolic link is created when the file with two CA certificates is processed. The certificates in it must be separate files or otherwise the certificates in `cafile.pem` can not be used.

```
% ls -la
drwxr-xr-x   5 mikem  staff   160 Dec 20 13:09 .
drwxr-xr-x  97 mikem  staff  3104 Dec 20 12:59 ..
-rw-r--r--   1 mikem  staff  9957 Dec 20 13:09 cas.pem
-rw-r--r--   1 mikem  staff  2383 Dec 20 13:02 root-CA-crl.pem
-rw-r--r--   1 mikem  staff  4992 Dec 20 12:59 root-CA-crt.pem

% openssl version
OpenSSL 3.0.5 5 Jul 2022 (Library: OpenSSL 3.0.5 5 Jul 2022)

% openssl rehash -v .
Doing .
rehash: warning: skipping cas.pem,it does not contain exactly one certificate or CRL
link root-CA-crt.pem -> 322a67d3.0
link root-CA-crl.pem -> 322a67d3.r0

% ls -la
drwxr-xr-x   7 mikem  staff   224 Dec 20 13:17 .
drwxr-xr-x  97 mikem  staff  3104 Dec 20 12:59 ..
lrwxr-xr-x   1 mikem  staff    15 Dec 20 13:17 322a67d3.0 -> root-CA-crt.pem
```

```
lrwxr-xr-x   1 mikem  staff    15 Dec 20 13:17 322a67d3.r0 -> root-CA-crl.pem
-rw-r--r--   1 mikem  staff  9957 Dec 20 13:09 cas.pem
-rw-r--r--   1 mikem  staff  2383 Dec 20 13:02 root-CA-crl.pem
-rw-r--r--   1 mikem  staff  4992 Dec 20 12:59 root-CA-crt.pem
```

Note

Similar LibreSSL command is (dot is one of the command parameters): `openssl certhash -v .`

Testing indicates LibreSSL does not ignore files with multiple certificates but creates one symbolic link. For more information these OpenSSL and LibreSSL commands, see https://www.openssl.org/docs/manmaster/man1/c_rehash.html and https://man.openbsd.org/openssl.1

3.11.4. TLS_CertificateFile

When TLS is enabled, this parameter specifies the name of the file containing the certificate that Radiator uses. This certificate is sent to the TLS peer and validated by the peer during TLS setup. The certificate file must be in PEM or ASN1 format, depending on the setting of the *`TLS_CertificateType`* parameter. The certificate file can also contain the certificate's private key if the *`TLS_PrivateKeyFile`* parameter specifies the same file.

Note

RadSec clients expect the server certificate to have a common name (CN) the same as the RadSec servers DNS host name or address.

3.11.5. TLS_CertificateChainFile

When TLS is enabled, this parameter specifies the name of the file containing Radiator's certificate chain. The certificate chain is sent to the TLS peer and validated by the peer during TLS setup. The certificate chain must be in PEM format. This should be used alternatively or additionally to *`TLS_CertificateFile`* for explicitly constructing the certificate chain, which is sent to the peer in addition to the Radiator's own certificate.

3.11.6. TLS_CertificateType

When TLS is enabled, this optional parameter specifies the format of the *`TLS_CertificateFile`* and *`TLS_PrivateKeyFile`*. Permitted options are:

- **`PEM`**
- **`ASN1`**

The default value is **`ASN1`** which means binary DER format. Special formatting characters are supported.

3.11.7. TLS_PrivateKeyFile

When TLS is enabled, this optional parameter specifies the name of the file containing the private key of the certificate configured with *`TLS_CertificateFile`*. It is sometimes in the same file as the *`TLS_CertificateFile`*. Special formatting characters are supported.

The private key file must be in PEM or ASN1 format, depending on the setting of the *`TLS_CertificateType`* parameter.

Usually the private key is encrypted, use *TLS_PrivateKeyPassword* is the key to decrypt it.
For more information, see Section 3.11.4. TLS_CertificateFile on page 82 and Section 3.11.8. TLS_PrivateKeyPassword on page 83.

3.11.8. TLS_PrivateKeyPassword

When TLS is enabled, this optional parameter specifies the password that is to be used to decrypt the *TLS_PrivateKeyFile*. Special characters are permitted.

3.11.9. TLS_Ciphers

TLS_Ciphers specifies which subset of cipher suites is permissible for a connection when TLSv1.2 or earlier is negotiated. The parameter format is the cipher list format documented in OpenSSL manual for *openssl ciphers* command. The default value is **DEFAULT:!EXPORT:!LOW**.

```
# Exclude cipher suites using RC4 too
TLS_Ciphers DEFAULT:!EXPORT:!LOW:!RC4
```

This parameter affects only TLSv1.2 and earlier TLS versions. For the respective parameter for TLSv1.3, see Section 3.11.10. TLS_Ciphersuites on page 83

When SSL library supports security levels, a level can be set for each configuration clause together with cipher string. For more information about security levels, see Section 3.11.11. TLS_SecurityLevel on page 83

```
# System default level 2 is too strict for this module
TLS_Ciphers DEFAULT:!EXPORT:!LOW@SECLEVEL=1
```

3.11.10. TLS_Ciphersuites

TLS_Ciphersuites specifies which subset of TLSv1.3 cipher suites is permissible for a connection. The parameter format is documented in OpenSSL manual for *SSL_CTX_set_ciphersuites()* API function. For example: **TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256**. Usually there is no need to set this parameter. There is no default and the system default is used.

```
# This cipher suite is mandatory for TLSv1.3
TLS_Ciphersuites TLS_AES_128_GCM_SHA256
```

This parameter affects only TLSv1.3. For the respective parameter for TLSv1.2 and earlier TLS versions versions, see Section 3.11.9. TLS_Ciphers on page 83

3.11.11. TLS_SecurityLevel

TLS_SecurityLevel is an integer parameter that sets the SSL library security level for the enclosing clause. Security levels were added in OpenSSL 1.1.0 and are detailed in OpenSSL manual page `SSL_CTX_set_security_level`. There is no default and the system default is used. This parameter may be needed if you see unexpected TLS errors with older TLS peers.

Security level can also set together with ciphers in which case *TLS_SecurityLevel* is not needed. For more information, see Section 3.11.9. TLS_Ciphers on page 83

```
# The default pn this system, 2, is too strict for our peers,
# lower it for this Radiator module
TLS_SecurityLevel 1
```

3.11.12. TLS_RequireClientCert

This is an optional flag, which is available only for servers. It specifies whether the server clause requires each client to present a valid client certificate during TLS handshake or not. If the client certificate is not a valid certificate, the TLS handshake fails and the TCP or SCTP connection is disconnected. Certificate validity is determined by the root certificates that are configured for the server clause with *TLS_CAFile* or *TLS_CAPath*, and other TLS configuration parameters, such as *TLS_ExpectedPeerName* and *TLS_SubjectAltNameURI*.

Note

For compliance with RFC 6614, *TLS_RequireClientCert* is enabled by default for ServerRADSEC.

3.11.13. TLS_Verify

TLS_Verify is an optional string parameter which is available only for TLS clients. It specifies whether the client clause requires the server to present a valid server certificate during TLS handshake or not. The default is to require and verify the server certificate. If the server certificate is not a valid certificate, the TLS handshake fails and the TCP or SCTP connection is disconnected. Certificate validity is determined by the root certificates that are configured for the client clause with *TLS_CAFile* or *TLS_CAPath*, and other TLS configuration parameters, such as *TLS_ExpectedPeerName* and *TLS_SubjectAltNameURI*.

When *TLS_Verify* is set to value **none**, the server certificate is not verified. This may be useful during testing but it's not recommended for production configurations. Here is an example of using *TLS_Verify*:

```
<AuthBy REST>
    # TLS and other parameters

    # While testing, skip certificate validation
    TLS_Verify none

    # More parameters
</AuthBy>
```

3.11.14. TLS_RandomFile

When TLS is enabled, this optional parameter specifies the name of a file containing randomness. Usually there is no need to set this parameter.

3.11.15. TLS_DHFile

When TLS is enabled, this optional parameter specifies the name of the DH group. Usually there is no need to set this parameter, but it may be required if you are using ephemeral DH keys.

3.11.16. TLS_ECDH_Curve

When TLS is enabled, this optional parameter enables ephemeral EC keying by specifying the name of the elliptic curve to use. The default is to not enable ephemeral EC keying.

```
# Curves often have multiple aliases. Need OpenSSL name here.
# openssl ecparam -list_curves
TLS_ECDH_Curve prime256v1
```

3.11.17. TLS_CRLCheck

When TLS is enabled for a server, or TLS server has been configured to check client certificates with *TLS_RequireClientCert*, this optional flag parameter specifies that Certificate Revocation List (CRL) must be checked for revoked certificates during validation of the peer certificate.

3.11.18. TLS_CRLCheckUseDeltas

This optional flag parameter specifies if Delta Certificate Revocation List must be checked for revoked certificates in addition to base CRL. Currently delta CRL files are loaded with *TLS_CRLFile* parameter, similar to base CRL files.

CAUTION

TLS_CRLCheckUseDeltas is currently experimental.

Before enabling *TLS_CRLCheckUseDeltas*, note the following requirements and restrictions:

- *TLS_CRLCheck* must be enabled in Radiator configuration
- Both base and delta CRLs must use CRL v2 format
- Do not use delta CRL files without enabling *TLS_CRLCheckUseDeltas*
- OpenSSL indicates only one delta CRL file can be used
- Review OpenSSL notes about delta CRLs on OpenSSL manual page for X509_VERIFY_PARAM_set_flags. [https://www.openssl.org/docs/manmaster/man3/X509_VERIFY_PARAM_set_flags.html]
- Test that your base and delta CRL work when CRL files are updated or refreshed

Please contact Radiator support about success or possible problems there might be with delta CRLs.

3.11.19. TLS_CRLCheckAll

When TLS is enabled, this optional flag parameter enables CRL checks for the entire certificate chain. This is not enabled by default. The CRL files for the intermediate CAs must be found or the certificate check fails. For more information, see *TLS_CRLFile* on page 85.

Note

TLS_CRLCheck must be enabled for any certificate checks to happen.

3.11.20. TLS_CRLFile

This optional parameter specifies one or more CRL files that are used to check peer certificates for revocation when all the following conditions apply:

- TLS is enabled.
- TLS is configured to check peer certificates with *TLS_RequireClientCert*.
- CRL checking is enabled with *TLS_CRLCheck*.

The CRL files are also used when *TLS_CRLCheckAll* is enabled.

If the CRL file is not found or the CRL says the certificate has been revoked, TLS authentication fails with an error:

```
SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
```

To ease automation, CRLs may follow a file naming convention where each CRL file uses a special file name in *TLS_CAPath* directory. Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81. In this case you do not need to configure *TLS_CRLFile*.

If CRLs are not stored in the CAPath directory, one or more CRLs can be named with multiple *EAPTLS_CRLFile* parameters. The intended way CRL reloading works is this: Each CRL file named with *TLS_CRLFile* will be automatically reloaded and reread at the start of each new TLS session if the modification date of the named CRL file has changed since the last time it was loaded. If the CRL for a particular issuer changes, it is sufficient to replace the existing CRL file with the newer version and Radiator will reload the new CRL when required.

Tip

Operating system wildcards are supported, so you can name multiple CRLs with a single wildcard like:

```
TLS_CRLFile %D/crls/revocations-*.pem
```

3.11.21. TLS_PolicyOID

When a TLS peer presents a certificate, this optional parameter enables the certificate policy checking and specifies one or more policy OIDs that must be present in the certificate path. It sets the 'require explicit policy' flag as defined in RFC3280. Using this parameter requires Perl `Net::SSLeay` 1.37 module or later. This parameter may be used for additional certificate validity checks, for example, with RadSec.

When multiple TLS_PolicyOID parameters are configured, the peer certificate needs to match only one of the configured OIDs, not all of them.

```
# Require just one policy
TLS_PolicyOID 1.3.6.1.4.1.9048.33.2
```

3.11.22. TLS_ExpectedPeerName

When a TLS peer presents a certificate, this optional parameter specifies a regular expression pattern that is required to match the Subject in the peer certificate.

The default value for servers, such as ServerRADSEC, is **.+** which means to accept any Subject.

Different configuration clauses have different defaults for certificate validation. See the documentation of the specific configuration clause, such as *<AuthBy RADSEC>*, for the details.

Here is an example of using *TLS_ExpectedPeerName*:

```
# Accept certificates with CN ending in .xyz.com
TLS_ExpectedPeerName CN=.*\.xyz\.com
```

3.11.23. TLS_SubjectAltNameURI

When a TLS peer presents a client certificate, this optional parameter specifies a regular expression pattern that must match against at least one *subjectAltName* of type URI in the peer certificate.

There is no default value and no *subjectAltName* checks are done.

Different configuration clauses have different defaults for certificate validation. See the documentation of the specific configuration clause, such as *<AuthBy RADSEC>*, for the details.

Here is an example of using *TLS_SubjectAltNameURI*:

```
# Accept certificates that have a subjectAltName type URI that
# ends in open.com.au:
TLS_SubjectAltNameURI .*open.com.au
```

3.11.24. TLS_SNIHostname

TLS_SNIHostname enables the TLS Server Name Indication (SNI) with the specified hostname at the start of the TLS handshake. Defaults to not set and no SNI is sent. Special formatting characters are supported. *TLS_SNIHostname* is intended for use by TLS clients, such as *<AuthBy RADSEC>*.

When the parameter is set to an empty value, SNI is not enabled. This can be useful with *<AuthBy RADSEC>* to suppress SNI for a *<Host ...>* clause while allowing the other clauses to use the default SNI.

```
TLS_SNIHostname radsec.example.com
```

3.11.25. TLS_SubjectAltNameDNS

This optional parameter is used only by TLS clients. When a TLS server presents a certificate, this defines an FQDN that matches against a *subjectAltNameDNS* of type DNS in that certificate. Currently, this is not a regular expression but this is subject to change.

By default, the TLS client checks if the server certificate contains a *subjectAltName* extension of type IPADDR or DNS that matches the Host name used to connect to the server. When *subjectAltNameDNS* is configured, type DNS values are checked against the value configured with *subjectAltNameDNS*. This parameter has no default value.

Here is an example of using *TLS_SubjectAltNameDNS*:

```
# Host is set to an IP address so we set the expected name here
TLS_SubjectAltNameDNS test.server.open.com.au
```

3.11.26. TLS_CertificateFingerprint

You can require that the peer matches one of a specified set of signatures with *TLS_CertificateFingerprint*. When a TLS peer presents a certificate, this optional parameter specifies one or more fingerprints, one of which must match the fingerprint of the peer certificate. The format is **algorithm:fingerprint**. No fingerprint checks are done by default. Using this parameter requires Net::SSLeay 1.37 or later.

Here is an example of using *TLS_CertificateFingerprint*:

```
TLS_CertificateFingerprint \
    sha-1:8E:94:50:0E:2F:D6:DE:16:1D:84:76:FE:2F:14:33:2D:AC:57:04:FF
TLS_CertificateFingerprint \
    sha-1:E1:2D:53:2B:7C:6B:8A:29:A2:76:C8:64:36:0B:08:4B:7A:F1:9E:9D
TLS_CertificateFingerprint \
    sha-256:EC:14:77:FA:33:AD:2C:20:FF:D2:C8:1C:46:31:73:04:28:9E:ED:\
           12:D7:8E:79:A0:24:C0:DE:0B:88:A9:DB:3C
TLS_CertificateFingerprint md5:2A:2D:F1:44:40:81:22:D4:60:6D:9A:B0:F4:BF:DD:24
```

3.11.27. TLS_CertificateVerifyHook

This optional parameter specifies a Perl function that will be called for a custom verification of the peer certificate. It is passed the certificate, and various other details, and returns 0 for verification success, a new verification result code or < 0 for a failure, or an undefined value to continue Radiator builtin certificate verification.

Note

This parameter is currently available only for servers such as *ServerRADSEC*.

The function is passed the following arguments:

- *$_[0]*: *$hostname*, for servers, such as *ServerRADSEC*, the peer's IP address
- *$_[1]*: *$x509_store_ctx*, the OpenSSL store context (you can pass this to Net::SSLeay::X509_STORE_CTX_get_current_cert)
- *$_[2]*: *$cert*, the current certificate, result of Net::SSLeay::X509_STORE_CTX_get_current_cert($x509_store_ctx)
- *$_[3]*: *$subject_name*, the certificates subject name, result of Net::SSLeay::X509_get_subject_name($cert)
- *$_[4]*: *$subject*, the certificate subject, result of Net::SSLeay::X509_NAME_oneline($subject_name)
- *$_[5]*: *$object*, the Stream object created for this connection

```
<ServerRADSEC>
    ...
    # Accept immediately all certificates with O='OSC Demo Certificates'
    TLS_CertificateVerifyHook sub { my $subject = $_[4]; \
        my ($org) = $subject =~ /O=([^\/]*)/; \
        return unless $org eq 'OSC Demo Certificates'; return 0; }

</ServerRADSEC>
```

3.11.28. TLS_CertificateVerifyFailedHook

This optional parameter specifies a Perl function that will be called if verifying the peer certificate fails. It is passed the certificate (if present), and various other details.

The peer certificate *$cert* is not always present. An example of such case is a policy OID mismatch .

Note

This parameter is currently available only for servers such as *ServerRADSEC*.

TLS_CertificateVerifyFailedHook is passed the following arguments:

- *$_[0]: $verify_error*

  This is the OpenSSL store context verification code.

- *$_[1]: $x509_store_ctx*

This is the OpenSSSL store context.

- *$_[2]: $cert*

  This is the current certificate. May be undefined.

- *$_[3]: $subject_name*

  This is the certificate's subject name. Undefined when *$cert* is undefined.

- *$_[4]: $subject*

  This is the certificate subject. Undefined when *$cert* is undefined.

- *$_[5]: $object*

  This is the Stream object created for this connection.

TLS_CertificateVerifyFailedHook must return a single value. This value is used as an OpenSSL error code to set the verify result code as follows:

- > 0: Non-zero error code

  This is a new verification result code.

- 0

  This changes verification failure to verification success.

- < 0

  The verification process is immediately stopped with "verification failed" state.

- Undefined

  This is handled as an OpenSSL error *X509_V_ERR_APPLICATION_VERIFICATION*.

Here are examples of using *TLS_CertificateVerifyFailedHook*.

```
<ServerRADSEC>
    ...
    # Accept all certificates
    # TLS_CertificateVerifyFailedHook sub { return 0; }

    # Allow expired certificates: 10 is X509_V_ERR_CERT_HAS_EXPIRED
    TLS_CertificateVerifyFailedHook sub { \
        if ($_[0] == 10) { return 0; } else { return $_[0]; } }

</ServerRADSEC>
```

Note

This parameter may cause security issues if not used properly. Use it only in special cases.

3.11.29. TLS_SRVName

TLS_SRVName is intended for use by *<AuthBy RADSEC>* and *<AuthBy DNSROAM>* to specify a DNS SRV Name that is matched against possible SubjectAltName:SRV extensions in the peer certificate. If *TLS_SRVName* is specified and the peer certificate contains SubjectAltName:SRV extensions, none of which match *TLS_SRVName*, the certificate is not accepted.

Format is **_service._transport.name** (this is the same format SRV names appear in DNS records). Only service and name are matched.

```
TLS_SRVName _radsec._tcp.example.com
```

3.11.30. TLS_UseCADefaultLocations

This is a flag parameter. When set, the default locations are used for loading CA certificates. This is not set by default.

The default CA certificate directory is `certs/` in the default OpenSSL directory. Use *`SSL_CERT_DIR`* environment variable to override the default location. The default CA certificate file is `cert.pm` in the default OpenSSL directory. Use *`SSL_CERT_FILE`* environment variable to override the default file name.

3.11.31. TLS_CAPartialChain

This is a flag parameter. When set, only a partial CA certificate chain is required to validate TLS peer certificates. This functionality is required when restricting valid CA certificates to only a subset of CA hierarchy. This is not set by default.

3.11.32. TLS_NoClientCert

This is a flag parameter. When set, it turns off any client certificate checks and disables loading of any CA certificates for client certificate verification. This allows you to simplify configuration when client certificates are not requested with *`TLS_RequireClientCert`*.

3.11.33. TLS_OCSPCheck

This flag defines if Online Certificate Status must be checked for revoked certificates. Using this parameter requires `LWP::UserAgent` and `HTTP::Request` Perl modules.

Here are the requirements for using OCSP parameters:

- OpenSSL 1.0.0 or later
- Net::SSLeay 1.59 or later. Net::SSLeay 1.82 or later is recommended.

  If you use *`TLS_OCSPCheck`* or *`TLS_OCSPStapling`*, Net::SSLeay 1.82 or later is required.

3.11.34. TLS_OCSPStapling

This flag specifies that Online Certificate Status of our own certificate must be checked and returned to the peer as OCSP staple when running as a server. When running as a client, it requests OCSP staple. Using this parameter requires `LWP::UserAgent` and `HTTP::Request` Perl modules.

Using *`TLS_OCSPStapling`* does not require that *`TSL_OCSPCheck`* is set, but the same requirements apply when you use only *`TLS_OCSPStapling`*. See the requirements list in Section 3.11.33. TLS_OCSPCheck on page 90.

3.11.35. TLS_OCSPURI

This string defines OCSP URI that is used for all OCSP queries.

3.11.36. TLS_OCSPFailureBackoffTime

If the OCSP query fails, the query is not sent again to the failed OCSP URI. This integer defines the time for how long the failed OCSP URI is not contacted. The unit is seconds.

3.11.37. TLS_OCSPStrict

This optional flag parameter defines if Online Certificate Status must be checked successfully, otherwise the peer certificate is rejected.

When *`TLS_OCSPStrict`* is set, the certificate check must be successful in all phases. Here are some examples when the certificate check fails if *`TLS_OCSPStrict`* is set:

- *`TLS_OCSPURI`* is not set and the certificate does not have OCSP URI.
- OCSP responder is not responding.
- OCSP responder cannot be reached.

3.11.38. TLS_OCSPCacheTime

This integer defines the time for how long OCSP replies are saved in the cache. The unit is seconds. The default value is **`1200`**, that is 20 minutes.

3.11.39. TLS_OCSPCacheSize

This integer defines the maximum number of OCSP cache entries. The default value is **`1000`**.

3.11.40. TLS_KeylogFilename

Sets the name of file Radiator uses for logging TLS key material. TLS key log allows fully decrypting EAP and Stream SSL/TLS sessions, including those that have forward security enabled. TLS keylog should only be used for debugging to avoid security issues.

The keylog file is written in NSS Key Log Format, also known as SSLKEYLOGFILE. Tools, such as Wireshark, can read this file and fully decrypt TLS, including TLS sessions that have forward security enabled.

```
# Enable when debugging, remove when in production
TLS_KeylogFilename %L/radsec-keylog
```

DANGER

Keylog should only be used for debugging to avoid security issues.

3.11.41. TLS_TraceState

`TLS_TraceState` is a flag parameter that turns on TLS state tracing for TLS-based Stream modules, such as AuthBy RADSEC, ServerRADSEC and HTTP Client. This may be useful for additional debugging of TLS handshake processing. *`TLS_TraceState`* is not set by default.

Radiator 4.27 with `Net::SSLeay` 1.92 and later automatically enables TLS handshake message logging when log level is set to DEBUG or higher with Trace on page 136 or PacketTrace on page 105 parameter. When TLS message logging is enabled, TLS state tracing can also be enabled by setting *`TLS_TraceState`*. TLS state and messaging logging contain similar information and in most cases it's not useful have them both enabled.

Here is an example of using *`TLS_TraceState`*:

```
# Turn on TLS state tracing for this ServerRADSEC
TLS_TraceState
```

3.11.42. UseTLS

This optional parameter forces the use of TLS for a configuration clause. When this parameter is enabled, the *TLS_** parameters are available for use. For compliance with RFC 6614, it is enabled by default for RadSec clauses and *<AuthBy DNSROAM>*.

CAUTION

This option is obsolete. UseTLS enables TLSv1 only. For more information about the current preferred method of setting up TLS versions and parameters, see Section 3.11.1. TLS_Protocols on page 80 and Section 3.11.9. TLS_Ciphers on page 83.

Here is an example of using *UseTLS*:

```
UseTLS
```

3.11.43. UseSSL

This optional parameter forces the use of SSLv3, TLSv1.1, or TLSv1.2 for the configuration clause. The availability of TLS versions depends on if the TLS library supports them. When this parameter is enabled, the *TLS_** parameters become available for use.

CAUTION

UseSSL is obsolete and use of SSLv3 is strongly discouraged. For more information about the current preferred methods of setting up TLS versions and parameters, see Section 3.11.1. TLS_Protocols on page 80 and Section 3.11.9. TLS_Ciphers on page 83.

Here is an example of using *UseSSL*:

```
UseSSL
```

3.11.44. TLS_SessionResumption

This is a flag that allows you to enable or disable support for TLS session resumption.

CAUTION

This parameter is obsolete. TLS Session resumption is not currently supported for the stream modules.

3.11.45. TLS_SessionResumptionLimit

This is an integer that specifies the limit how long (in seconds) after the initial session that a TLS session can be resumed.

CAUTION

This parameter is obsolete. TLS Session resumption is not currently supported for the stream modules.

3.12. HTTP client configuration

Radiator's HTTP client support is implemented by `Radius::HTTPClient` module. It provides common configuration to specify HTTP backend servers, HTTP request methods, content encoding, TLS parameters and failure behaviour. HTTP connection establishment and response processing are done asynchronously. This allows non-blocking communication even with slow backends. `Radius::HTTPClient` module is built on top of Radiator Stream module and provides configuration similar to other stream configuration clauses such as RadSec.

The following configuration clauses utilise `Radius::HTTPClient`:

- <AuthBy REST> on page 279
- EAP-SIM, EAP-AKA and EAP-AKA' authentication clauses in Radiator SIM Pack.

All `HTTP::Client` configuration clauses support stream TLS parameters on page 80.

3.12.1. URL

This string parameter specifies an URL for a HTTP backend service. HTTPS is supported. Multiple *`URL`* parameters are supported. At least one *`URL`* must be defined. Optional comma separated priority value may follow an URL. If priority is not defined, it defaults to 1. When URL is configured to use **`https`** scheme, TLS client parameters must be configured. For more information, see Section 3.11. TLS configuration on page 80

```
# Service is reachable from two local ports
URL http://127.0.0.1:8888/radius
URL http://127.0.0.1:8889/radius

# Use port 443 and non-default priority
URL https://api.example.com/radius, 2
URL https://api.example.net/radius, 3
```

Radiator's special formatting characters, see Section 3.3. Special formatters on page 21, in *`URL`* are processed only when *`FormatURL`* is set. The supported formatters are documented for each clause that uses HTTP client parameters. See *`AuthBy REST`* on page 279 for an example.

3.12.2. FormatURL

This is an optional flag parameter. When this parameter is set, Radiator special character formatting is first applied to the current URL. The available formatters depend on the clause, see *`AuthBy REST`* on page 279 for an example.

3.12.3. RequestMethod

This string parameter defines the HTTP method to use. One of **`GET`**, **`DELETE`**, **`POST`**, **`PUT`** or **`PATCH`**. Defaults to **`GET`**.

```
# The service we use requires POST
RequestMethod POST
```

3.12.4. RequestContentType

This string parameter defines the Content-Type HTTP header value for **`POST`**, **`PUT`** and **`PATCH`** request methods. One of:

- **`application/json`**

- **application/x-www-form-urlencoded**
- **text/plain**
- **application/text**
- **application/plain**

If set to empty or to one of text or plain types, simple **keyword=value** encoding without escaping is used. Defaults to **application/x-www-form-urlencoded**

```
# The service we use requires POSTs in JSON format
RequestContentType application/json
```

3.12.5. RequestHeader

This string parameter defines an optional HTTP header fields to add to request. You can define one or more *RequestHeader* parameters. Format is header name, with an optional ':', followed by space and the header value. Defaults to not set.

```
# Add our custom HTTP header
RequestHeader User-Agent Radiator/4.xx AuthBy REST
```

3.12.6. HTTP_AuthenticationScheme

This optional string parameter defines the HTTP authentication scheme to use. Supported values is: **Basic**. Defaults to not set. When HTTP_AuthenticationHook on page 94 is configured, *HTTP_AuthenticationScheme* is ignored.

```
# API access requires HTTP authentication
HTTP_AuthenticationScheme Basic
HTTP_Username mikem
HTTP_Password fred
```

3.12.7. HTTP_AuthenticationHook

This optional parameter defines a Perl hook for API authentication. Defaults to not set. When *HTTP_AuthenticationHook* is configured, *HTTP_AuthenticationScheme* is ignored. The hook is passed the following arguments:

- Reference to the current `Radius::HTTPClient` derived clause
- Reference to an instance of `HTTP::Request` class representing the API request being currently built

The first parameter provides access to the configuration parameters, such as username and password shown in this example.

```
# API access requires HTTP authentication
HTTP_AuthenticationHook file:"%D/http-auth-hook.pl"
HTTP_Username mikem
HTTP_Password fred
```

3.12.8. HTTP_Username

This optional string parameter defines username for HTTP authentication when *HTTP_AuthenticationScheme* is configured. This parameter may also be accessed by hook configured with *HTTP_AuthenticationHook*. See *HTTP_AuthenticationScheme* on page 94 for an example.

3.12.9. HTTP_Password

This optional string parameter defines password for HTTP authentication when *HTTP_AuthenticationScheme* is configured. This parameter may also be accessed by hook configured with *HTTP_AuthenticationHook*. See *HTTP_AuthenticationScheme* on page 94 for an example.

3.12.10. HTTP_Version

This optional string parameter defines the HTTP version to use. Supported values are: **1.0** and **1.1**. Defaults to **1.1**.

```
# API access requires HTTP/1.0
HTTP_Version 1.0
```

3.12.11. NoreplyTimeout

This optional integer parameter defines time in seconds to wait for response. Defaults to 2 seconds. This example shows all customised failure related parameters.

```
# Time related values are in seconds.
NoreplyTimeout 5
FailureBackoffTime 60
MaxFailedRequests 3
MaxFailedGraceTime 10
```

3.12.12. FailureBackoffTime

This optional parameter sets the backoff time in seconds for failed backends. During the backoff time the backend is not used. Backend is considered failed after *MaxFailedRequests* on page 95 consecutive failures. Backend failure occurs when a backend does not respond or HTTP response status code indicates a failure. Defaults to not set. See *NoreplyTimeout* on page 95 for an example.

3.12.13. MaxFailedRequests

This optional parameter sets the number of consecutive failures before *FailureBackoffTime* is triggered. Defaults to not set. See *NoreplyTimeout* on page 95 for an example.

3.12.14. MaxFailedGraceTime

MaxFailedGraceTime specifies the time period (in seconds) over which *MaxFailedRequests* failures will cause the backend to be be assumed to be failed. Defaults to not set. The default value lets the HTTP client to choose a reasonable value which is typically very large. Very short values require a high number of failures to set target host as failed. This value should only be changed in special cases.

3.12.15. Connections

This integer parameter sets the initial number of connections per a backend URL. Defaults to 10.

```
# Backend allows only a small number of connections
Connections 1
MaxConnections 10
```

3.12.16. MaxConnections

This integer parameter sets the maximum number of connections per a backend URL. Defaults to 100. See *Connections* on page 95 for an example.

3.12.17. ReconnectTimeout

This optional parameter specifies the number of seconds to wait before attempting to reconnect a failed, dropped or disconnected connection. It also specifies the timeout for the initial connect.

3.12.18. ConnectOnDemand

This optional flag parameter tells Radiator not to connect to the server as soon as possible, but to wait until a message must be sent to that server. After the connection has been established and message has been delivered, the connection remains as long as possible. If the connection is lost for any reason, it is only re-established when and if there is another message to be sent.

3.12.19. MapResponseHook

This optional parameter defines a Perl hook for pre-processing HTTP server responses. Defaults to not set. The hook is passed the following arguments:

- Reference to a hash with server response and response related information

MapResponseHook has full access to a complete server response. This allows the hook to do any local modifications that are needed.

```
# Do some pre-processing before passing response to upper layers.
MapResponseHook file:"%D/http-map-response-hook.pl"
```

Here's an example of how to update a decoded JSON array response to a format described by *AuthBy REST* on page 279.

```
# Replace an array reference with the array's first member
# Received JSON: [ { "a1": 1 }, { "b2": 2 }]
# AuthBy REST needs just the first object, the one with key "a1"
<AuthBy REST>
  # Parameters, location within AuthBy does not matter
  MapResponseHook sub { $_[0]->{server_response} = $_[0]->{server_response}->[0]; }
  # More parameters
</AuthBy>
```

3.12.20. LocalAddress and LocalPort

These parameters control the address and optionally the port number used for the client source port, although this is usually not necessary. *LocalPort* is a string, it can be a port number or name. It binds the local port if *LocalAddress* is defined. If *LocalPort* is not specified or if it is set to **0**, a port number is allocated in the usual way.

When SCTP multihoming is supported, multiple comma separated addresses can be configured. All addresses defined with *LocalAddress* must be either IPv4 or IPv6 addresses.

```
LocalAddress 203.63.154.29
LocalPort 12345
```

3.12.21. MaxBufferSize

This optional advanced parameter specifies the maximum number of octets that are output and input buffered by Radiator's Stream modules. This advanced parameter usually does not need adjusting.

3.12.22. Debug

This optional flag parameter enables debug logging of both HTTP requests and responses. Debug logging is done with *Trace* level **5 EXTRA_DEBUG**. The example below shows how to enable HTTP request and response logging with *Trace* level 5.

```
# Enable global extra debugging
Trace 5

# AuthBy REST is a HTTP::Client
<AuthBy REST>
    Debug
    # Other parameters
</AuthBy>
```

3.13. Gossip configuration

This section lists the parameters that are common to all Gossip modules.

3.13.1. Identifier

All Gossip backends, such as GossipRedis and GossipUDP, must be configured with Identifier. Identifier will be required in later releases to reference the chosen Gossip instance.

```
# Modules that require GossipRedis can refer to us with this Identifier
Identifier gossip-redis
```

3.13.2. Debug

Debug configuration parameter is available for all Gossip implementations. This optional flag enables debugging within the Radius::Gossip and its derived modules. It enables the logging of the details of all messages sent and received when Trace is set to 4 or higher. This is not set by default.

Requires Perl Data::Printer module.

```
# See all message details.
Debug
```

3.13.3. Secret

Secret is an optional parameter that enables encryption and decryption of Gossip messages. *Secret* does not store the key in an encrypted format and using *EncryptedSecret* is preferred. For more information, see Section 3.13.4. EncryptedSecret on page 98.

The format is *n,value* where:

- *n* is key index. The valid values are from **1** and **65535**. The key index cannot be **0**, it is a reserved value and an error is logged if **0** is used.
- *value* is the key

Using *Secret* requires the following Perl modules:

- Crypt::GCM
- Crypt::Rijndael

CAUTION

If none of the *Secret* parameters can be successfully parsed, Gossip does not work and logs an error.

Here is an example of using *Secret*:

```
<GossipUDP>
    # Secret also works with GossipREDIS clause
    Secret 1,fred1
    Secret 2,fred2
    Secret 3,fred3
</GossipUDP>
```

3.13.4. EncryptedSecret

EncryptedSecret is an optional parameter that enables encryption and decryption of Gossip messages. *EncryptedSecret* stores the key in an encrypted format and thus it is preferable to use it instead of *Secret*.

The format is *n,{method}encrypted-value* where:

- *n* is key index. The valid values are from **1** and **65535**. The key index cannot be **0**, it is a reserved value and an error is logged if **0** is used.
- *method* is the encryption method. Currently, there is one supported method, **rcrypt**
- *encrypted-value* is the key encrypted with the encryption method

Using *EncryptedSecret* requires the following Perl modules:

- Crypt::GCM
- Crypt::Rijndael

CAUTION

If none of the *EncryptedSecret* parameters can be successfully parsed and decrypted, Gossip does not work and logs an error.

Here is an example of using *EncryptedSecret*:

```
<GossipUDP>
    # EncryptedSecret also works with GossipREDIS clause
    EncryptedSecret 1,{rcrypt}OjJXcK4bIA8sJERMzD2R0/Gx
    EncryptedSecret 2,{rcrypt}GyqZa52CYTRwsFqYQvsHNWbc
    EncryptedSecret 3,{rcrypt}I1vIkmow6FgLJyg5/pMfOOv2
</GossipUDP>
```

3.14. <Client xxxxxx>

The beginning of a Client clause. The clause continues until </Client> is seen on a line. A Client clause specifies a RADIUS client that this server will listen to. Requests received from any client not named in a Client

clause in the configuration file will be silently ignored. The DEFAULT client (if defined) will handle requests from clients that are not defined elsewhere.

You must have a Client clause for every RADIUS client which your server is expected to serve, or else a DEFAULT Client. In each Client clause replace the xxxxxx with either the DNS name or the IP address of the host machine where the RADIUS client is running, or with MAC: and the MAC address of the client. IPv4 and IPv6 addresses are supported. IPv6 addresses can only be received if an IPv6 BindAddress has been specified (See Section 3.7.9. BindAddress on page 33). IPv4 and IPv6 CIDR address notation is permitted.

DNS names are resolved only once when the client instance is created during the configuration. MAC address of the client is looked up from Called-Station-Id RADIUS attribute.

Client clauses are also used by TACACS+. For more information, see Section 3.120. <ServerTACACSPLUS> on page 405.

If an incoming request is capable of matching multiple Client clauses, the clause is chosen as follows:

- If multiple Client clauses have an exact IP match (same IP address or hostname resolving to the same IP address), the last one listed in the configuration is chosen. The same applies between multiple clauses with the same CIDR value: the last one listed in the configuration is chosen.
- Exact match is chosen over CIDR match
- Longer prefix CIDR match is chosen over shorter prefix CIDR match
- CIDR match is chosen over MAC match
- MAC match is chosen over DEFAULT

Tip

IPv6 addresses are not required to be prefixed with ‘ipv6:’ with Radiator 4.13 or later.

In the following example, the radius server will only respond to requests received from either oscar.open.com.au or from IPv4 address 203.63.154.7 or IPv4 network 203.10.1.0/24 or from IPv6 address 2001:db8:100:f101:0:0:0:1 or from IPv6 network 2001:db8:100::/64 and each client has a different shared secret. MAC match and DEFAULT client are commented out and thus not used.

Note

For more information about addresses starting with `::ffff`, see Section 3.7.10. BindV6Only on page 33.

```
<Client oscar.open.com.au>
      Secret XG1gFty566
</Client>
<Client 203.63.154.7>
      # An IPv4 client
      Secret kj1fgkj77878&
</Client>
<Client 203.10.1.0/24>
      # An IPv4 class C address group
      Secret ljdfhjlsd
</Client>
```

```
<Client ::ffff:203.10.1.0/120>
      # See the note above
      Secret ljdfhjlsd
</Client>
<Client 2001:db8:100:f101:0:0:0:1>
      # An IPv6 client
      Secret pqr
</Client>
<Client 2001:db8:100::/64>
      # An IPv6 /64 sized network
      Secret pqr
</Client>

#<Client MAC:2a-1f-09-5a-25-2a>
#      # Client identified by its MAC address
#      Secret gshgs
#</Client>
## Handle all other clients with this secret
#<Client DEFAULT>
#      Secret xyzzy
#</Client>
```

Each Client clause can have a number of different parameters set, as described below.

Tip

If you are using an SQL database, you can list your clients in a RADCLIENTLIST table and use <ClientListSQL>. For more information, see Section 3.16. <ClientListSQL> on page 118 instead of listing them in your config file. This may be convenient, especially if you are using RAdmin to manage your RADIUS system.

3.14.1. Secret

This defines the shared secret that is used to encrypt and decrypt `User-Password` and some other less frequently used attributes. Shared secret is also used for RADIUS message integrity checking with the exception of Access-Request messages. You must define a shared secret for each Client, and it must match the secret configured into the client RADIUS software. There is no default. The secret can be any number of ASCII characters. Any ASCII character except newline is permitted, but it might be easier if you restrict yourself to the printable characters. For a reasonable level of security, the secret should be at least 16 characters, and a mixture of upper and lower case, digits and punctuation. You should not use just a single recognisable word.

```
# This better agree with the client at
# 10.20.30.40 or we won't understand them!
<Client 10.20.30.40>
      Secret 66+6obaFGkmRNs-R
</Client>
```

3.14.2. EncryptedSecret

This defines the shared secret that is used with messages exchanged with this client. You must define a shared secret for each Client, and it must match the secret configured into the client RADIUS software.

EncryptedSecret is in encrypted format and is preferred over *Secret*. See Section 3.14.1. Secret on page 100 for more about RADIUS shared secrets.

EncryptedSecret is currently experimental and will be documented later.

```
# This better agree with the client at
# 10.20.30.41 or we won't understand them!
<Client 10.20.30.41>
      EncryptedSecret {rcrypt}1F67Kx6WXHKHpOuRZHSaIJdI
</Client>
```

3.14.3. DynAuthSecret

This defines the shared secret that is used to encrypt RADIUS dynamic authorisation requests that are sent to this client. The default is the client's *Secret*.

```
# 10.20.30.40 has separate secret for dynauth requests
<Client 10.20.30.40>
      Secret 66+6obaFGkmRNs-R
      DynAuthSecret 7e4+674.4a614A1b
</Client>
```

3.14.4. EncryptedDynAuthSecret

This defines the shared secret that is used to encrypt RADIUS dynamic authorisation requests which are sent to this client. *EncryptedDynAuthSecret* is in encrypted format and is preferred over *DynAuthSecret*.

EncryptedDynAuthSecret is currently experimental and will be documented later.

```
# 10.20.30.41 has separate secret for dynauth requests
<Client 10.20.30.41>
      EncryptedSecret {rcrypt}1F67Kx6WXHKHpOuRZHSaIJdI
      EncryptedDynAuthSecret {rcrypt}oVYMAWI/UUYPj4zbo3l95lKk
</Client>
```

3.14.5. TACACSPLUSKey

This is a per-client TACACSPLUS key which is used as the TACACS+ key if there is no Key defined in the ServerTACACSPLUS clause. Do not use this with RADIUS requests. For more information,see Section 3.120.1. Key on page 408.

3.14.6. EncryptedTACACSPLUSKey

This is a per-client TACACSPLUS key which is used as the TACACS+ key. EncryptedTACACSPLUSKey is in encrypted format and is preferred over TACACSPLUSKey. Do not use it with RADIUS requests. For more information, see Section 3.120.1. Key on page 408.

EncryptedTACACSPLUSKey is currently experimental and will be fully documented later.

3.14.7. DefaultRealm

This optional parameter can be used to specify a default realm to use for requests that have a User-Name that does not include a realm. The realm can then be used to trigger a specific <Realm> or <Handler> clause. This is useful if you operate a number of NASs for different customer groups and where some or all of your customers log in without specifying a realm.

```
# Realmless logins to this NAS will be treated
# as if they are for realm open.com.au
<Client acc1.open.com.au>
      Secret ....
      DefaultRealm open.com.au
</Client>
<Realm open.com.au>
      .....
</Realm>
```

Tip

Under some circumstances, some NASs send requests with no User-Name (usually for administrative reports). In that case, DefaultRealm will not add a realm to the User-Name, nor create a User-Name.

3.14.8. DupInterval

If more than one RADIUS request is received with the same source and destination IP address, source port, RADIUS authenticator and RADIUS Identifier within DupInterval seconds, the 2nd and subsequent requests are deemed to be duplicates or retransmissions. If the earlier request has already been replied to, then that reply will be resent back to the NAS. Otherwise the duplicate request will be dropped. A value of 0 means duplicates and retransmissions are always accepted, which might not be very wise, except during testing. RFC 5080 recommends a value between 5 and 30 seconds. Default is 10 seconds, which will detect and ignore duplicates due to multiple transmission paths and common retransmission intervals. In general you should never need to worry about or set this parameter. Ignore it and accept the default.

```
# brian.open.com.au is being tested
<Client brian.open.com.au>
      Secret 666obaFGkmRNs666
      DupInterval 0
</Client>
```

3.14.9. RewriteUsername

This is an optional parameter. It enables you to alter the username in authentication and accounting requests. For more details and examples, see Section 8. Rewriting user names on page 484.

3.14.10. IdenticalClients

This optional parameter specifies a list of other clients that have an identical setup. You can use this parameter to avoid having to create separate Client clauses for lots of otherwise identical clients. The value is a list of client names or addresses, separated by white space or comma. Each client may be specified as an IP address (IPv4 or IPv6), a MAC address in the form MAC:aa-bb-cc-dd-ee-ff, or an IPv4 or IPv6 CIDR address in the form nn.nn.nn/nn. You can have any number of IdenticalClients lines.

Tip

When you use the Client-Id check item, it matches against the name or address in the <Client xxxxxx> line, but not against any of the values listed in IdenticalClients.

```
IdenticalClients 10.1.1.1 10.1.1.2 nas.mydomain.com
IdenticalClients 10.1.1.7 10.1.1.8 10.1.1.9
```

```
IdenticalClients 203.63.154.1 localhost
IdenticalClients MAC:11-22-33-44-55-66
IdenticalClients 203.10.1.0/24 220.10.0.0/16
IdenticalClients 2001:db8:22:1::/64 2001:db8:22:2::/64
```

3.14.11. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  PreHandlerHook is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

  PreHandlerHook specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

  If *EAP_LEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-MSCHAPv2

  If *EAP_PEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-GTC

  If *EAP_GTC_PAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  PreHandlerHook is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  PreHandlerHook is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  PreHandlerHook is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  PreHandlerHook is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's *PrehandlerHook* is run before ServerTACASPLUS's *PreHandlerHook*. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

`PreHandlerHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *`PreHandlerHook`*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.14.12. StatusServer

Normally, when a Status-Server request is received, Radiator replies with some statistics including the total number of requests handled, the current request rate and so on. You can control Status-Server response by setting *`StatusServer`* to one of the following values:

- **`off`**

  Status-Server requests are ignored.

- **`minimal`**

  Reply without any attributes.

- **`uptime`**

  Reply without server uptime and, in case of RADIUS/TCP and RADIUS/TLS (RadSec), connection uptime .

- **`default`**

  Reply with statistics.

3.14.13. StatusServerShowClientDetails

Normally, when a Status-Server request is received, Radiator will reply with some statistics including the total number of requests handled, the current request rate etc. When you set the optional StatusServerShowClientDetails for a Client, the reply to Status-Server will also include details about that Client. This can result in a lengthy reply packet. The default is not to send the additional Client details for any Clients.

```
<Client xxxxxx>
      # Show stats about this client in Server-Status replies
      StatusServerShowClientDetails
      Secret xxxxxx
      ....
</Client>
```

Tip

You can test sending Status-Server requests with `radpwtst -noacct -noauth -trace 4 -status`

3.14.14. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It can also be used in Client-Identifier matches with Handlers:

```
<Client 10.1.2.3>
      Identifier        www-proxy
      Secret            mysecret
</Client>

# www-proxy
<Handler Client-Identifier=www-proxy>
      <AuthBy FILE>
            Filename    %D/www-proxy-users
      </AuthBy>
</Handler>
```

3.14.15. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with *`PacketTrace`* set off or **`0`**.

`PacketTrace` is available for the following clauses:

- *`Client`*
- *`Handler`*
- *`Realm`*
- *`AuthBy`*
- *`ServerDIAMETER`*
- *`ServerRADSEC`*
- *`ServerTACACSPLUS`*

Here is an example of using *`PacketTrace`*:

```
# Debug any packets that pass through here
PacketTrace
```

3.14.16. DefaultReply

This is similar to Section 3.14.20. AddToReply on page 106 except it adds attributes to an Access-Accept only if there would otherwise be no reply attributes. *`StripFromReply`* does never remove any attributes added by *`DefaultReply`*. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. You can use any of the special % formats in the attribute values. There is no default.

Although this parameter can be used in any AuthBy method, it is most useful in methods like *`<AuthBy UNIX>`* and *`<AuthBy SYSTEM>`*, which do not have a way of specifying per-user reply items. In other AuthBy methods you can also very easily set up a standard set of reply items for all users, yet you can still override reply items on a per-user basis.

```
# If the user had no reply items set some
```

```
DefaultReply Service-Type=Framed,Framed-Protocol=PPP
```

3.14.17. StripFromReply

Strips the named attributes from Access-Accepts before replying to the originating client. The value is a comma separated list of RADIUS attribute names. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. There is no default. This is useful, for example, with AuthBy RADIUS to prevent downstream RADIUS servers sending attributes you do not like back to your NAS.

```
# Remove dangerous attributes from the reply
StripFromReply Framed-IP-Netmask,Framed-Compression
```

3.14.18. AllowInReply

This optional parameter is the complement to *`StripFromReply`*: It specifies the only attributes that are permitted in an Access-Accept. It is useful, for example, to limit the attributes that are passed back to the NAS from a proxy server. This way you can prevent downstream customer RADIUS servers from sending back illegal or troublesome attributes to your NAS.

AllowInReply does not prevent other attributes being added locally by DefaultReply, AddToReply and AddToReplyIfNotExist.

```
# Only permit a limited set of reply attributes.
AllowInReply Session-Timeout, Framed-IP-Address
```

3.14.19. AllowInReject

This optional parameter specifies the only attributes that are permitted in an Access-Reject. This can be useful in Handlers with multiple AuthBys where the attributes added before a rejecting AuthBy need to be stripped from the resulting Access-Reject.

AllowInReject is not set by default and does not remove anything from Access-Rejects.

```
# Only permit a limited set of attributes in a reject.
AllowInReject Message-Authenticator, EAP-Message
```

3.14.20. AddToReply

Adds attributes to reply packets. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. You can use any of the special % formats in the attribute values. There is no default. AddToReply adds attributes to replies to all types of request that are handled by this clause.

Although this parameter can be used in any AuthBy method, it is most useful in methods like AuthBy UNIX, which don’t have a way of specifying per-user reply items.

```
# Append some necessary attributes for our pops
AddToReply cisco-avpair="ip:addr_pool=mypool"
```

3.14.21. AddToReplyIfNotExist

This is similar to AddToReply, but only adds an attribute to a reply if and only if it is not already present in the reply. Therefore, it can be used to add, but not override a reply attribute. This is contributed by Vincent Gillet <vgi@oleane.net>.

3.14.22. NoIgnoreDuplicates

This optional parameter specifies one or more RADIUS packet types where duplicates are not ignored. NoIgnoreDuplicates is a comma or space separated list of request types, such as:

- Access-Request
- Accounting-Request
- Status-Server

By default any request with an identical identifier received from the same NAS within the DupInterval period will be ignored or the previous reply retransmitted. If the request type is specified in NoIgnoreDuplicates, it will not be ignored, irrespective of the time since reception of a previous copy.

```
# Always handle dups of Accounting-Request packets
NoIgnoreDuplicates Accounting-Request
```

3.14.23. StripFromRequest

Strips the named attributes from the request before passing it to any Handlers or Realms. The value is a comma separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.14.24. AddToRequest

Adds attributes to the request before passing it to any Handlers or Realms. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.14.25. AddToRequestIfNotExist

Adds attributes to the request before passing it to any Handlers or Realms. Unlike AddToRequest, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name if they are not there already
AddToRequestIfNotExist Calling-Station-Id=1,Login-IP-Host=%h
```

3.14.26. ClientHook

This hook will be called for each request after the request has been decoded but before any other per-Client processing is done. A reference to the current request is passed as the only argument.

3.14.27. RequireMessageAuthenticator

`Client` clause checks the value of any *`Message-Authenticator`* attribute in incoming EAP or other requests. An incorrect authenticator causes the request to be ignored.

The optional *RequireMessageAuthenticator* flag causes the clause to require a correct *Message-Authenticator* attribute to be present in all incoming requests that support *Message-Authenticator* attribute. Most of the request types support *Message-Authenticator*. Accounting requests and responses do not support *Message-Authenticator*.

This flag parameter is not set by default.

3.14.28. LimitProxyState

The optional *LimitProxyState* flag parameter causes the *Client* to require a *Message-Authenticator* attribute to be present when one or more *Proxy-State* attributes are received. Radius clients that are not proxies should never send *Proxy-State* attributes. Proxies should always include a *Message-Authenticator* with the messages they forward. Limiting *Proxy-State* to requests which also include *Message-Authenticator* allows the server to discard unexpected requests from non-proxy clients.

This flag parameter is not set by default. Consider this parameter only for clients that are not proxies and are not capable of sending *Message-Authenticator*. Consider *RequireMessageAuthenticator* on page 107 for proxies and other clients that support *Message-Authenticator*.

3.14.29. DynAuthPort

This defines a destination port to which RADIUS dynamic authorisation requests are sent. There is no default. This allows modules, such as *AuthBy DYNAUTH*, to provide a per request value that is not overwritten by this parameter. When set, a common value is **3799**.

```
# Certain vendor may default to non-standard port
<Client 172.3.2.2>
      Secret 666obaFGkmRNs666
      DynAuthPort 1700
</Client>
```

3.14.30. UseMessageAuthenticator

This flag parameter adds a Message-Authenticator attribute to dynauth requests sent to this Client. Disabled by default.

```
# This NAS requires Message-Authenticator for dynauth requests
<Client 172.3.2.3>
      Secret 666obaFGkmRNs666
      DynAuthPort 1700
      UseMessageAuthenticator
</Client>
```

3.14.31. VsaVendor

VsaVendor sets the NAS vendor name for VsaTranslateIn and VsaTranslateOut. The NAS vendor names correspond to Perl modules in Radiator's `/Radius/Nas/` directory. For more information about VSA translation, see Section 3.14.33. VsaTranslateIn on page 109.

3.14.32. VsaType

VsaVendor sets the NAS vendor type for VsaTranslateIn and VsaTranslateOut. Each NAS vendor module in Radiator can have multiple types, for example, IOS and IOS XE for Cisco. For more information about VSA translation, see Section 3.14.33. VsaTranslateIn on page 109.

3.14.33. VsaTranslateIn

Attributes in incoming and outgoing RADIUS messages can be translated to and from internal presentations. For example, different MAC address formats can be normalised for logging and values for reply attributes can now be set based on the *Client* or *<AuthBy RADIUS>* vendor type. The general format for *VsaTranslateIn* and *VsaTranslateOut* is:

VsaTranslateIn source_attr, dest_attr[, translation, strip|nostrip, extras]

- **source_attr** is the translation source attribute.
- **dest_attr** defines the name for the newly translated attribute.
- **translation** defaults to copy which does no modification. The available translation depend on the *VsaVendor* and *VsaType* configuration parameters.
- **strip** or **nostrip** defaults to **nostrip** for *VsaTranslateIn* and **strip** for *VsaTranslateOut*.
- **extras** depends on the translation, *VsaVendor* and *VsaType*.

Full example showing *VsaVenedor*, *VsaType*, *VsaTranslateIn*, *VsaTranslateOut*, and the related configuration parameters is in goodies/vsa-translate.cfg.

```
# Translate incoming MAC address to common internal format
# and copy the DNS address in a VSA
<Client 192.168.3.49>
    Identifier juniper
    Secret mysecret

    # These control how the translation is done
    VsaVendor Juniper
    VsaType junos

    # macaddr transforms MAC addresses to and from internal presentation
    VsaTranslateIn Unisphere-Pppoe-Description, OSC-Mac-Address, macaddr

    # Convert internal presentation to what this client expects
    VsaTranslateOut OSC-DNS-Address1, Unisphere-Primary-Dns
</Client>
```

3.14.34. VsaTranslateOut

VsaTranslateOut an optional translation of attributes sent by Radiator to RADIUS clients. See VSATranslateIn on page 109 for more information.

3.14.35. NasType

This optional parameter specifies the vendor type of this Client. It is required if you want Radiator to directly query the NAS to check on simultaneous sessions.

You can specify the maximum number of sessions allowable for a single user with the Simultaneous-Use check item, or for all the users in a Realm with the *MaxSessions* parameter in <Realm> or <Handler> clauses. In either case, during authentication, Radiator first checks its Session Database to see if the user's session count is exceeded. For more information, see Section 3.18. <SessionDatabase SQL> on page 122 and Section 3.20. <SessionDatabase DBM> on page 130. Since this count can be inaccurate in the face of NAS reboots, lost packets etc. Radiator can also double check the count by interrogating the NAS directly (you enable this by specifying *NasType* in the Client clause).

If you specify **unknown** or do not specify any value at all, Radiator will never try to contact the NAS to check the user's sessions, and it will always assume that the sessions it thinks are present are correct. If you specify **ignore**, Radiator will never try to contact the NAS to check the users sessions, and it will always assume that there are no multiple sessions.

Tip

If Radiator detects problems or time-outs when using finger to verify simultaneous connections, it assumes that the user is still online (i.e. it assumes that the Session Database is correct).

Tip

You can also use *NasType* as a check item, to confirm that a request came from a client with a specific *NasType*.

The allowable values for *NasType* are:

Table 8. Allowable values of NasType, and their NAS query method

NasType	Method used to connect to NAS
Livingston	SNMP
Portslave	Finger
PortslaveLinux	Finger. For use with Portslave running on a Linux host, understands Linux finger format.
PortslaveMoxa	Finger, requires ctlportslave to be installed as fingerd on the target Linux host. Supports Linux running Portslave and a Moxa multiport.
Cisco	SNMP
CiscoVPDN	SNMP, detects users terminated on a l2tp, pptp or l2f tunnel.
Colubris	SNMP
Ascend	Finger
AscendSNMP	SNMP
Computone	Finger
Cyclades	SNMP
Hiper	SNMP
NomadixSNMP	SNMP
Redback	SNMP
Shiva	Finger
TotalControl	pmwho
TotalControlSNMP	SNMP
Bay, Bay5399SNMP, Bay8000SNMP	SNMP

NasType	Method used to connect to NAS
Bay4000SNMP	SNMP
BayFinger	Finger
Tigris, TigrisNew	SNMP for new version of the Tigris MIB (i.e. firmware revision 10.1.4.14 or greater)
TigrisOld	SNMP for old versions of the Tigris MIB
NortelCVX1800	SNMP
Xyplex	Finger
Patton	SNMP
Portmaster3	SNMP
Portmaster4	pmwho For use with Portmaster 4's running ComOS 4.1 or later
Ping	Verifies a login by ICMP pinging the Framed-IP-Address of the dialup user. This is not foolproof if the IP address has been reallocated. Requires that Radiator be run with root permissions (on Unix).
ignore	Does not contact NAS under any circumstances. Always assumes that there are no multiple logins.
unknown	The default value. Does not connect to the NAS under any circumstances. Always assumes the Session Database is correct.
CiscoSessionMIB	SNMP, using the Session MIB available in Cisco IOS 12.2.15T and later.

Tip

You can add support for a new or custom *NasType*s by adding a suitably named module to the Radius/Nas directory. Your new module should implement the isOnline function. See the existing `Radius/Nas/*pm` modules for examples. If you do implement your own module, send us a copy so we may include it in future releases.

Radiator uses a number of global parameters to specify how to communicate with the NAS. See Section 3.7.22. SnmpgetProg on page 36, Section 3.7.24. FingerProg on page 37, Section 3.7.25. PmwhoProg on page 37, Section 3.7.26. LivingstonMIB on page 37, Section 3.7.27. LivingstonOffs on page 37, and Section 3.7.28. LivingstonHole on page 37.

```
# Make Radiator ask the NAS to confirm multiple logins.
# its a Total Control box
NasType TotalControl
```

3.14.36. SNMPCommunity

This optional parameter specifies the SNMP Community name to use to connect to the NAS when NasType uses SNMP. It is ignored for any other NasType. Defaults to public.

```
SNMPCommunity private
```

3.14.37. FramedGroup

This optional parameter acts similarly to Framed-Group reply items, but it applies to all Access-Requests authenticated by this clause. If FramedGroup is set and a matching FramedGroupBaseAddress is set in the Client from where the request came, then a Framed-IP-Address reply item is automatically calculated by adding the NASPort in the request to the FramedGroupBaseAddress specified by FramedGroup. For more information, see Section 3.14.38. FramedGroupBaseAddress on page 112.

Note

You can override the value of FramedGroup for a single user by setting a Framed-Group reply item for the user.

```
# Work out the users IP address from the first
# FramedGroupBaseAddress specified in out client
FramedGroup 0
```

3.14.38. FramedGroupBaseAddress

This optional parameter is used in conjunction with the Framed-Group reply attribute or the FramedGroup AuthBy parameter to automatically generate IP addresses for users logging in. It is ignored unless the user has a Framed-Group reply item, or unless their AuthBy clause contains a FramedGroup parameter. You can have as many FramedGroupBaseAddress items as you like.

You would only need to use this mechanism if you are using a NAS that is unable to choose IP addresses from an address pool, or if you want a more complicated address allocation policy than your NAS can support.

When a user logs in, Radiator can automatically choose an IP address for the user and return it to the NAS in a Framed-IP-Address reply attribute. To make this happen, you must specify one or more FramedGroupBaseAddress items in each Client clause, and you must specify a Framed-Group reply item for each user for whom you want address allocation. If the user is authenticated, Radiator will generate a Framed-IP-Address using Framed-Group reply item and the NAS-Port in the request. The Framed-Group in a user record selects the nth FramedGroupBaseAddress (0 based) from the Client they are logging in to, and NAS-Port is added to the last byte (modulo 255 or FramedGroupMaxPortsPerClassC) to generate a Framed-IP-Address.

Tip

Radiator also includes a much more sophisticated mechanism for allocating IP addresses from an SQL database or DHCP. For more information, see Section 3.53. <AuthBy DYNADDRESS> on page 241.

In the example below, if the user logs in on the Client at port 5, and their Framed-Group reply item is 1, they will be allocated a Framed-IP-Address of 10.0.1.6 (i.e. 10.0.1.1 + 5).

In the Radiator configuration file:

```
<Client ..>
      # This is the base address for Framed-Group = 0
      FramedGroupBaseAddress 10.0.0.1
      # This is the base address for Framed-Group = 1
      FramedGroupBaseAddress 10.0.1.1
```

```
      # This is the base address for Framed-Group = 2
      FramedGroupBaseAddress 10.0.2.1
      ....
</Client>
```

In the users file for each user you want to allocate an address for:

```
mikem       User-Password = "fred"
            Framed-Group = 1,
            Framed-Protocol = PPP,
            etc.
```

Alternatively, in an AuthBy clause:

```
<AuthBy whatever...>
      # This will cause all users authorised by this clause to get
      # an address allocated from the block starting 10.0.1.1,
      # unless overridden by a user-specific Framed-Group
      FramedGroup 1
      .....
</AuthBy>
```

3.14.39. FramedGroupMaxPortsPerClassC

This optional parameter defines the maximum number of ports that can be mapped to a class C or class B FramedGroupBaseAddress. The default is 255, which means that any address from 0 up to 255 in the 3rd or 4th octets will be permitted. It actually specifies the modulus for computing the 3rd and 4th octets of addresses calculated from FramedGroupBaseAddress. You might use this to limit the number of addresses used in each address block, or to prevent the allocation of the last address in a class C address block.

3.14.40. UseContentsForDuplicateDetection

This optional flag causes the Client to use an alternative duplicate detection algorithm. Normally, Client detects duplicate requests by looking at the source IP address, source IP port, the RADIUS request authenticator and the RADIUS request identifier of incoming requests. Requests with those identical attributes received in the last DupInterval seconds are deemed to be duplicates.

Note

This parameter is not needed in a ServerFarm environment when shared duplicate cache is used by the farm workers. See Section 3.7.55. DupCache on page 43.

In a ServerFarm environment, where the server farm is being used to load balance using, say, the EAPBALANCE module, the back end servers will receive requests (and potentially duplicate requests) from any of the front end servers in the server farm. This can defeat the normal duplicate detection algorithm. UseContentsForDuplicateDetection causes Client to look only at the contents of the request that would not be altered by passing through a proxy server, allowing duplicates to be reliably detected, even in a server farm. The packet contents used for duplicate detection when UseContentsForDuplicateDetection is set are:

- RADIUS authenticator
- User-Name
- Called-Station-Id

- Calling-Station-Id

It is essential that this parameter be defined in the Client clauses of back end servers of an EAPBALANCE Server Farm architecture, otherwise duplicate detection will not be performed correctly.

3.14.41. DynamicReply

This optional parameter specifies a reply item that will be eligible for run-time variable substitution. That means that you can use any of the % substitutions in Section 3.3. Special formatters on page 21 in that reply item. You can specify any number of DynamicReply lines, one for each reply item you want to do replacements on. Any packet-specific replacement values will come from the Access-Accept message being constructed, and not from the incoming Access-Request. That means that special characters like %n will not be replaced by the received User-Name, because User-Name is in the request, but not the reply.

In the following example, substitution is enabled for USR-IP-Input-Filter in an AuthBy clause. When a user authenticates, the %a in the filter will be replaced by the users IP Address, which makes the filter an anti-spoof filter.

```
<AuthBy whatever>
      ......
      UseAddressHint
      DynamicReply USR-IP-Input-Filter
</AuthBy>
```

In the users file:

```
DEFAULT User-Password = "UNIX"
      Framed-IP-Address = 255.255.255.254,
      Framed-Routing = None,
      Framed-IP-Netmask = 255.255.255.255,
      USR-IP-Input-Filter = "1 REJECT src-addr != %a;",
      Service-Type = Framed-User
```

Note

This parameter used to be called *`Dynamic`*. That name is still recognised as a synonym for *`DynamicReply`*.

3.14.42. IgnoreAcctSignature

If you define *`IgnoreAcctSignature`*, it prevents the server from checking the Authenticator field in requests received from this client. Contrary to its name, it applies to all message types and also prevents checking the *`Message-Authenticator`* attribute. This parameter is useful because some clients do not send Authenticators that conform to RADIUS RFCs.

By default, the server logs and ignores messages that do not have a correct Authenticator, or any messages that do not have a correct *`Message-Authenticator`* attribute. Regardless of the setting of this parameter, the server always sends a correctly computed Authenticator and *`Message-Authenticator`* attribute.

CAUTION

This parameter is seldom required with current RADIUS implementations. You should first check that the shared secret between Radiator and client is correct before enabling this parameter.

If you get bad authenticator log messages and the accounting requests are not being stored even though authentication as such does not fail, and you have checked that the shared secrets are correct, try enabling *IgnoreAccSignature*. The bad authenticator log message looks this:

```
Bad authenticator in request from <client name> (<nas identifier>)
```

If you get bad EAP Message-Authenticator log messages and you have checked that the shared secrets are correct, it is possible that the NAS is sending an incorrect implementation of Message-Authenticator. Try enabling *IgnoreAccSignature*. The bad EAP Message-Authenticator log message looks this:

```
Bad EAP Message-Authenticator
```

Tip

Some NASs have separate secrets for authentication and accounting requests.

```
# brian.open.com.au has a broken legacy NAS
<Client 10.20.30.40>
      Identifier brian.open.com.au
      Secret 666obaFGkmRNs666
      IgnoreAcctSignature
</Client>
```

3.14.43. LivingstonOffs

Specifies the value of where the last S port is before the one or two ports specified in LivingstonHole are skipped (usually 22 for US, 29 for Europe). This optional parameter is only used if you are using Simultaneous-Use with a NasType of Livingston in this Client clause. Defaults to the global value of LivingstonOffs, see Section 3.7.27. LivingstonOffs on page 37.

3.14.44. LivingstonHole

Specifies the value of the size of the hole in the port list (usually 1 for US, 2 for Europe) that occurs at LivingstonOffs. This optional parameter is only used if you are using Simultaneous-Use with a NasType of Livingston in this Client clause. Defaults to the global value of LivingstonOffs, see Section 3.7.28. LivingstonHole on page 37.

3.14.45. UseOldAscendPasswords

This optional parameter tells Radiator to decode all passwords received from this Client using the old style (non RFC compliant) method that Ascend used to use on some NASs. The symptom that might indicate a need for this parameter is that passwords longer than 16 characters are not decoded properly.

3.15. <ClientListLDAP>

This optional clause allows you to specify your RADIUS and TACACS+ clients in an LDAP database in addition to or instead of your Radiator configuration file. When Radiator starts up and receives a SIGHUP signal, it queries the LDAP database with the *SearchFilter*. The results of that query are used to add details of RADIUS Clients that Radiator responds to. One Client clause is created for each matching LDAP record found. *<ClientListLDAP>* fetches the LDAP attributes specified by the *ClientAttrDef* parameters, and uses them to set the parameters in each Client clause. You can have some client details in your Radiator configuration file and some in *<ClientListLDAP>* although this can be confusing to future administrators.

This clause supports all the common LDAP configuration parameters. For more information about the LDAP configuration parameters, see Section 3.9. LDAP configuration on page 53.

Tip

There is a sample LDAP schema compatible with the default behaviour of *<ClientListLDAP>* in `goodies/radiator-ldap.schema` in your Radiator distribution. There are some example LDAP records for this schema in `goodies/radiator-ldap.ldif`.

Tip

There is an example configuration file showing how to configure *<ClientListLDAP>* in `goodies/ldapradius.cfg` in your Radiator distribution.

3.15.1. BaseDN

This is the base DN, where searches are made. It is used in similar way as with all LDAP modules. For more information, see Section 3.9.1. BaseDN on page 53.

Special formatting characters are permitted.

This *BaseDN* use example is specifically for *<ClientListLDAP>*:

```
# Start looking here
BaseDN ou=RadiusClients, o=University of Michigan, c=US
```

3.15.2. SearchFilter

This parameter specifies the LDAP search filter that is used to find the LDAP records that contain Client data. The default value is `(objectclass=oscRadiusClient)`, which is compatible with the example schema provided in `goodies/radiator-ldap.schema` in your Radiator distribution. Special characters are supported.

This example finds oscRadiusClient LDAP objects, but only the ones for a specific location. It shows how you can use LDAP boolean expressions to select records from the LDAP database:

```
SearchFilter (&(objectclass=oscRadiusClient)(location=my_pop_1))
```

3.15.3. ClientAttrDef

This optional parameter specifies the name of an LDAP attribute to fetch, and the name of the Client parameter that it will be used for in the Client clause. The format is:

```
ClientAttrDef ldapattrname,clientparamname
```

where *ldapattrname* is the name of the LDAP attribute to fetch, and *clientparamname* is the name of the Client clause parameter. For more information, see Section 3.14. <Client xxxxxx> on page 98 There can be (and usually are) multiple *ClientAttrDef* parameters. If the specified *ldapattrname* is not present in a record, then the matching *clientparamname* will not be set and will assume its default value according to the normal behaviour of the Client clause.

If no *ClientAttrDef* lines are defined, defaults to the equivalent of the following, which is compatible with the example schema provided in `goodies/radiator-ldap.schema`. Note that not all these attributes are required in your LDAP database. The only ones that must be provided are for **`Name`** and **`Secret`**.

```
ClientAttrDef oscRadiusClientName,Name
ClientAttrDef oscRadiusSecret,Secret
ClientAttrDef oscRadiusInoreAcctSignature,IgnoreAcctSignature
ClientAttrDef oscRadiusDupInterval,DupInterval
ClientAttrDef oscRadiusNasType,NasType
ClientAttrDef oscRadiusSNMPCommunity,SNMPCommunity
ClientAttrDef oscRadiusLivingstonOffs,LivingstonOffs
ClientAttrDef oscRadiusLivingstonHole,LivingstonHole
ClientAttrDef oscRadiusFramedGroupBaseAddress,FramedGroupBaseAddress
ClientAttrDef oscRadiusFramedGroupMaxPortsPerClassC,FramedGroupMaxPortsPerClassC
ClientAttrDef oscRadiusFramedGroupPortOffset,FramedGroupPortOffset
ClientAttrDef oscRadiusRewriteUsername,RewriteUsername
ClientAttrDef oscRadiusUseOldAscendPasswords,UseOldAscendPasswords
ClientAttrDef oscRadiusStatusServerShowClientDetails,StatusServerShowClientDetails
ClientAttrDef oscRadiusPreHandlerHook,PreHandlerHook
ClientAttrDef oscRadiusPacketTrace,PacketTrace
ClientAttrDef oscRadiusIdenticalClients,IdenticalClients
ClientAttrDef oscRadiusNoIgnoreDuplicates,NoIgnoreDuplicates
ClientAttrDef oscRadiusDefaultReply,DefaultReply
ClientAttrDef oscRadiusFramedGroup,FramedGroup
ClientAttrDef oscRadiusStripFromReply,StripFromReply
ClientAttrDef oscRadiusAllowInReply,AllowInReply
ClientAttrDef oscRadiusAddToReply,AddToReply
ClientAttrDef oscRadiusAddToReplyIfNotExist,AddToReplyIfNotExist
ClientAttrDef oscRadiusDynamicReply,DynamicReply
ClientAttrDef oscRadiusStripfromRequest,StripfromRequest
ClientAttrDef oscRadiusAddToRequest,AddToRequest
ClientAttrDef oscRadiusAddToRequestIfNotExist,AddToRequestIfNotExist
ClientAttrDef oscRadiusDefaultRealm,DefaultRealm
ClientAttrDef oscRadiusIdentifier,Identifier
ClientAttrDef oscTacacsPlusKey,TACACSPLUSKey
```

3.15.4. RefreshPeriod

If this optional parameter is set to non-zero, it specifies the time period in seconds that ClientListLDAP will refresh the client list by rereading the database. If set to 0, then ClientListLDAP will only read the client list from the database at startup and on SIGHUP. Defaults to 0. The % formats are permitted.

```
# Reread the client list every hour
RefreshPeriod 3600
```

3.15.5. FarmWorkerSpacing

If this optional parameter is set to non-zero, it specifies the time in seconds for spacing out refresh done by server farm workers. Defaults to not set which causes all farm workers to refresh client list at the same moment. This parameter has only effect when both *RefreshPeriod* and global *FarmSize* on page 42 parameters are configured.

```
# Reread the client list every hour, use 30 second offset between each worker
RefreshPeriod 3600
FarmWorkerSpacing 30
```

3.15.6. PostSearchHook

This optional parameter allows you to define a Perl function that is is called after the LDAP search results have been received, and after Radiator has processed the attributes it is interested in. Hook authors can use LDAP library routines to extract other attributes and process them in any way.

PostSearchHook is called once for each LDAP result and allows changing client's parameters before it is instantiated.

PostSearchHook has the following arguments:

- Handle to the current ClientListLDAP object
- Reference to client Name. To change the name, set the reference to this argument
- Reference to a hash with client configuration parameter values collected from the entry.
- Search result entry

Here is an example of *PostSearchHook*:

```
# Do not add clients that have 'test' in their name
PostSearchHook sub {my $name = $_[1]; $$name = '' if $$name =~ 'test';}
```

Tip

You can change client Name as shown in the above example. If you set the name to empty string, Radiator will skip this client.

3.16. <ClientListSQL>

This optional clause allows you to specify your RADIUS and TACACS+ clients in an SQL database table in addition to (or instead of) your Radiator configuration file. When Radiator starts up (and when it receives a SIGHUP signal), it queries the SQL database with the *GetClientQuery* SQL query, and the results of that query are used to add details of RADIUS Clients that Radiator will respond to. If you wish, you can have some client details in your Radiator configuration file, and some in *ClientListSQL* (although this might be confusing to future administrators).

This clause supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

Tip

The example database schemas provided in the goodies directory of your Radiator distribution all include an example RADCLIENTLIST table that will work with *ClientListSQL*.

3.16.1. GetClientQuery

This parameter specifies the SQL query that is used to fetch client details from the SQL database specified by *DBSource*. The database can store all the same parameters that are used to configure a *<Client>* clause. For more information, see Section 3.14. <Client xxxxxx> on page 98. The recommend configuration with Radiator 4.24 and later is to use *ClientColumnDef* with *GetClientQuery*. For more information about ClientColumDef, see Section 3.16.2. ClientColumnDef on page 120.

The default *GetClientQuery* works with the sample database schemas provided in the goodies/ of your Radiator distribution. *GetClientQuery* defaults to:

```
select NASIDENTIFIER,SECRET,IGNOREACCTSIGNATURE,DUPINTERVAL,
            DEFAULTREALM,NASTYPE,SNMPCOMMUNITY,LIVINGSTONOFFS,
            LIVINGSTONHOLE,FRAMEDGROUPBASEADDRESS,
            FRAMEDGROUPMAXPORTSPERCLASSC,REWRITEUSERNAME,
            NOIGNOREDUPLICATES,PREHANDLERHOOK from RADCLIENTLIST
```

Your database table must include at least the first and second fields, which are the NAS name or IP address or MAC address and the shared secret. All the other fields are optional.

When *ClientColumnDef* is *not* configured, the other fields must occur in the given order. When they occur, they are used to initialise the *Client* parameter of the same name as shown above. The *FRAMEDGROUPBASEADDRESS* column may contain multiple comma-separated base addresses. The *PREHANDLERHOOK* column can contain either the text of a hook or a hook filename in the form '`file:/path/to/hook`'. You can customise the *GetClientQuery* select clause to have additional fields. If they are present in the result of *GetClientQuery*, they are used as described below. Field number 0 as the first field, so for example *Identifier*, field 14 has an index of 14, but is the 15th entry in the returned array.

- *Identifier* field as field 14
- *DefaultReply* as field 15
- *FramedGroup* as field 16
- *StripFromReply* as field 17
- *AllowInReply* as field 18
- *AddToReply* as field 19
- *AddToReplyIfNotExist* as field 20
- *DynamicReply* as field 21
- *AddToRequest* as field 22
- *StripFromRequest* as field 23
- *AddToRequestIfNotExist* as field 24
- *ClientHook* as field 25
- *UseContentsForDuplicateDetection* as field 26
- A comma-separated list of flag names as field 27. Each comma-separated name in the field is used to set a Client flag type parameter. For example, if field 27 has the value "**IgnoreAcctSignature,UseOldAscendPasswords,StatusServerShowClientDetails**", it sets the *IgnoreAcctSignature*, *UseOldAscendPasswords*, and *StatusServerShowClientDetails* flag parameters in the resulting *Client*.
- *TACACSPLUSKey* as field 28

Here is an example that fetches the required information and DefaultRealm:

```
# Our custom client table only has NAS identifier,
# shared secret and default realm in it:
GetClientQuery select NAME,SECRET,NULL,NULL,DREALM from CLIENTS
```

Here is the same example with *ClientColumnDef*:

```
# We do not need to pad with NULL columns
```

```
GetClientQuery select NAME,SECRET,DREALM from CLIENTS
ClientColumnDef 0, Name
ClientColumnDef 1, Secret
ClientColumnDef 2, DefaultRealm
```

3.16.2. ClientColumnDef

This optional parameter allows you to specify an alternate mapping between the fields returned by *GetClientQuery* and the parameters used to define a Client. If *ClientColumnDef* is not specified, the mapping is the default as described in Section 3.16.1. GetClientQuery on page 118

The format of *ClientColumnDef* is:

```
ClientColumnDef n,clientparamname
```

where **n** is the column number of the fields as returned by *GetClientQuery* (starting at **0**), and *clientparamname* is the name of the Client clause parameter or special value **GENERIC**.

For more information about Client parameters, see Section 3.14. <Client xxxxxx> on page 98. There must be at least two, and usually there are multiple, *ClientColumnDef* parameters. The only ones that must be provided are for **Name** and **Secret**. If the specified column has NULL value, then the matching *clientparamname* will not be set and will assume its default value according to the normal behaviour of the Client clause.

Special *clientparamname* **GENERIC** indicates that the column is a list of comma separated *clientparamname=value* pairs. For examples of this format, see AuthColumnDef in AuthBy SQL on page 192

Here is a minimal example. It uses the sample database schema provided in the `goodies/` of your Radiator distribution.

```
GetClientQuery select NASIDENTIFIER, SECRET from RADCLIENTLIST
ClientColumnDef 0, Name
ClientColumnDef 1, Secret
```

Here is an example that sets *Identifier* parameter, see Section 3.14.14. Identifier on page 105. Column **3** must contain *name=value* pairs, for example: **StatusServer=minimal,RequireMessageAuthenticator=1**.

```
GetClientQuery select NASIDENTIFIER, SECRET, CLIENT_IDENTIFIER, GENERIC \
                      from RADCLIENTLIST
ClientColumnDef 0, Name
ClientColumnDef 1, Secret
ClientColumnDef 2, Identifier
ClientColumnDef 3, GENERIC
```

3.16.3. RefreshPeriod

If this optional parameter is set to non-zero, it specifies the time period in seconds that ClientListSQL will refresh its client list by rereading the database. If set to 0, then ClientListSQL will only read the client list from the database once at startup and on SIGHUP. Defaults to 0. The % formats are permitted.

When the RefreshPeriod expires and the list of clients is read from the SQL database, any Clients previously created by this ClientList are cleared and a new set of clients read from the database. This means that Clients defined in the configuration file will not be removed. It also means that multiple ClientListSQL clauses with non-zero RefreshPeriods will not remove each others Clients.

```
# Reread the client list every hour
RefreshPeriod 3600
```

3.16.4. DisconnectAfterQuery

This optional parameter causes the SQL database to be disconnected after each database query. This can be helpful in cases where firewalls etc close connections that have been idle for a long time.

3.16.5. ConnectionHook

This optional parameter specifies a Hook that is run every time this clause connects or reconnects to the SQL database. This is most useful for executing *`func()`* to configure the database connection in customised ways. The hook is called with 2 arguments. The first is a reference to the clause object that is making the connection. The second argument is the DBH handle to the newly connected database.

In the following example, the hook calls DBI func() to configure an Interbase database connection for custom requirements:

```
ConnectionHook sub {$_[1]->func(-access_mode => 'read_write',\
      -isolation_level => 'read_committed',\
      -lock_resolution => 'wait',\
      'ib_set_tx_param')}
```

3.16.6. ConnectionAttemptFailedHook

You can run this hook whenever Radiator attempts to connect to an SQL database and fails to connect. The default is to log the failure. The hook is called with 4 arguments: *`$object`*, *`$dbsource`*, *`$dbusername`*, *`$dbauth`*. *`$object`* is the SqlDb object trying to connect. The other parameters are the currently used values for *`DBSource`*, *`DBUsername`*, and *`DBAuth`*.

In the following example the default hook is replaced with a hook that logs unobscured password.

```
ConnectionAttemptFailedHook sub { \
 my $self = $_[0]; my $dbsource = $_[1]; \
 my $dbusername = $_[2]; my $dbauth = $_[3]; \
 $self->log($main::LOG_ERR, "Could not connect to SQL database with DBI->connect \
            $dbsource, $dbusername, $dbauth: $@ $DBI::errstr"); }
```

3.16.7. NoConnectionsHook

You can run this hook whenever Radiator fails connect to any SQL server. The default is to log the failure. The hook is called with 1 argument: *`$object`*. *`$object`* is the SqlDb object that was trying to connect.

In the following example the default hook is replaced with a hook that logs a very short message.

```
NoConnectionsHook sub { \
  my $self = $_[0]; \
  $self->log($main::LOG_ERR, "Could not connect to any SQL database"); }
```

3.16.8. FarmWorkerSpacing

If this optional parameter is set to non-zero, it specifies the time in seconds for spacing out refresh done by server farm workers. Defaults to not set which causes all farm workers to refresh client list at the same moment. This parameter has only effect when both *`RefreshPeriod`* and global *`FarmSize`* on page 42 parameters are configured.

```
# Reread the client list every hour, use 30 second offset between each worker
RefreshPeriod 3600
FarmWorkerSpacing 30
```

3.17. <SessionDatabase xxxxxx>

The Session Database is used to hold information about current sessions as part of Simultaneous-Use limit checking. It can also be used by external utilities for querying the on-line user population. If you do not specify a SessionDatabase clause in your configuration file, the database will be kept internal to *radiusd*, which is faster, but cannot be used to synchronise multiple instances of Radiator.

If you want to enforce Simultaneous-Use limits and you are running multiple instances of Radiator, you must specify an external Session Database for each Radiator, and you must ensure that all instances of Radiator use the same Session Database. If you fail to do this, Radiator will not be able to correctly enforce Simultaneous-Use limits, and may allow people to log in who have already exceeded their limit.

3.17.1. Identifier

This optional parameter assigns a name to the Session Database, so it can be referred to in other parts of the configuration file, such as the SessionDatabase parameter in Handler.

```
# Here is a useful name for this Session Database
Identifier xxxxxx
```

3.18. <SessionDatabase SQL>

This optional clause specifies an external SQL Session Database for *radiusd*.

SessionDatabase SQL has a number of customisable SQL statements (AddQuery, DeleteQuery, UpdateQuery, ReplaceQuery, ClearNasQuery and CountQuery). These statements are used to add, remove, maintain and count the entries in the SQL Session Database. The default statements will work with the example RADONLINE table in the example SQL schemas in the goodies directory. If you wish, you can use more or fewer columns in your SQL Session Database, and you can change the names of the columns or the table. If you do use a different table schema, you will probably have to change statements to match your schema.

Radiator 4.22 added new SQL statements AddSessionQuery, GetSessionQuery, UpdateSessionQuery and DeleteSessionQuery. See `goodies/hotspot.sql` for a sample database schema used by `goodies/hotspot-fidelio.cfg`.

You can configure the SQL database(s) that *SessionDatabase SQL* uses by defining one or more DBSource, DBUsername and DBAuth lines. For more information about SQL configuration and failure behaviour, see Section 3.8. SQL configuration on page 46.

SessionDatabase SQL is tolerant of database failures. If your database server goes down, Radiator will try to reconnect to a database as described above, starting again at the first database you specified. Whichever database Radiator connects to, it will stay connected to it until that database becomes unreachable, at which time it will again search for a database, starting at the first again. If on the other hand, Radiator is not able to connect to any SQL server, it will stop enforcing Simultaneous-Use limits until one of its databases comes back on line.

SessionDatabase SQL understands also the same parameters as *SessionDatabase xxxxxx*. For more information, see Section 3.17. <SessionDatabase xxxxxx> on page 122. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.18.1. SQL Bind Variables

All SessionDatabase SQL statements support SQL bind variables. For more information, see Section 3.8.1. SQL bind variables on page 47. An example of DeleteQuery with bind variables is:

```
DeleteQuery delete from RADONLINE where NASIDENTIFIER =? and NASPORT=?
DeleteQueryParam %1
DeleteQueryParam %2
```

3.18.2. AddQuery

This SQL statement is executed whenever a new user session starts (i.e. when an Accounting-Request Start message is received). It is expected to record the details of the new session in the SQL database. Special formatting characters may be used. %1 by the NAS IP address, %2 by the NAS-Port, %3 by the SQL quoted Acct-Session-Id. If AddQuery is defined as an empty string, then the query will not be executed.

%0 is replaced by the quoted original user name or rewritten user name, see Section 3.31.23. SessionDatabaseUseRewrittenName on page 158.

It defaults to:

```
insert into RADONLINE (USERNAME, NASIDENTIFIER, NASPORT,
ACCTSESSIONID, TIME_STAMP, FRAMEDIPADDRESS, NASPORTTYPE,
SERVICETYPE) values (%0, '%1', %2, %3, %{Timestamp},
'%{Framed-IP-Address}', '%{NAS-Port-Type}', '%{Service-Type}')
```

3.18.3. DeleteQuery

This SQL statement is executed whenever a user session finishes (i.e. when an Accounting-Request Stop message is received). It is expected to remove the details of the session from the SQL database. Special formatting characters may be used. %1 by the NAS IP address, %2 by the NAS-Port, %3 by the SQL quoted Acct-Session-Id and %4 by the Framed-IP-Address. If DeleteQuery is defined as an empty string, then the query will not be executed.

Delete is executed by default when an Access-Request message is received or a new session is added. This attempts to clear any defunct existing session for the port. If the port is not unique for a session, or there's some other reason to control session database updates, see SessionDatabaseOptions on page 157.

%0 is formatted as defined in section Section 3.18.2. AddQuery on page 123. Section 3.8.1. SQL bind variables on page 47 are supported.

It defaults to:

```
delete from RADONLINE where NASIDENTIFIER='%1' and NASPORT=0%2
```

3.18.4. UpdateQuery

This SQL statement is executed whenever Accounting-Request Alive or Interim-Update message is received. It is expected to update the details of the session in the SQL database. Special formatting characters may be used (the %{attribute} ones are probably the most useful). %1 by the NAS IP address, %2 by the NAS-Port, %3 by the SQL quoted Acct-Session-Id. If UpdateQuery is defined as an empty string, then the query will not be executed and ReplaceQuery, if defined, or AddQuery otherwise, will be used. The default is the empty string.

%0 is formatted as defined in section Section 3.18.2. AddQuery on page 123. Section 3.8.1. SQL bind variables on page 47 are supported.

3.18.5. ClearNasQuery

This SQL statement is executed whenever a NAS reboot is detected. It is expected to clear the details of all sessions on that NAS from the SQL database. Special formatting characters may be used (the %{attribute} ones are probably the most useful). %0 is replaced by the NAS identifier. If ClearNasQuery is defined as an empty string, then the query will not be executed. SQL bind variables are supported.

It defaults to:

```
delete from RADONLINE where NASIDENTIFIER='%0'
```

3.18.6. CountQuery

This SQL statement is executed whenever a Simultaneous-Use check item or MaxSessions must be checked during an Access-Request. It is expected to find and return details of all the user sessions currently in the Session Database for the given User-Name. For each entry, it is expected to return the NAS-Identifier, NAS-Port and Acct-Session-Id, IP Address and optionally a user name (in that order) of each session currently in the Session Database. The returned rows are counted, and if there are apparently too many sessions, SessionDatabase SQL will query each NAS and port to confirm if the user is still on line at that port with that session ID. If a user name is present as the fifth field returned by the query, that is the user name that will be used to confirm the user is still on line. If CountQuery is defined as an empty string, then the query will not be executed, and the current session count will be fixed at 0.

%0 is formatted as defined in section Section 3.18.2. AddQuery on page 123. %1 replaced by the AuthBy's DefaultSimultaneousUse, Simultaneous-Use check item or Handler's Max-Sessions value, depending on the context. SQL bind variables are supported.

It defaults to:

```
select NASIDENTIFIER, NASPORT, ACCTSESSIONID, FRAMEDIPADDRESS \
from RADONLINE where USERNAME=%0
```

Tip

You can make SessionDatabase SQL count sessions in different ways depending on how you want to restrict your sessions. For example, you could limit the number of users permitted to log in to a particular realm with something like:

```
CountQuery select NASIDENTIFIER, NASPORT, ACCTSESSIONID, \
      FRAMEDIPADDRESS from RADONLINE where USERNAME like ?
CountQueryParam %%@%R
```

If your Session Database table included the Called-Station-Id for each session, you could limit the maximum number of users with the same Called-Station-ID with something like:

```
CountQuery select NASIDENTIFIER, NASPORT, ACCTSESSIONID, FRAMEDIPADDRESS \
           from RADONLINE where CALLEDSTATIONID = ?
CountQueryParam %{Called-Station-Id}
```

3.18.7. ReplaceQuery

If this optional parameter is defined, it will be used to replace a record in the session database. If it is not defined, DeleteQuery and AddQuery will be used instead. By default, ReplaceQuery is not defined. %1 by the NAS IP address, %2 by the NAS-Port, %3 by the SQL quoted Acct-Session-Id.

%0 is formatted as defined in section Section 3.18.2. AddQuery on page 123. Section 3.8.1. SQL bind variables on page 47 are supported.

This option is provided because some databases (such as MySQL) offer a more efficient way to ‘insert or replace’ queries. Special formatting characters may be used.

3.18.8. CountNasSessionsQuery

This SQL statement is executed whenever Radiator needs the number of sessions currently logged on to a particular NAS. This is only required if HandleAscendAccessEventRequest is defined and an Ascend-Access-Event-Request is received. %1 is replaced by the NAS IP address. SQL bind variables are supported.

It defaults to:

```
select ACCTSESSIONID from RADONLINE where NASIDENTIFIER='%0'
```

3.18.9. ClearNasSessionQuery

This SQL statement is executed whenever Radiator needs the number of sessions currently logged on to a particular NAS. This is only required if HandleAscendAccessEventRequest is defined and an Ascend-Access-Event-Request is received and Radiator finds that there is a session in the session database that is not recorded in the NAS. %0 is replaced by the NAS IP address and %1 is replaced by the session ID. SQL bind variables are supported.

It defaults to:

```
delete from RADONLINE where NASIDENTIFIER='%0' and ACCTSESSIONID = %1
```

3.18.10. SessionIdentifier

This optional parameter sets the name of a Radius attribute which is used to identify a session. It is useful, for example, when the authentication request contains the session identifier attribute for the subsequent accounting session. The default value is Acct-Session-Id.

```
# This vendor sends id of subsequent accounting session during authentication
SessionIdentifier Vendor-Session-Id
```

3.18.11. AddSessionQuery

An SQL statement to add a new session. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If AddSessionQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are replaced with session parameter names and their values, respectively, and can not currently be used as SQL bind parameters.

`AddSessionQuery` defaults to:

```
INSERT INTO SESSIONS (%0) VALUES (%1)
```

3.18.12. GetSessionQuery

An SQL statement to get a session. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If GetSessionQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

GetSessionQuery defaults to:

```
SELECT * FROM SESSIONS WHERE tenant_id=%2 AND (name=%4 OR id=%3)
```

3.18.13. UpdateSessionQuery

An SQL statement to update an existing session. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If UpdateSessionQuery is defined as an empty string, then the query will not be executed.

%0 contains list of session key=value pairs and can not currently be used as a SQL bind parameter. %1 is undefined.

UpdateSessionQuery defaults to:

```
UPDATE SESSIONS SET %0 WHERE tenant_id=%2 AND id=%3
```

3.18.14. DeleteSessionQuery

An SQL statement to delete a session. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If DeleteSessionQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

DeleteSessionQuery defaults to:

```
DELETE FROM SESSIONS WHERE tenant_id=%2 AND id=%3
```

3.19. <SessionDatabase REDIS>

<SessionDatabase REDIS> provides support for using GossipRedis as session database backend. For more information about GossipRedis, see Section 3.134. <GossipRedis> on page 439. *<SessionDatabase REDIS>* supports SessionDatabase methods to query all sessions and specific sessions for a user. These provide support for RFC 5176 dynamic authorisation support for sending Disconnect-Request and CoA-Request messages. See SessionDatabase REDIS sample file `goodies/sess-redis.cfg`.

<SessionDatabase REDIS> has currently experimental features and will be fully documented later.

<SessionDatabase REDIS> understands also the same parameters as *<SessionDatabase xxxxxx>*. For more information, see Section 3.17. <SessionDatabase xxxxxx> on page 122.

3.19.1. SessionIdentifier

This optional parameter sets the name of a Radius attribute which is used to identify a session. It is useful, for example, when the authentication request contains the session identifier attribute for the subsequent accounting session. The default value is Acct-Session-Id.

```
# This vendor sends id of subsequent accounting session during authentication
SessionIdentifier Vendor-Session-Id
```

3.19.2. SessionKey

Redis key used to store Redis Hash for sessions.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158
- %1 is the NAS Id as defined by Radiator from the request and its attributes
- %2 is the NAS-Port attribute value
- %3 is the Acct-Session-Id attribute value

Here is an example of using *SessionKey*:

```
SessionKey Radiator:SessREDIS:Session:%1:%2:%3:%0
```

3.19.3. SessionEndedKey

Optional Redis key to store Redis Set of ended sessions. If defined, session information for a session will not be deleted when ending the session, but an id of ended session will be added to Redis Set defined by this key. There is no default.

The following specials for format are available:

- %0 is the NAS Id as defined by Radiator from the request and its attributes

Here is an example of using *SessionEndedKey*:

```
SessionEndedKey  Radiator:SessREDIS:EndedSession:%0
```

3.19.4. SessionUpdateKey

Optional Redis key to store Redis Hash of session updates. If defined, session will not be updated with SessionKey, but added with this separate SessionUpdateKey. There is no default.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158

Here is an example of using *SessionUpdateKey*:

```
SessionUpdateKey Radiator:SessREDIS:SessionUpdates:%0
```

3.19.5. NasKey

Redis key to store Redis Set of sessions on NAS.

The following specials for format are available:

- %0 is the NAS Id as defined by Radiator from the request and its attributes

Here is an example of using *NasKey*:

```
NasKey Radiator:SessREDIS:NAS:%0
```

3.19.6. UserKey

Redis key to store Redis Set of sessions for a user.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158

Here is an example of using *UserKey*:

```
UserKey Radiator:SessREDIS:User:%0
```

3.19.7. AddSessionParamDef

AddSessionParamDef specifies the Redis Hash attributes to store session information. The general format is:

```
AddSessionParamDef redishashattributename, radiusattributename[, defaultvalue[, type[, formatted
```

There is no default.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158
- %1 is the NAS Id as defined by Radiator from the request and its attributes
- %2 is the NAS-Port attribute value
- %3 is the Acct-Session-Id attribute value

Formats are considered when the *`formatted`* flag is set.

Supported values for *`type`* are:

- request (the default)
- reply

Here is an example of using *`AddSessionParamDef`*:

```
AddSessionParamDef username,%0,'unknown-user',request,formatted
AddSessionParamDef nas_id,%1,'unknown-nas',request,formatted
AddSessionParamDef nas_port,%2,'0',request,formatted
AddSessionParamDef session_id,%3,'unknown-session',request,formatted
AddSessionParamDef framed_ip,%{Framed-IP-Address},'unknown-address',request,formatted
```

3.19.8. UpdateSessionParamDef

UpdateSessionParamDef specifies the Redis Hash attributes to update session information. The general format is:

```
UpdateSessionParamDef redishashattributename, radiusattributename[, defaultvalue[, formatted[, o
```

There is no default.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158
- %1 is the NAS Id as defined by Radiator from the request and its attributes
- %2 is the NAS-Port attribute value
- %3 is the Acct-Session-Id attribute value

Formats are considered when the *`formatted`* flag is set.

Supported values for *`operator`* are:

- add
- sub

- sub_to_zero

Supported values for *attribtype* are:

- postpaid
- prepaid

Both *operator* and *attribtype* can be used for example with a hotspot service, see goodies/README.hotspot and goodies/hotspot.cfg for a configuration sample. Both *operator* and *attribtype* are currently experimental.

Here is an example of using *UpdateSessionParamDef*:

```
UpdateSessionParamDef username,%0,'unknown-user',formatted
UpdateSessionParamDef nas_id,%1,'unknown-nas',formatted
UpdateSessionParamDef nas_port,%2,'0',formatted
UpdateSessionParamDef session_id,%3,'unknown-session',formatted
UpdateSessionParamDef framed_ip,%{Framed-IP-Address},'unknown-address',formatted
```

3.19.9. DeleteSessionParamDef

DeleteSessionParamDef specifies Redis Hash attributes to update session information when ending a session. The general format is:

```
DeleteSessionParamDef redishashattributename, radiusattributename[, defaultvalue[, formatted[, o
```

There is no default.

The following specials for format are available:

- %0 is the username, possibly changed by Section 3.31.23. SessionDatabaseUseRewrittenName on page 158
- %1 is the NAS Id as defined by Radiator from the request and its attributes
- %2 is the NAS-Port attribute value
- %3 is the Acct-Session-Id attribute value

Formats are considered when the *formatted* flag is set.

Supported values for *operator* are:

- add
- sub
- sub_to_zero

Supported values for *attribtype* are:

- postpaid
- prepaid

Both *operator* and *attribtype* can be used for example with a hotspot service, see goodies/README.hotspot and goodies/hotspot.cfg for a configuration sample. Both *operator* and *attribtype* are currently experimental.

Here is an example of using *DeleteSessionParamDef*:

```
DeleteSessionParamDef username,%0,'unknown-user',formatted
DeleteSessionParamDef nas_id,%1,'unknown-nas',formatted
```

```
DeleteSessionParamDef nas_port,%2,'0',formatted
DeleteSessionParamDef session_id,%3,'unknown-session',formatted
DeleteSessionParamDef framed_ip,%{Framed-IP-Address},'unknown-address',formatted
```

3.19.10. ContextTimeout

Specifies the number of seconds to store session attributes in context. The default value for *`ContextTimeout`* is 2.

Here is an example of using *`ContextTimeout`*:

```
ContextTimeout 2
```

3.20. <SessionDatabase DBM>

This optional clause specifies an external DBM file Session Database for *`radiusd`*. The Session Database is used to hold information about current sessions as part of Simultaneous-Use limit checking. It can also be used by external utilities for querying the online user population If you do not specify a SessionDatabase clause, the database will be kept internal to *`radiusd`*, which is faster, but does not make the data available to other processes.

CAUTION

Because Radiator does not lock the DBM file, a single SessionDatabase DBM file should never be used by more than one instance of Radiator. When attempting to enforce Simultaneous-Use limits across multiple Radiator servers, always use Session-Database SQL.

Radiator will choose the best format of DBM file available to you, depending on which DBM modules are installed on your machine. (Hint: You can force it to choose a particular format by using the DBType parameter)

<SessionDatabase DBM> understands also the same parameters as <SessionDatabase xxxxxx>. For more information, see Section 3.17. <SessionDatabase xxxxxx> on page 122.

3.20.1. Filename

Specifies the filename that holds the Session Database. Defaults to `%D/online`, The actual file names will depend on which DBM format Perl selects for you, but will usually be something like `online.dir` and `online.pag` in DbDir. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21.

```
# Session database in called online2.* in DbDir
Filename %D/online
```

3.20.2. DBType

By default, Radiator and Perl will choose the ‘best’ format of DBM file available to you, depending on which DBM modules are installed on your machine. You can override this choice by specifying DBType as the name of one of the DBM formats supported on your platform. The typical choices are:

- AnyDBM_File
- NDBM_File
- DB_File

- GDBM_File
- SDBM_File
- ODBM_File

But not may be available on your platform. The default is AnyDBM_File, which chooses the best available implementation.

```
# Force it to use DB_File
DBType DB_File
```

3.20.3. SessionIdentifier

This optional parameter sets the name of a Radius attribute which is used to identify a session. It is useful, for example, when the authentication request contains the session identifier attribute for the subsequent accounting session. The default value is Acct-Session-Id.

```
# This vendor sends id of subsequent accounting session during authentication
SessionIdentifier Vendor-Session-Id
```

3.21. <SessionDatabase NULL>

This type of session database stores no session details and always permits multiple logins. It is useful in environments with large user populations, and where no simultaneous-use prevention is required. <SessionDatabase NULL> uses much less memory and fewer CPU cycles than <SessionDatabase INTERNAL> (which is the default session database). The code for <SessionDatabase NULL> was contributed by Daniel Senie (dts@senie.com).

<SessionDatabase NULL> understands also the same parameters as <SessionDatabase xxxxxx>. For more information, see Section 3.17. <SessionDatabase xxxxxx> on page 122.

3.22. <ServiceDatabase xxxxxx>

A Service Database holds information about services and subscriptions. Examples of services are data plans and pre-paid packages offered by hotspot services. Service attributes can be data and time quota, bandwidth limitations, pre- and post-paid requirements. Some services may also place restrictions to re-subscribing the service and have special requirements to the action taken when quota is exceeded.

When a user subscribes to a service, a Service Database also keeps track of per-user subscription state. For example, if a user currently has no active session, there is no entry in a Session Database, but any resources applicable for a service that the user has subscribed to, is tracked by a subscription entry in a Service Database.

If you are running multiple instances of Radiator which all provide the same service, for example a hotspot service, you must specify an external Service Database for each Radiator, and you must ensure that all instances of Radiator use the same Service Database. If you fail to do this, Radiator will not be able to correctly track usage. This requirement is similar to SessionDatabase, see Section 3.17. <SessionDatabase xxxxxx> on page 122 for more information.

See `goodies/README.hotspot` and `goodies/hotspot.cfg` for a configuration sample.

3.22.1. Identifier

This optional parameter assigns a name to the Service Database, so it can be referred to in other parts of the configuration file, such as the ServiceDatabase parameter in *`AuthBy HOTSPOT`*.

```
# Allow other parts of configuration to refer to this Service Database
```

```
Identifier xxxxxx
```

3.22.2. Tenant

This optional parameter assigns a name to the Service Database. For example, if the same SQL database is used with multiple Service Databases that need to be disjoint, these databases should be configured with different Tenant value. This parameter defaults to *`default`*.

```
# We use one database for multiple hotels
Tenant hotspot-hotel1
```

3.23. <ServiceDatabase INTERNAL>

This optional clause specifies an internal Service Database for *`radiusd`*.

`SessionDatabase SQL` is simple to configure but can not be used to synchronise multiple instances of Radiator. Internal Service Database loses all information when *`radiusd`* is shut down.

`ServiceDatabase INTERNAL` understands also the same parameters as *`ServiceDatabase xxxxxx`*. For more information, see Section 3.22. <ServiceDatabase xxxxxx> on page 131.

3.23.1. Service

This optional parameter defines one service for a Service Database. You can define multiple *`Service`* parameters. Format is "key1:value1 key2:value2 ...".

Supported keys are:

- id
- name
- price
- poolIPv4
- poolIPv6
- checkItems
- replyItems
- requireConfirm
- expiry
- prepaidTime
- postpaidTime
- prepaidQuota
- postpaidQuota
- allowQuotaToExceed
- disallowResubscribe
- prepaidUpRate
- prepaidDownRate
- postpaidUpRate
- postpaidDownRate

- throtUpRate
- throtDownRate

For example a sample configuration, see goodies/README.hotspot and goodies/hotspot.cfg. Here is an example of defining multiple Services.

```
# Service 1: free, 1 hour, 50M data, no policers
Service name:free price:0 prepaidTime:1h prepaidQuota:50M \
        replyItems:"OSC-AVPAIR=Test1,OSC-AVPAIR=Test2"

# Service 2: price 1000 cents, 1 day, 100M data, 100M/10M policers
Service name:premium price:1000 prepaidTime:24h prepaidQuota:100M \
        prepaidUpRate:10M prepaidDownRate:100M

# Service 3: price 2000 cents, 1 day, 1G data, 100M/100M policers
Service name:gold price:2000 prepaidTime:24h prepaidQuota:1G \
        prepaidUpRate:100M prepaidDownRate:100M
```

3.24. <ServiceDatabase SQL>

This optional clause specifies an external SQL Service Database for *radiusd*.

ServiceDatabase SQL has a number of customisable SQL statements (AddServiceQuery, GetServiceQuery, UpdateServiceQuery, DeleteServiceQuery, AddSubscriptionQuery, GetSubscriptionQuery, UpdateSubscriptonQuery and DeleteSubscriptionQuery). These statements are used to access the entries in the SQL Service Database. The default statements will work with the example SERVICES and SUBSCRIPTIONS tables in the example SQL schema in file `goodies/hotspot.sql`. If you wish, you can use more or fewer columns in your SQL Service Database, and you can change the names of the columns or the table. If you do use a different table schema, you will probably have to change statements to match your schema.

You can configure the SQL database(s) that *ServiceDatabase SQL* uses by defining one or more DBSource, DBUsername and DBAuth lines. For more information about SQL configuration and failure behaviour, see Section 3.8. SQL configuration on page 46.

ServiceDatabase SQL understands also the same parameters as *ServiceDatabase xxxxxx*. For more information, see Section 3.22. <ServiceDatabase xxxxxx> on page 131. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.24.1. SQL Bind Variables

All ServiceDatabase SQL statements support SQL bind variables. Certain exceptions are documented for each query. For more information, see Section 3.8.1. SQL bind variables on page 47. An example of DeleteSubscriptionQuery with bind variables is:

```
DeleteSubscriptionQuery DELETE FROM SUBSCRIPTIONS WHERE tenant_id=? AND id=?
DeleteSubscriptionQueryParam %2
DeleteSubscriptionQueryParam %3
```

3.24.2. AddServiceQuery

An SQL statement to add a new service. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If AddServiceQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are replaced with service parameter names and their values, respectively, and can not currently be used as SQL bind parameters.

AddServiceQuery defaults to:

```
INSERT INTO SERVICES (%0) VALUES (%1)
```

3.24.3. GetServiceQuery

An SQL statement to get a service. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If GetServiceQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

GetServiceQuery defaults to:

```
SELECT * FROM SERVICES WHERE tenant_id=%2 AND (name=%4 OR id=%3)
```

3.24.4. UpdateServiceQuery

An SQL statement to update an existing service. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If UpdateServiceQuery is defined as an empty string, then the query will not be executed.

%0 contains list of service key=value pairs and can not be currently used as a SQL bind parameter. %1 is undefined.

UpdateServiceQuery defaults to:

```
UPDATE SERVICES SET name=%4 WHERE tenant_id=%2 AND id=%3
```

3.24.5. DeleteServiceQuery

An SQL statement to delete a service. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If DeleteServiceQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

DeleteServiceQuery defaults to:

```
DELETE FROM SERVICES WHERE tenant_id=%2 AND id=%3
```

3.24.6. AddSubscriptionQuery

An SQL statement to add a new subscription. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If AddServiceQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are replaced with subscription parameter names and their values, respectively, and can not currently be used as SQL bind parameters.

AddSubscriptionQuery defaults to:

```
INSERT INTO SUBSCRIPTIONS (%0) VALUES (%1)
```

3.24.7. GetSubscriptionQuery

An SQL statement to get a subscription. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If GetSubscriptionQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

`GetSubscriptionQuery` defaults to:

```
SELECT * FROM SUBSCRIPTIONS WHERE tenant_id=%2 AND (name=%4 OR id=%3)
```

3.24.8. UpdateSubscriptionQuery

An SQL statement to update an existing subscription. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If UpdateSubscriptionQuery is defined as an empty string, then the query will not be executed.

%0 contains list of subscription key=value pairs and can not currently be used as a SQL bind parameter. %1 is undefined.

`UpdateSubscriptionQuery` defaults to:

```
UPDATE SUBSCRIPTIONS SET %0 WHERE tenant_id=%2 AND id=%3
```

3.24.9. DeleteSubscriptionQuery

An SQL statement to delete a subscription. Special formatting characters may be used. Special variable %2 contains a tenant id, %3 contains object's id and %4 contains object's name. If DeleteSubscriptionQuery is defined as an empty string, then the query will not be executed.

%0 and %1 are undefined.

`DeleteSubscriptionQuery` defaults to:

```
DELETE FROM SUBSCRIPTIONS WHERE tenant_id=%2 AND id=%3
```

3.25. <Log FILE>

This optional clause creates a FILE logger, which logs all messages with a priority level of Trace or more to a file. The logging is in addition to any logging to the file defined by global *`LogFile`*. For more information, see Section 3.7.13. LogFile on page 34. The log file is opened, written, and closed for each message, which means you can rotate it at any time.

Tip

The logger becomes active when it is encountered in the configuration file. It logs parse errors from later in the configuration file and subsequent run-time events. Parse errors from earlier in the configuration file are not logged through this clause.

CAUTION

Logging to STDOUT by specifying *`Filename`* as a single dash `-` is no longer supported in Radiator 4.18.

Tip

Global *LogFile* and *Trace* configuration parameters are basically the same as the following:

```
<Log FILE>
      #Trace 0
      Filename filename
</Log>
```

Tip

You can place *<Log xxxxxx>* clause inside any clause in the configuration file. This causes messages originating from within that clauses code to be logged with the logger prior to being logged with any global loggers. This can be handy for debugging or tracing only certain Handler or AuthBy clauses:

```
<Handler>
      # This logs messages from within the Handler
      <Log FILE>
            Trace 4
            Filename xxxxxx
            ...
      </Log>
</Handler>
```

3.25.1. Filename

This specifies the name of the log file. The file name can include special characters. For more information, see Section 3.3. Special formatters on page 21. The default value is **%L/logfile**, a file named `logfile` in *LogDir*. For more information, see Section 3.7.12. LogDir on page 34. Special character **%0** is replaced by the priority integer and **%1** by the log message.

Here is an example of using *Filename*:

```
# Log file goes in /var/log, with year number
Filename /var/log/%Y-radius.log
```

If *Filename* starts with a vertical bar character `|`, the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output is appended to the named file.

```
# Pipe to my-log-prog
Filename |/usr/local/bin/my-log-prog
```

3.25.2. Trace

This parameter defines the priority level of messages to be traced. For more information, see Section 3.7.3. Trace on page 30.

Note

Packet dumps appear only if the global Trace level is set to 4 or more.

3.25.3. LogMicroseconds

This optional parameter makes *<Log FILE>* to log the current microseconds at the end of the time string.

3.25.4. LogTraceId

LogTraceId flag parameter allows logging messages related to an authentication exchange and to its subsequent accounting session with a tracing identifier. *LogTraceId* can be configured for global level and Log clause level. *LogTraceId* enables prepending a tracing ID to messages logged to STDOUT, when *LogStdout* is enabled, and to log file configured with *<Log FILE>* and *<Log SYSLOG>*. For more information, see Section 3.7.2. LogStdout on page 30.

Tip

Support for using the same tracing identifier with accounting messages requires enabling parameter in the authenticating Handler clause. For more information, see Section 3.31.38. AutoClass on page 162.

Tracing ID works in conjunction with the Radiator load balancer allowing coordinated log message indexing and lookup between frontend load balancers and backend workers.

```
# Prepend tracing id to log messages
LogTraceId
```

3.25.5. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It can also be referred to by *Log xxxxxx* in any other clause.

Here is an example of using *Identifier*:

```
<AuthBy whatever>
      # With an Identifier, can refer to this logger from other clauses
      <Log FILE>
            Identifier mylogger
            Filename xxxxxx
      </Log>
      ....
</AuthBy>
<AuthBy whatever>
      # This AuthBy will log to the Log FILE above
      Log mylogger
      .....
</AuthBy>
```

3.25.6. IgnorePacketTrace

This parameter excludes this logger from *PacketTrace* debugging. For more information, see Section 3.14.15. PacketTrace on page 105.

3.25.7. LogFormat

This optional parameter permits you to customise the log string when *LogFormatHook* is not defined. Special formatting characters are permitted. The variables are replaced as follows:

- **%0** by the message severity as an integer
- **%1** by the severity as a string
- **%2** by the log message
- **%3** by tracing identifier string

When using <*Log FILE*>, there is no default value for *LogFormat* and the format is similar to **LogFormat %l:%1:%2**.

When using <*Log SYSLOG*>, there is no default value for *LogFormat* and the format is similar to **LogFormat %2**.

3.25.8. LogFormatHook

This specifies an optional Perl hook that runs for each log message when defined. By default, no hook is defined and *LogFormat* or the default format is used. The hook must return a single value. If the value is defined, it is used as the message to log. An undefined value causes the Log clause to return without logging. This allows *LogFormatHook* to function as a filter to suppress unwanted log messages.

The hook parameters are the following:

- Message severity (integer)
- Log message
- Reference to the current request
- Tracing identifier (string)

See `goodies/logformat.cfg` for a sample configuration file with JSON and CEF (ArcSight Common Event Format) formats.

Note

Consider installing `Cpanel::JSON::XS` or `JSON::XS` for higher performance JSON encoding.

3.26. <Log SYSLOG>

This optional clause creates a SYSLOG logger, which logs all messages with a priority level of Trace or more to the syslog system. <*Log SYSLOG*> requires `Sys::Syslog`. The logging is in addition to any logging to the file defined by LogFile. For more information, see Section 3.7.13. LogFile on page 34.

Messages are logged to syslog with priority levels that depend on the severity of the message. There are 5 defined priority levels and they are logged to the equivalent syslog priority. See the Trace parameter for a description of the priority levels supported.

Ensure that your host's syslog is configured to do something with err, warning, notice, info, and debug priority messages from the Syslog facility you specify, otherwise you do not see any messages. See `/etc/syslog.conf` or its equivalent on your system.

Tip

The logger becomes active when it is encountered in the configuration file. It logs parse errors from later in the configuration file and subsequent run-time events. Parse errors from earlier in the configuration file are not logged through this clause.

Tip

You can place a *<Log xxxxxx>* clause inside any clause in the configuration file. This causes messages originating from within that clauses code to be logged with the logger prior to being logged with any global loggers. This can be handy for debugging or tracing only certain Realms or AuthBy clauses:

```
<Handler>
      # This will log messages from within the Handler
      <Log SYSLOG>
            #Trace 2
            ...
      </Log>
</Handler>
```

3.26.1. Facility

The name of the syslog facility that will be logged to. The default is **user**.

```
# Log to the syslog facility called 'auth'
Facility auth
```

3.26.2. Trace

This defines the priority level of messages to be traced. For more information, see Section 3.7.3. Trace on page 30.

Tip

Packet dumps appear only if the global Trace level is set to 4 or more.

3.26.3. IgnorePacketTrace

Exclude this logger from PacketTrace debugging.

3.26.4. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It can also be referred to by <Log xxxxxx> in any other clause.

```
<AuthBy whatever>
      # With an Identifier, can refer to this logger from
      # other clauses
      <Log SYSLOG>
            Identifier mylogger
            Facility user
      </Log>
      ....
</AuthBy>
<AuthBy whatever>
      # This AuthBy will log to the Log SYSLOG above
      Log mylogger
```

```
      .....
</AuthBy>
```

3.26.5. LogSock

This optional parameter specifies what type of socket to use to connect to the syslog server. The possible values are:

- **native**
- **eventlog**
- **unix**
- **inet**

  This means that TCP is tried first, then UDP.
- **tcp**
- **udp**
- **stream**
- **pipe**
- **console**

The default is to use the *Sys::Syslog* default of **native, tcp, udp, unix, pipe, stream, console**.

CAUTION

Due to limitations in the Sys::Syslog Perl module, if you have multiple *<AuthLog SYSLOG>*, *<AcctLog SYSLOG>* or *<Log SYSLOG>* clauses and if any one has *LogSock* defined, all of them must have *LogSock* defined.

Note

If you use TCP, we recommend you to define both *LogHost* and *LogPort*. If you have not defined *LogPort* and you see error "TCP service unavailable", this means Sys::Syslog is unable to find the destination port. To resolve this, either use *LogPort* to define the port or add *syslog/tcp* or *syslogng/tcp* definitions to /etc/services file. For more information about *LogPort*, see Section 3.26.10. LogPort on page 141.

3.26.6. LogPath

When LogSock is set to unix or stream or pipe, this optional parameter specifies the syslog path. Defaults to _PATH_LOG macro (if your system defines it).

```
LogPath /run/mysyslog/log.sock
```

3.26.7. LogHost

When *LogSock* is set to **tcp** or **udp** or **inet**, this optional parameter specifies the name or address of the syslog host. Defaults to the local host. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

Note

The *LogHost* parameter is passed directly to Perl's Sys::Syslog module which will likely do a DNS query for each logged message. This can cause performance problems and high number of DNS requests with verbose log levels. It is recommended to not set *LogSock* and let the local syslog to do remote logging.

Note

Sys::Syslog does not support IPv6. To log over IPv6, leave *LogSock* unset and let the local syslog do remote logging over IPv6.

```
# Log to a remote host via syslog over udp:
LogSock udp
LogHost your.syslog.host.com
```

3.26.8. LogOpt

This optional parameter allows control over the syslog options passed to Sys::Syslog::openlog. *LogOpt* is a comma separated list of words from the set:

- **cons**
- **ndelay**
- **nofatal**
- **nowait**
- **perror**
- **pid**

As described in the Perl Sys::Syslog documentation.

Defaults to **pid**. Special characters are supported.

```
LogOpt pid,perror
```

3.26.9. LogIdent

This optional string parameter specifies an alternative ident name Sys::Syslog prepends to every syslog message. Defaults to the executable name used to run radiusd. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

```
# Also log server farm instance number
LogIdent %h-%O
```

3.26.10. LogPort

This optional parameter specifies an alternative TCP or UDP destination port on the syslog host. There is no default, which means Sys::Syslog chooses the port. Here is an example of using *LogPort*:

```
LogPort 5514
```

CAUTION

This parameter requires `Sys::Syslog` 0.28 or later.

3.26.11. LogFormat

This optional parameter permits you to customise the log string when *`LogFormatHook`* is not defined. Special formatting characters are permitted. The variables are replaced as follows:

- **`%0`** by the message severity as an integer
- **`%1`** by the severity as a string
- **`%2`** by the log message
- **`%3`** by tracing identifier string

When using <*`Log FILE`*>, there is no default value for *`LogFormat`* and the format is similar to **`LogFormat %l:%1:%2`**.

When using <*`Log SYSLOG`*>, there is no default value for *`LogFormat`* and the format is similar to **`LogFormat %2`**.

3.26.12. LogFormatHook

This specifies an optional Perl hook that runs for each log message when defined. By default, no hook is defined and *`LogFormat`* or the default format is used. The hook must return a single value. If the value is defined, it is used as the message to log. An undefined value causes the Log clause to return without logging. This allows *`LogFormatHook`* to function as a filter to suppress unwanted log messages.

The hook parameters are the following:

- Message severity (integer)
- Log message
- Reference to the current request
- Tracing identifier (string)

See `goodies/logformat.cfg` for a sample configuration file with JSON and CEF (ArcSight Common Event Format) formats.

Note

Consider installing `Cpanel::JSON::XS` or `JSON::XS` for higher performance JSON encoding.

3.26.13. MaxMessageLength

This optional parameter specifies a maximum message length (in characters) for each message to be logged. If specified, each log message is truncated to the specified number of characters prior to logging. Defaults to 0, which means no truncation.

3.27. <Log SQL>

This optional clause creates an SQL logger, which will log all messages with a priority level of Trace or more to an SQL database. The logging is in addition to any logging to the file defined by LogFile. For more information,

see Section 3.7.13. LogFile on page 34. This clause supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

The messages are inserted with the following SQL statement:

```
insert into tablename (TIME_STAMP, PRIORITY, MESSAGE)
            values (time, priority, 'message')
```

You must create a table to insert into before you can use this clause. There are example logging tables created in the example SQL files in the `goodies/` directory of the Radiator distribution.

Tip

The logger becomes active when it is encountered in the configuration file. It will log parse errors from later in the configuration file and subsequent run-time events. Parse errors from earlier in the configuration file will not be logged through this clause.

Tip

You can place a <Log xxxxxx> clause inside any clause in the configuration file. This will cause messages originating from within that clauses code to be logged with the logger prior to being logged with any global loggers. This can be handy for debugging or tracing only certain Realms or AuthBy clauses:

```
<Handler>
      # This will log messages from within the Handler
      <Log SQL>
            #Trace 2
            DBSource dbi:.......
            ...
      </Log>
</Handler>
```

Tip

It is good practice to use a different DBSource than for other SQL clauses in your configuration file. This allows SQL errors on DBSource to be logged.

3.27.1. Table

Defines the name of the SQL table to insert into. Defaults to **`RADLOG`**. Special formatting characters are permitted.

```
# Insert into a table called mylog
Table mylog
```

Tip

You could have Log SQL log to a different table every month with something like:

```
Table RADLOG%Y%m
```

3.27.2. Trace

Defines the priority level of messages to be traced. For more information, see Section 3.7.3. Trace on page 30.

Note

Packet dumps will only appear if the global Trace level is set to 4 or more.

3.27.3. IgnorePacketTrace

Exclude this logger from PacketTrace debugging.

3.27.4. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It can also be referred to by <Log xxxxxx> in any other clause.

```
<AuthBy whatever>
      # With an Identifier, can refer to this logger from
      # other clauses
      <Log SQL>
            Identifier mylogger
            DBSource xxxxxx
            ....
      </Log>
      ....
</AuthBy>
<AuthBy whatever>
      # This AuthBy will log to the Log SQL above
      Log mylogger
      .....
</AuthBy>
```

3.27.5. LogQuery

This optional parameter allows you to control the SQL query that is used to insert log messages into the database. Special formatting characters are permitted.

The default is:

```
insert into %3 (TIME_STAMP, PRIORITY, MESSAGE)
values (%t, %0, %2)
```

The variables are:

- %t is translated as a special character to the current time.
- %0 is converted to the message priority (in integer in the range 0 to 4 inclusive).
- %1 is converted to an ASCII string describing the message priority.

- %2 is converted to the log message, quoted and escaped.
- %3 is converted to the table name defined by the Table parameter above.
- %4 is replaced by the SQL quoted User-Name (or 'NULL' if there is no current Access-Request).
- %5 is replaced by the tracing identifier string.

Tip

You might want to use either %1 or %2 in your query, but rarely both.

3.27.6. LogQueryParam

This optional parameter specifies a bind variable to be used with LogQuery. Special formatting characters %0 - %5 are not quoted when used with LogQueryParam.For more information, see Section 3.8.1. SQL bind variables on page 47.

Note

Many databases do not allow the table name to be defined as a bind variable. If you need %3 and want to use LogQueryParam, use something like this:

```
LogQuery insert into %3 (TIME_STAMP, PRIORITY_MESSAGE) values
(?, ?, ?)
LogQueryParam %t
LogQueryParam %0
LogQueryParam %2
```

3.27.7. MaxMessageLength

This optional parameter specifies a maximum message length (in characters) for each message to be logged. If specified, each log message is truncated to the specified number of characters prior to logging. Defaults to 0, which means no truncation.

Truncation is done prior to SQL quoting of escapes. MaxMessageLength is useful for some types of SQL server that complain if given a string longer than the column it is going in to.

3.28. <Log EMERALD>

This logging module is based on Log SQL and is used to log messages to the Emerald RadLogs table. Emerald is a third-party commercial ISP billing package.

<Log EMERALD> understands the same parameters as <Log SQL>.

3.29. <SNMPAgent>

This optional clause enables an SNMP Agent that will allow you to fetch statistics from Radiator using SNMP version 1 or 2c. Radiator supports all the SNMP objects described in the draft IETF standard defined in draft-ietf-radius-servmib-04.txt, as well as:

- RFC2619 - RADIUS Authentication Server MIB
- RFC2621 - RADIUS Accounting Server MIB

- RFC4669 - RADIUS Authentication Server MIB for IPv6
- RFC4671 - RADIUS Accounting Server MIB for IPv6

Copies of the standards documents are included in the doc directory of the Radiator distribution.

SNMPAgent requires `SNMP_Session-0.92.tar.gz` or later to be installed first.

Red Hat Enterprise Linux, CentOS and Oracle Linux users can install the vendor supplied `perl-SNMP_Session` RPM package. For .deb based systems the package may be available as `libsnmp-session-perl` or other similar name. The source package is available for download from SNMP-session [https://github.com/sleinen/snmp-session]

If you do not include this clause in your Radiator configuration file, it will not respond to any SNMP requests.

```
# Example, showing how to enable SNMP handling
<SNMPAgent>
      ROCommunity mysnmpsecret
</SNMPAgent>
```

If you enable SNMPAgent, you will be able to collect server statistics using a 3rd party SNMP package such as MRTG, Open View etc. You can also use SNMP to reset the server.

You can test that it is working properly with a command on Unix like this one, that gets the value of radiusServIdent from the old draft MIB:

```
$ snmpget -c public -v 1 localhost .iso.org.dod.internet.
3.79.1.1.1.1
SNMPv2-SMI::experimental.79.1.1.1.1 = STRING: "Radiator 3.6 on zulu"
```

And this gets the uptime from the new Authentication server MIB:

```
$ snmpget -c public -v 1 localhost .1.3.6.1.2.1.67.1.1.1.1.2
SNMPv2-SMI::mib-2.67.1.1.1.1.2 = Timeticks: (200) 0:00:02.00
```

3.29.1. Port

This optional parameter specifies the UDP port number that the SNMP Agent is to listen on. It defaults to 161. There should only rarely be any reason to change it. The argument may be either a numeric port number or an alphanumeric service name as specified in `/etc/services/` (or its moral equivalent on your system). Port may contain special formatting characters. A typical use of special formatting characters is with GlobalVar and command line arguments. When FarmSize on page 42 is configured, each worker process needs to use a different port.

```
# Use a non-standard port
Port 9991

# When FarmSize is configured, each worker uses a separate port
#Port 161%O
```

3.29.2. BindAddress

This optional parameter specifies a single host address to listen for SNMP requests on. It is only useful if you are running Radiator on a multi-homed host (i.e. a host that has more than one network address). Defaults to the global value of BindAddress, usually 0.0.0.0 i.e. listen on all networks connected to the host. For more information, see Section 3.7.9. BindAddress on page 33 BindAddress can include special formatting characters. Requires SNMP_Session version 0.92 or greater.

```
# Only listen on one network, not all the ones connected
BindAddress 203.63.154.0
```

3.29.3. Community

SNMP V1 provides a weak method of authenticating SNMP requests, using the ‘community name’.
This optional parameter allows you to specify the SNMP V1 community name that will be honoured by SNMPAgent. Any SNMP request that does not include the correct community name will be ignored. Defaults to nothing. We strongly recommend that you choose a community name and keep it secret.

Community is now deprecated, but is still honoured for backwards compatibility. New implementations should use ROCommunity and /or RWCommunity.

```
# Use a secret community.
Community mysnmpsecret
```

3.29.4. ROCommunity

SNMP V1 provides a weak method of authenticating SNMP requests, using the ‘community name’. This optional parameter allows you to specify the SNMP V1 community name that will be honoured by SNMPAgent for read-only access. Defaults to nothing, you have to define one by yourself.

We strongly recommend that you choose a community name and keep it secret.

```
# Use a secret community.
ROCommunity mysnmprosecret
```

3.29.5. RWCommunity

This optional parameter allows you to specify the SNMP V1 community name that will be used by SNMPAgent to authenticate read-write access. Knowing this secret you are able to reset Radiator via SNMP. Defaults to nothing. If you do not need resetting via SNMP use only ROCommunity.

```
# only necessary for resetting via SNMP
RWCommunity extremelysecure
```

3.29.6. Managers

This optional parameter specifies a list of SNMP managers that have access to SNMPAgent. The value is a list of host names or addresses, separated by white space or comma. You can have any number of Managers lines. Defaults to nothing with all hosts allowed.

```
# allowed SNMP managers
Managers    foo.bar.edu 192.168.1.11, noc.rz.uni-ulm.de
Managers    baz.bar.com,10.1.1.254
```

3.29.7. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It is not used by the standard Radiator code.

3.29.8. SNMPVersion

This optional parameter allows you to specify the SNMP version the agent uses. Currently supported versions are 1 and 2c. Defaults to 1.

3.29.9. PacketTrace

This is an optional flag parameter. It enables logging of received and sent SNMP messages in human-readable form on trace level 4. This is a useful feature during testing and debugging.

3.30. <Realm realmname>

Note

We recommend using Handler clauses for all new configurations. Handlers provide more flexibility for defining how to match requests and make future configuration changes easier to manage.

Note

Using both Realms and Handlers in the same configuration is allowed but may make the Realm/Handler selection hard to understand. For more information, see Section 3.31. <Handler attribute=value,attribute=value, ...> on page 149.

A Realm can be easily converted to a Handler. For example: `<Realm example.com>` becomes `<Handler Realm=example.com>` and `<Realm /\.example\.com$/>` becomes `<Handler Realm=/\.example\.com$/>`. The closing `</Realm>` must also be changed to `</Handler>`.

The beginning of a Realm clause. The clause continues until `</Realm>` is seen on a line. A Realm clause specifies a single RADIUS realm that this server will service. A realm is the part of the users login name that follows the '@' sign. For example if a user logs in as "mikem@open.com.au", then "open.com.au" is the realm. All requests from all users with the realm named in the `<Realm realmname>` line will be handled in the way specified by the rest of the Realm clause. You can configure one or more realms into your server, possibly with a different AuthBy authentication method for each.

The realmname can be either an exact realm name or it can be a Perl regular expression (regexp) including the opening and closing slashes that will match zero or more realms. You can also use the 'x' and 'i' modifiers. If you use a regexp, you should be very careful to check that you regexp will match only those realms you mean it to. Consult your Perl reference manual for more information on writing Perl regexps.

If you omit the realm name from the `<Realm>` line, the clause will match requests with a NULL realm (i.e. where the user did not enter a realm-qualified user name, such as a bare "fred" or "alice").

When Radiator looks for a `<Realm realmname>` clause to match an incoming request, it first looks for an exact match with the Realm name. If no match is found, it will try to do a regexp match against Realm names that look like regexps (i.e. have slashes at each end). If still no match, it looks for a Realm called ***DEFAULT***. If still no match, it logs an error and ignores (i.e. does not reply to) the request. For more information about exceptions, see Section 3.31. <Handler attribute=value,attribute=value, ...> on page 149.

The special ***DEFAULT*** realm (if it is defined) will be used to handle requests from users in realms for which there is no other matching Realm clause.

```
# Handle requests with no realm with UNIX,
# from user@open.com.au with SQL
# from any realm ending in .au by forwarding
# and from any other realm with DBFILE
<Realm>
      <AuthBy UNIX>
            .....
```

```
      </AuthBy>
</Realm>
<Realm open.com.au>
      <AuthBy SQL>
            ......
      </AuthBy>
</Realm>

# Any realm ending in .au
<Realm /.*\.au/>
      <AuthBy RADIUS>
            .....
      </AuthBy>
</Realm>

# Any realm ending in .au, .AU, .Au, .aU (ie its case
# insensitive)
<Realm /.*\.au/i>
      <AuthBy RADIUS>
            .....
      </AuthBy>
</Realm>

# Any other realm
<Realm DEFAULT>
      <AuthBy DBFILE>
            .......
      </AuthBy>
</Realm>
```

A `<Realm>` is a special type of `<Handler>`, and you can use all the same parameters that are described in Section 3.31. <Handler attribute=value,attribute=value, ...> on page 149. However, you can only use realms for selecting the requests. This will not work: `<Realm example.com, Client-Identifier=mynas>`.

3.31. <Handler attribute=value,attribute=value, ...>

This begins a *`<Handler>`* clause. When using the *`<Handler>`* clause, all requests with a specific set of attributes are handled in the same way. You can configure one or more *`<Handler>`* clauses into your server, possibly with a different AuthBy authentication methods for each.

The difference between *`<Handler>`* and *`<Realm>`* is that *`<Handler>`* groups together requests based on the value of any attributes in the request, not just the user's realm as *`<Realm>`* does. That makes *`<Handler>`* clauses more powerful but they are not required as often. A common situation is that you need to distinguish between authentication methods based on the user's realm. Use *`<Handler>`* instead of *`<Realm>`* if you need to choose authentication methods based on other attribute than *`Realm`*, such as *`Called-Station-Id`*, *`Request-Type`*, or some other attribute in the incoming RADIUS request.

In *`<Handler checklist>`* the checklist expression is a list of request attributes that must all match before *`<Handler>`* is used to handle the request. The format is similar as a list of check items in a user file: a list of *`attribute=value`* pairs, separated by commas. For more information, see Section 7.1. Check items on page 462.

If you omit the expression name from the *`<Handler>`* line, the clause matches all requests.

When Radiator looks for *<Handler>* to match an incoming request, it searches each *<Handler>* clause in the order in which they appear in your configuration file. It continues searching until *<Handler>* is found where every check item in the expression matches the request. If any check item does not match, it continues onto the next *<Handler>* until all the Handlers are exhausted. If none of the *<Handler>* clauses match, Access-Request is rejected and Accounting-Request is accepted.

Radiator uses the following algorithm to find the *<Realm>* or *<Handler>* to handle each request:

1. Look for a Realm with an exact match on the realm name.
2. Look for a matching regular expression Realm.
3. Look for a *<Realm DEFAULT>*.
4. Look at each Handler in the order they appear in the configuration file until one where all the check items match the request.
5. Ignore the request.

It is possible to use both *<Handler>* and *<Realm>* clauses in the same configuration file but it is not recommended. Using them together may lead to complicated handler selections and behaviour.

The following example configuration uses 3 authentication methods used in different requests:

- *<AuthBy SQL>*

  This used when *Called-Station-Id* is **662543** and *Service-Type* is **Framed-User**.

- *<AuthBy DBM>*

  This is used when *Called-Station-Id* is **678771** and *Realm* is **open.com.au**.

- *<AuthBy RADIUS>*

  This is used authenticating all the requests that do not match to the previous criteria.

```
<Handler Called-Station-Id=662543,Service-Type=Framed-User>
      <AuthBy SQL>
            ......
      </AuthBy>
</Handler>
<Handler Called-Station-Id=678771,Realm=open.com.au>
      <AuthBy DBM>
            .....
      </AuthBy>
</Handler>
# Handles anything that was not handled above:
<Handler>
      <AuthBy RADIUS>
            .......
      </AuthBy>
</Handler>
```

All instances of the given attribute are checked, so it is possible to define a check that requires a request to contain multiple values. Here are 2 examples of multiple values, both of them are allowed:

```
# Request has attributes in a following order:
# OSC-Group-Identifier=A
# OSC-Group-Identifier=B
```

```
# Does match
<Handler OSC-Group-Identifier=B>
    ....
</Handler>
```

```
# Request has attributes in a following order:
# OSC-Group-Identifier=A
# OSC-Group-Identifier=B

# Does not match
<Handler OSC-Group-Identifier=B,OSC-Group-Identifier=C>
    ....
</Handler>

# Does match
<Handler OSC-Group-Identifier=B,OSC-Group-Identifier=A>
    ....
</Handler>
```

3.31.1. RewriteUsername

This is an optional parameter. It enables you to alter the username in authentication and accounting requests. For more details and examples, see Section 8. Rewriting user names on page 484.

3.31.2. RewriteFunction

This optional parameter allows you to define you own special Perl function to rewrite user names. You can define an arbitrarily complex Perl function that might call external programs, search in databases or whatever. The user name is changed to whatever is returned by this function.

If you define a RewriteFunction for a Realm or Handler, it will be used in preference to any RewriteUsername. RewriteUsername will be ignored for that Realm or Handler.

```
# Strip out NULs, trailing realms, translate to
# lower case and remove single quotes
RewriteFunction sub { my($a) = shift; $a =~ s/[\000]//g; $a =~
s/^([^@]+).*/$1/; $a =~ tr/[A-Z]/[a-z]/; $a =~ s/'//g; $a; }
```

3.31.3. AcctLogFileName

The names of the files used to log `Accounting-Request` messages in the standard radius accounting log format. All `Accounting-Request` messages will be logged to the files, regardless of their `Acct-Status-Type`. For more information about log file format, see Section 9.5. Accounting log file on page 491. If no *`AcctLogFileName`* is defined, accounting messages will not be logged for this realm. The default is no logging. The file name can include special formatting characters as described in Section 3.3. Special formatters on page 21, which means that, for example, using the **`%c`** specifier, you can maintain separate accounting log files for each Client. The *`AcctLogFileName`* files are always opened written and closed for each message, so you can safely rotate them at any time.

If the AuthBy module you select does no special accounting logging, you may want to enable this parameter for the Realm. Note that logging to *`AcctLogFileName`* is in addition to any recording that a specific AuthBy module might do (such as, say, AuthBy SQL). The user name that is recorded in the log file is the rewritten user name when *`RewriteUsername`* is enabled.

You can specify any number of *AcctLogFileName* parameters. Each one will result in a separate accounting log file. If need to modify accounting messages and log the modified result, consider using an AcctLog on page 158.

If the file name starts with a vertical bar character (‘|’) then the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output will be appended to the named file:

```
# Pipe to my-log-prog
AcctLogFileName |/usr/local/bin/my-log-prog
```

Tip

You can change the logging format with AcctLogFileFormat:

```
# Log all accounting to a single log file in LogDir
AcctLogFileName %L/details
```

3.31.4. AcctLogFileFormat

This optional parameter is used to alter the format of the accounting log file from the standard radius format when AcctLogFileFormatHook is not defined. AcctLogFileFormat is a string containing special formatting characters. It specifies the format for each line to be printed to the accounting log file. A newline will be automatically appended. It is most useful if you use the %{attribute} style of formatting characters (to print the value of the attributes in the current packet).

```
AcctLogFileFormat %{Timestamp} %{Acct-Session-Id}\
%{User-Name}
```

3.31.5. AcctLogFileFormatHook

Specifies an optional Perl hook that will is used to alter the format of the accounting log file from the standard radius format when defined. The hook must return the formatted accounting record. A newline will be automatically appended. By default no hook is defined and AcctLogFileFormat or the default format is used. The hook parameter is the reference to the current request.

See `goodies/logformat.cfg` for a sample configuration file with JSON format.

Tip

Consider installing `Cpanel::JSON::XS` or `JSON::XS` for higher performance JSON encoding.

3.31.6. AccountingHandled

This parameter forces Radiator to acknowledge Accounting requests, even if the AuthBy modules for the Realm or Handler would have normally ignored the request. This is useful if you do not really want to record Accounting requests, but your NAS keeps retransmitting unless it gets an acknowledgment. If the accounting message is also proxied, consider *IgnoreAccountingResponse* on page 218 on the proxy configuration.

Note

All backend processing, such as SQL queries, is done first, before the reply is sent back to the NAS.

```
# My AuthBy SQL ignores accounting
AccountingHandled
```

3.31.7. AccountingAccepted

This is a flag parameter. When set, it forces Radiator to acknowledge an Accounting request when it is received, even if the AuthBy modules for the Realm or Handler would otherwise ignore or reject the request. This is useful if you want to send an acknowledge to NAS immediately without waiting for SQL, proxy or other backend. Usually this parameter is not used.

Unlike *AccountingHandled* on page 152, this parameter does not require *IgnoreAccountingResponse*. The response is automatically ignored when *AccountingAccepted* is set.

3.31.8. PreProcessingHook

This optional parameter allows you to define a Perl function that will be called during packet processing. *PreProcessingHook* is called for each request before per-Realm user name rewriting, before accounting log files are written, and before any PreAuthHooks. A reference to the current request is passed as the first argument, and a reference to the reply packet currently being constructed is passed as the second argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

PreProcessingHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Fake a new attribute into the request, so its logged in the
# accounting log file
PreProcessingHook sub { ${$_[0]}->add_attr('My-Realm', \
      'some.realm.com');}
```

3.31.9. PostProcessingHook

This optional parameter allows you to define a Perl function that will be called during packet processing. *PostProcessingHook* is called for each request after all authentication methods have been called and just before a reply is sent back to the requesting NAS. A reference to the current request is passed as the first argument, and a reference to the reply packet currently being constructed is passed as the second argument. If the processing results in no reply (for example, if the request is proxied) then *PostProcessingHook* is not called.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore, you should not use trailing comments in your hook.

PostProcessingHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current reply or many other things.

```
# Fake a new attribute into the reply
PostProcessingHook sub { ${$_[1]}->add_attr('Class', \
      'billing rate 1');}
```

3.31.10. PreAuthHook

This optional parameter allows you to define a Perl function that will be called during packet processing. *PreAuthHook* is called for each request after per-Realm user name rewriting and before it is passed to any AuthBy clauses. A reference to the current request is passed as the first argument, and a reference to the reply packet currently being constructed is passed as the second argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

PreAuthHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Fake a new attribute into the request
PreAuthHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.31.11. PostAuthHook

This optional parameter allows you to define a Perl function that is called during packet processing. *PostAuthHook* is called for each request after it has been passed to all the *AuthBy* clauses. A reference to the current request is passed as the first argument, and a reference to the reply packet currently being constructed is passed as the second argument. The third argument is a reference to the result of the authentication (*$main::ACCEPT*, *$main::REJECT* and so on). The fourth argument is a reference to a string variable holding the reason for a reject, or undefined if none is available. To change the type of reply, set the reference to third argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook is executed. Multiline hooks, with trailing backslashes (\), are parsed by Radiator into a one long line. Therefore do not use trailing comments in your hook.

PostAuthHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Add some reply attributes to the reply message
# if it is a REJECT and there is 1 or fewer there already
PostAuthHook sub { ${$_[1]}->add_attr('test-attr', \
      'test-value') \
      if ${$_[2]} == $main::REJECT \
      && ${$_[1]}->attr_count() <= 1; }
```

3.31.12. AuthByPolicy

This parameter allows you to control how multiple AuthBy clauses in this Handler or Realm are used. For more information, see Section 3.38.1. AuthByPolicy on page 185.

3.31.13. AuthBy

This specifies that the Handler is to be authenticated with an <AuthBy> clause that is defined elsewhere. The argument must specify the Identifier of the AuthBy clause to use. The AuthBy clause may be defined anywhere else: at the top level, or in a Realm or Handler clause. You can have as many AuthBy parameters as you wish. They will be used in the order that they appear in the configuration file (subject to AuthByPolicy) in the same way as <AuthBy > clauses.

Tip

This is a convenient way to reuse the same authenticator for many Realms or Handlers.

```
<AuthBy xxxxxx>
      Identifier myidentifier
</AuthBy>
<Realm xxxxxx>
      # This authenticates through the AuthBy defined above
      AuthBy myidentifier
</Realm>
```

3.31.14. Identifier

This optional parameter acts as a label that can be useful for custom code in hooks. It is not used by the standard Radiator code.

3.31.15. StripFromRequest

Strips the named attributes from the request before passing it to any authentication modules. The value is a comma separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.31.16. AddToRequest

Adds attributes to the request before passing it to any authentication modules. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.31.17. AddToRequestIfNotExist

Adds attributes to the request before passing it to any authentication modules. Unlike AddToRequest, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name if they are not there already
AddToRequestIfNotExist Calling-Station-Id=1,Login-IP-Host=%h
```

3.31.18. <AuthBy xxxxxx>

This marks the beginning of an AuthBy clause in a Handler or Realm, which defines how to authenticate and record accounting information for all the users in this Realm or Handler. The xxxxxx is the name of a specific AuthBy module. For more information about configuring specific AuthBy clauses, see the relevant sections.

You can have 0 or more AuthBy clauses in a Handler or Realm. The AuthBy clauses will be executed in order until the AuthByPolicy for the Realm or Handler is satisfied.

<AuthBy xxxxxx> both defines an authentication method and specifies where it should be used.

Note that something like

```
<Realm xxxxxx>
      <AuthBy xxxxxx>
            ....
      </AuthBy>
      ....
</Realm>
```

Is equivalent to

```
<AuthBy xxxxxx>
      Identifier myidentifier
      .....
</AuthBy>
<Realm xxxxxx>
      # This authenticates through the AuthBy defined above
      AuthBy myidentifier
      ....
</Realm>
```

3.31.19. UsernameCharset

This optional parameter checks that every user name consists only of the characters in the specified character set. This can be useful to reject requests that have user names that cannot be valid. The value of the parameter is a Perl character set specification. See your Perl reference manual for details about how to construct Perl character set specifications. Note that the some special characters must be escaped with a backslash. This parameter is not set by default and no character set check is done.

`UsernameCharset` is available as a global and Handler level parameter. The character set checks are done for both User-Name attribute and EAP identity.

When a request is processed by a Handler, User-Name attribute must pass both global and per Handler *`UsernameCharset`* checks. When an EAP-Response/Identity message is handled by an AuthBy, the EAP identity must pass both global and per Handler *`UsernameCharset`* checks. The Handler is the last Handler that processed the request before it was passed to the AuthBy.

This example permits only alphanumeric, period, underscore, the @-sign, and dash. Note that a dash at the end of character class needs not to be escaped with a backslash:

```
UsernameCharset a-zA-Z0-9._@-
```

3.31.20. RejectHasReason

Normally, when Radiator rejects an Access-Request, it sets the reply message to "Request Denied". This optional parameter forces Radiator to put an additional Reply- Message into Access-Reject indicating why the rejection occurred. This may be useful for debugging in some cases, since some NASs will display the Reply-Message during an interactive login. However it should be noted that some PPP clients do not show the Reply-Message to the end user.

```
# Show any rejection reason to the end user (maybe)
```

```
RejectHasReason
```

3.31.21. SessionDatabase

This optional parameter specifies a particular Session Database to use for the enclosing Realm or Handler. The value of the parameter must be the Identifier of a SessionDatabase clause. The default behaviour is to use the last global SessionDatabase specified in the configuration file. If no SessionDatabases are specified in the configuration file, then the default INTERNAL session database will be used.

`SessionDatabaseOptions` is available for changing how a Handler uses its session database. For more information, see Section 3.31.22. SessionDatabaseOptions on page 157.

Tip

The user name used to access the session database depends on the setting of *`SessionDatabaseUseRewrittenName`* and *`SessionDatabaseOptions`*.

Tip

This is an advanced use parameter. Only very unusual configurations need this parameter to be set. The default behaviour suits for most situations.

```
<Handler xxxxxx>
      # If this was not here, we would default to
      # myspecialsessiondb2 (the last one in the config file)
      SessionDatabase myspecialsessiondb1
      .....
</Handler>
<SessionDatabase SQL>
      Identifier myspecialsessiondb1
      ....
</SessionDatabase>
<SessionDatabase SQL>
      Identifier myspecialsessiondb2
      ....
</SessionDatabase>
```

3.31.22. SessionDatabaseOptions

SessionDatabaseOptions allows controlling how Handler communicates with its session database.

The following options are available:

- **`NoDeleteBeforeAuthentication`** suppresses the call to delete a session from session database when Access-Request is received. Calling delete is useful, for example, when a NAS port can only have a single active session. An Access-Request indicates that a new session is starting and calling delete allows to clear the old session if its accounting Stop was lost.
- **`AddBeforeAuthentication`** causes this Handler to add a new session before passing the Access-Request to AuthBys

- **AddAfterAuthentication** causes this Handler to add a new session after successful authentication without waiting for Accounting Start
- **UseRewrittenName** tells the Handler to use rewritten user name with session database. If the option is set, the rewritten user name (as rewritten by RewriteUsername etc) is used. Otherwise the original User-Name attribute from the request is used.
- **NoDeleteOnSessionStop** replaces a call to delete a session from session database with a call to update a session when an accounting stop is received. This allows the session database to hold records of ended sessions.

Enabling UseRewrittenName provides the same functionality as setting the SessionDatabaseUseRewrittenName (for more information, see Section 3.31.23. SessionDatabaseUseRewrittenName on page 158). New configurations must use this option instead of setting the configuration parameter.

You can enter multiple options by separating them with commas. Here is an example how to skip session delete during authentication:

```
SessionDatabaseOptions NoDeleteBeforeAuthentication
```

3.31.23. SessionDatabaseUseRewrittenName

This optional parameter controls the user name used to update and access the session database for this Handler. If SessionDatabaseUseRewrittenName is true, the rewritten user name (as rewritten by RewriteUsername and the others) is used. Otherwise the original User-Name attribute from the request is used. This is false by default.

New configurations must use SessionDatabaseOptions to control this and any other possible options. For more information, see Section 3.31.22. SessionDatabaseOptions on page 157.

3.31.24. AcctLog

This specifies that the Handler is to log all accounting messages with an *<AcctLog>* clause that is defined elsewhere. The argument must specify the Identifier of the AcctLog clause to use. The AcctLog clause may be defined anywhere else: at the top level, or in a Realm or Handler clause. You can have as many AcctLog parameters as you wish. They will be used in the order that they appear in the configuration file in the same way as *<AcctLog xxxxxx>* clauses.

Tip

This is a convenient way to reuse the same authentication logger for many Realms or Handlers.

```
<AcctLog xxxxxx>
      Identifier myacctlogidentifier
      ....
</AuthLog>
<Handler xxxxxx>
      # This logs through the AcctLog defined above
      AcctLog myacctlogidentifier
</Handler>
```

3.31.25. AuthLog

This specifies that the Handler is to log all authentications with an *<AuthLog>* clause that is defined elsewhere. The argument must specify the Identifier of the AuthLog clause to use. The AuthLog clause may be defined anywhere else: at the top level, or in a Realm or Handler clause. You can have as many AuthLog parameters as

you wish. They will be used in the order that they appear in the configuration file in the same way as *<AuthLog xxxxxx>* clauses.

Tip

This is a convenient way to reuse the same authentication logger for many Realms or Handlers.

```
<AuthLog xxxxxx>
      Identifier myidentifier
      ....
</AuthLog>
<Handler xxxxxx>
      # This logs through the AuthLog defined above
      AuthLog myidentifier
</Handler>
```

3.31.26. <AuthLog xxxxxx>

Indicates that logging of authentication success and failure should be handled in a special way. *<AuthLog FILE>*, *<AuthLog SQL>* and *<AuthLog SYSLOG>* are currently supported. When authentication is complete, each of the AuthLog clauses defined for the handler will be executed in order. For each AuthLog, the authentication details will be logged according to the parameters for that clause. You can have as many *<AuthLog xxxxxx>* clauses as you wish, or none at all.

<AuthLog xxxxxx> both defines a logging method and specifies where it should be used.

Note that something like

```
<Handler xxxx>
      <AuthLog xxxxxx>
            ....
      </AuthLog>
      ....
</Handler>
```

Is identical to

```
<AuthLog xxxxxx>
      Identifier myidentifier
</AuthLog>
<Handler xxxxxx>
      # This log through the AuthLog defined above
      AuthLog myidentifier
</Handler>
```

3.31.27. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with *PacketTrace* set off or **0**.

PacketTrace is available for the following clauses:

- *Client*

- *Handler*
- *Realm*
- *AuthBy*
- *ServerDIAMETER*
- *ServerRADSEC*
- *ServerTACACSPLUS*

Here is an example of using *PacketTrace*:

```
# Debug any packets that pass through here
PacketTrace
```

3.31.28. LogRejectLevel

Log level for rejected authentication attempts. Defaults to global LogRejectLevel value.

3.31.29. DefaultReply

This is similar to Section 3.14.20. AddToReply on page 106 except it adds attributes to an Access-Accept only if there would otherwise be no reply attributes. *StripFromReply* does never remove any attributes added by *DefaultReply*. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. You can use any of the special % formats in the attribute values. There is no default.

Although this parameter can be used in any AuthBy method, it is most useful in methods like *<AuthBy UNIX>* and *<AuthBy SYSTEM>*, which do not have a way of specifying per-user reply items. In other AuthBy methods you can also very easily set up a standard set of reply items for all users, yet you can still override reply items on a per-user basis.

```
# If the user had no reply items set some
DefaultReply Service-Type=Framed,Framed-Protocol=PPP
```

3.31.30. StripFromReply

Strips the named attributes from Access-Accepts before replying to the originating client. The value is a comma separated list of RADIUS attribute names. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. There is no default. This is useful, for example, with AuthBy RADIUS to prevent downstream RADIUS servers sending attributes you do not like back to your NAS.

```
# Remove dangerous attributes from the reply
StripFromReply Framed-IP-Netmask,Framed-Compression
```

3.31.31. AllowInReply

This optional parameter is the complement to *StripFromReply*: It specifies the only attributes that are permitted in an Access-Accept. It is useful, for example, to limit the attributes that are passed back to the NAS from a proxy server. This way you can prevent downstream customer RADIUS servers from sending back illegal or troublesome attributes to your NAS.

AllowInReply does not prevent other attributes being added locally by DefaultReply, AddToReply and AddToReplyIfNotExist.

```
# Only permit a limited set of reply attributes.
AllowInReply Session-Timeout, Framed-IP-Address
```

3.31.32. AddToReply

Adds attributes to reply packets. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. You can use any of the special % formats in the attribute values. There is no default. AddToReply adds attributes to replies to all types of request that are handled by this clause.

Although this parameter can be used in any AuthBy method, it is most useful in methods like AuthBy UNIX, which don't have a way of specifying per-user reply items.

```
# Append some necessary attributes for our pops
AddToReply cisco-avpair="ip:addr_pool=mypool"
```

3.31.33. AddToReplyIfNotExist

This is similar to AddToReply, but only adds an attribute to a reply if and only if it is not already present in the reply. Therefore, it can be used to add, but not override a reply attribute. This is contributed by Vincent Gillet <vgi@oleane.net>.

3.31.34. AddExtraCheck

This is a string that adds check items before passing to the authentication modules. The value is a list of comma-separated check items value pairs.

3.31.35. RejectReason

This optional parameter specifies a string that is used as the Reply-Message if *<AuthBy INTERNAL>* rejects a request. The enclosing Realm or Handler must also have *RejectHasReason* enabled.

Here is an example of using *RejectReason*:

```
RejectReason Your account has been disabled
```

3.31.36. DynamicReply

This optional parameter specifies a reply item that will be eligible for run-time variable substitution. That means that you can use any of the % substitutions in Section 3.3. Special formatters on page 21 in that reply item. You can specify any number of DynamicReply lines, one for each reply item you want to do replacements on. Any packet-specific replacement values will come from the Access-Accept message being constructed, and not from the incoming Access-Request. That means that special characters like %n will not be replaced by the received User-Name, because User-Name is in the request, but not the reply.

In the following example, substitution is enabled for USR-IP-Input-Filter in an AuthBy clause. When a user authenticates, the %a in the filter will be replaced by the users IP Address, which makes the filter an anti-spoof filter.

```
<AuthBy whatever>
      ......
      UseAddressHint
      DynamicReply USR-IP-Input-Filter
</AuthBy>
```

In the users file:

```
DEFAULT User-Password = "UNIX"
      Framed-IP-Address = 255.255.255.254,
```

```
        Framed-Routing = None,
        Framed-IP-Netmask = 255.255.255.255,
        USR-IP-Input-Filter = "1 REJECT src-addr != %a;",
        Service-Type = Framed-User
```

Note

This parameter used to be called *Dynamic*. That name is still recognised as a synonym for *DynamicReply*.

3.31.37. MaxSessions

This parameter allows you to apply a simple limit to the number of simultaneous sessions a user in this Realm is permitted to have. It is most common to limit users to either one session at a time or unlimited, but Radiator also supports other numbers.

MaxSessions works by looking at each accounting request for a realm when it arrives. Whenever Start is seen for a user, the count of their number if current sessions is incremented, and whenever a Stop is seen, it is decremented. When an access request is received, the number of sessions current for that user is compared to MaxSessions. If the user already has MaxSessions sessions or more, Radiator replies with an access denial. By setting MaxSessions to 0, you can temporarily deny access to all users in the realm.

MaxSessions applies a hard limit that cannot be overridden by DefaultSimultaneousUse parameter or by per-user Simultaneous-Use check items. For more information, see Section 3.32.14. DefaultSimultaneousUse on page 169. For many applications, you may wish to consider using DefaultSimultaneousUse instead of MaxSessions. You can control the maximum number of sessions on a per-user basis with the Simultaneous-Use check item. For more information, see Section 7.1.16. Simultaneous-Use on page 470.

The session count for each user is stored entirely within Radiator (unless you specify a SessionDatabase clause). This means that if you restart or reinitialise Radiator, it will lose count of the number of current sessions for each user. Radiator can use SNMP to confirm whether a user is already logged in or not. For more information, see Section 3.14.35. NasType on page 109.

Note that if Radiator fails to receive an accounting Stop request, it may result in incorrectly assume the users are not permitted to log in when, in fact, they are. Correct this by restarting Radiator, or by sending an artificial accounting stop for the user using the *radpwtst* utility or by configuring Radiator to query the NAS directly. For more information, see Section 6.1. radpwtst on page 449 and Section 3.14.35. NasType on page 109.

```
# Limit all users in this realm to max of 1 session
MaxSessions 1
```

Tip

You can set the maximum number of simultaneous sessions for individual users with the Simultaneous-Use check item. In this case, the smaller of MaxSessions and the users Simultaneous-Use will apply.

3.31.38. AutoClass

This optional parameter will force Radiator to automatically reply with a Class attribute with Access-Accept. The Class attribute should be echoed back by the NAS with the subsequent Accounting-Request messages that relate to this authentication. The Class attribute works in conjunction with tracing identifier, for more

information about logging authentication and accounting messages, see Section 3.7.4. LogTraceId on page 30. AutoClass is currently experimental.

Even if you do not plan to use tracing identifier, enabling AutoClass may be useful for matching authentication logs with accounting sessions.

```
# Turn on global LogTraceId
LogTraceId

# Generate a Class attribute that NAS will echo back
<Handler ...>
      ......
      AutoClass
</Handler>
```

AutoClass supports optional parameters for further Class attribute formatting. These arguments are added to the values that AutoClass adds by default. The currently support arguments are *`uuid`* and *`formatted`* which add a hex value UUID or Radiator formatted string. The format is specified after a whitespace, see below for an example. The default is not to do further formatting.

```
# Add a v3 UUID to Class
AutoClass uuid
```

Note

Perl module Data::UUID is required for the *`uuid`* parameter.

```
# Add the farm instance to Class
AutoClass formatted %O
```

3.31.39. WtmpFileName

The name of a Unix SVR4 wtmp format file to log Accounting-Request messages. All Accounting-Request messages will be logged. If WtmpFileName is not defined, no messages will be logged in this format. The default is no logging. The file name can include Section 3.3. Special formatters on page 21, which means that, for example, using the %c specifier, you can maintain separate accounting log files for each Client. The WtmpFileName file is always opened written and closed for each message, so you can safely rotate it at any time. Start messages are logged as USER_PROCESS (7), all other messages are logged as DEAD_PROCESS (8).

You may wish to use your standard Unix administration tools (such as last(1)) to process information in the wtmp file.

Tip

Red Hat Enterprise Linux 6.1 and later uses a modified format for wtmp files, compared with earlier versions. In order to use tools like **`last`** on a wtmp file produced by Radiator, you will need to use the `-o` flag (which specifies lib5c format).

3.31.40. FramedGroup

This optional parameter acts similarly to Framed-Group reply items, but it applies to all Access-Requests authenticated by this clause. If FramedGroup is set and a matching FramedGroupBaseAddress is set in the Client from where the request came, then a Framed-IP-Address reply item is automatically calculated by adding the NASPort in the request to the FramedGroupBaseAddress specified by FramedGroup. For more information, see Section 3.14.38. FramedGroupBaseAddress on page 112.

Note

You can override the value of FramedGroup for a single user by setting a Framed-Group reply item for the user.

```
# Work out the users IP address from the first
# FramedGroupBaseAddress specified in out client
FramedGroup 0
```

3.31.41. PasswordLogFileName

The name of file to log authentication attempts to. Attempts where the user is not found, are not logged. The default is no logging. The file name can include Section 3.3. Special formatters on page 21, which means that using the %C, %c and %R specifiers, you can maintain separate password log files for each Realm or Client or a combination.

Each login attempt that generates a password check will be logged to the file, one attempt per line. For more information about file format, see Section 9.6. Password log file on page 492.

```
# Help desk want to see all password attempts
PasswordLogFileName %L/password.log
```

If the file name starts with a vertical bar character ('|') then the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output will be appended to the named file:

```
# Pipe to my-log-prog
PasswordLogFileName |/usr/local/bin/my-log-prog
```

3.31.42. ExcludeFromPasswordLog

For security reasons, you can exclude certain users from the passwords logged to PasswordLogFileName. The value is a white space separated list of user names.

```
# Do not log password from our sysadmin or root
ExcludeFromPasswordLog root admin ceo nocboss
```

3.31.43. ExcludeRegexFromPasswordLog

Similar to ExcludeFromPasswordLog, but the value is a Perl regular expression.

```
# Do not log password from anyone that begins with root
ExcludeRegexFromPasswordLog ^root
```

3.31.44. HandleAscendAccessEventRequest

This optional parameter causes Radiator to respond to Ascend-Access-Event-Request messages. These messages are sent by some types of specially configured Ascend NASs. They contain a count of the number of sessions the NAS thinks it currently has in each Class.

When HandleAscendAccessEventRequest is enabled, it causes Radiator to confirm that its SQL Session Database is up to date. If the session counters do not agree, Radiator will use snmpwalk to confirm the existence of its sessions, deleting from the SQL Session Database the ones that do not really exist.

Tip

HandleAscendAccessEventRequest requires that an SQL Session Database is configured for this Handler or Realm. See Section 3.31.21. SessionDatabase on page 157.

Tip

You must set NasType to Ascend for the SNMP session confirmation to work correctly.

```
# handle Ascend-Access-Event-Request messages
HandleAscendAccessEventRequest
```

3.32. <AuthBy xxxxxx>

This marks the beginning of an AuthBy clause, which defines how to authenticate and record accounting information. The xxxxxx is the name of a specific AuthBy module. This section lists the parameters all AuthBys use. AuthBys may also use other parameters that are specific for that AuthBy. AuthBy clauses may be defined at the top level or within a Realm or Handler clause.

Under special circumstances, you can have more than one AuthBy clause for a Realm or Handler or may want to use <AuthBy GROUP>, see Section 3.38. <AuthBy GROUP> on page 184. This makes the Handler or Realm or <AuthBy GROUP> try each AuthBy method in turn until one of them either Accepts or Rejects the request. You can change this with AuthByPolicy. For more information, see Section 3.38.1. AuthByPolicy on page 185. For example, it is useful to have an <AuthBy SQL> followed by an <AuthBy RADIUS>, which causes all authentication and accounting requests to be forwarded, and also all accounting requests will be recorded in SQL. This is good for keeping track of all requests forwarded to, say a global roaming server.

If there are no AuthBy clauses in a Realm or Handler, then Access requests will be rejected, and Accounting requests will be ignored.

3.32.1. Identifier

This allows you to assign a symbolic name to an AuthBy clause and its configuration. This allows you to refer to it by name in an Auth-Type check item when authenticating a user.

The most common use of this is to create a “System” authenticator, typically with an <AuthBy UNIX> clause. A typical example configuration file that uses this feature might be:

```
<Realm DEFAULT>
      <AuthBy FILE>
      </AuthBy>
</Realm>
<AuthBy UNIX>
```

```
      Identifier System
</AuthBy>
```

You can then have something like this in your users file:

```
DEFAULT Auth-Type = System
      Framed-IP-Netmask .....
      ...........
```

In this example, all users in all realms will match the DEFAULT user in the users file. This will in turn check their user name and password against a UNIX password file as configured by the AuthBy UNIX clause in the configuration file. If the password checks out, they will get the RADIUS attributes specified in the second and subsequent lines of the DEFAULT user entry in the users file (in this example, Framed-IP-Netmask).

3.32.2. StripFromReply

Strips the named attributes from Access-Accepts before replying to the originating client. The value is a comma separated list of RADIUS attribute names. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. There is no default. This is useful, for example, with AuthBy RADIUS to prevent downstream RADIUS servers sending attributes you do not like back to your NAS.

```
# Remove dangerous attributes from the reply
StripFromReply Framed-IP-Netmask,Framed-Compression
```

3.32.3. AllowInReply

This optional parameter is the complement to *`StripFromReply`*: It specifies the only attributes that are permitted in an Access-Accept. It is useful, for example, to limit the attributes that are passed back to the NAS from a proxy server. This way you can prevent downstream customer RADIUS servers from sending back illegal or troublesome attributes to your NAS.

AllowInReply does not prevent other attributes being added locally by DefaultReply, AddToReply and AddToReplyIfNotExist.

```
# Only permit a limited set of reply attributes.
AllowInReply Session-Timeout, Framed-IP-Address
```

3.32.4. AllowInReject

This optional parameter specifies the only attributes that are permitted in an Access-Reject. This can be useful in Handlers with multiple AuthBys where the attributes added before a rejecting AuthBy need to be stripped from the resulting Access-Reject.

AllowInReject is not set by default and does not remove anything from Access-Rejects.

```
# Only permit a limited set of attributes in a reject.
AllowInReject Message-Authenticator, EAP-Message
```

3.32.5. AddToReply

Adds attributes to reply packets. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromReply removes attributes from the reply before AddToReply adds any to the reply. You can use any of the special % formats in the attribute values. There is no default. AddToReply adds attributes to replies to all types of request that are handled by this clause.

Although this parameter can be used in any AuthBy method, it is most useful in methods like AuthBy UNIX, which don't have a way of specifying per-user reply items.

```
# Append some necessary attributes for our pops
AddToReply cisco-avpair="ip:addr_pool=mypool"
```

3.32.6. AddToReplyIfNotExist

This is similar to AddToReply, but only adds an attribute to a reply if and only if it is not already present in the reply. Therefore, it can be used to add, but not override a reply attribute. This is contributed by Vincent Gillet <vgi@oleane.net>.

3.32.7. DefaultReply

This is similar to Section 3.14.20. AddToReply on page 106 except it adds attributes to an Access-Accept only if there would otherwise be no reply attributes. *`StripFromReply`* does never remove any attributes added by *`DefaultReply`*. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. You can use any of the special % formats in the attribute values. There is no default.

Although this parameter can be used in any AuthBy method, it is most useful in methods like *`<AuthBy UNIX>`* and *`<AuthBy SYSTEM>`*, which do not have a way of specifying per-user reply items. In other AuthBy methods you can also very easily set up a standard set of reply items for all users, yet you can still override reply items on a per-user basis.

```
# If the user had no reply items set some
DefaultReply Service-Type=Framed,Framed-Protocol=PPP
```

3.32.8. FramedGroup

This optional parameter acts similarly to Framed-Group reply items, but it applies to all Access-Requests authenticated by this clause. If FramedGroup is set and a matching FramedGroupBaseAddress is set in the Client from where the request came, then a Framed-IP-Address reply item is automatically calculated by adding the NASPort in the request to the FramedGroupBaseAddress specified by FramedGroup. For more information, see Section 3.14.38. FramedGroupBaseAddress on page 112.

Note

You can override the value of FramedGroup for a single user by setting a Framed-Group reply item for the user.

```
# Work out the users IP address from the first
# FramedGroupBaseAddress specified in out client
FramedGroup 0
```

3.32.9. NoDefault

Normally if Radiator searches for a user in the database and either does not find one, or finds one but the users check items fail, Radiator will then consult the DEFAULT user entry. However, if the NoDefault parameter is set, Radiator will never look for a DEFAULT.

```
# Save time by never looking for a default
NoDefault
```

3.32.10. NoDefaultIfFound

Normally if Radiator searches for a user in the database and finds one, but the users check items fail, Radiator will then consult the DEFAULT user entry. However, if the NoDefaultIfFound parameter is set, Radiator will only look for a DEFAULT if there were no entries found in the user database for the user.

```
# do not fall through to DEFAULT if a users check item failed
NoDefaultIfFound
```

3.32.11. DefaultLimit

This optional parameter specifies the maximum number of DEFAULT users to look up in the database.

3.32.12. AcceptIfMissing

Normally, if a user is not present in the user database, they will always be rejected. If this optional parameter is set, and a user is not in the database they will be unconditionally accepted. If they are in the database, they will be accepted if and only if their check items pass in the normal way.

This option is usually only useful in conjunction with a following AuthBy that will actually check all passwords. It can therefore be used to impose additional checks on a subset of your user population.

Note

When you specify *`AcceptIfMissing`*, all reply attributes set for this AuthBy (such as *`DefaultReply`*, *`AddToReply`*, *`AddToReplyIfNotExist`*, etc.) will be applied.

Note

This does not automatically accept DEFAULT users.

```
# Apply some extra checks for those users in the users file,
# then authenticate them with a SQL DB.
<Handler>
      AuthByPolicy ContinueWhileAccept
      <AuthBy FILE>
            AcceptIfMissing
            Filename %D/users
      </AuthBy>
      <AuthBy SQL>
            # whatever
      </AuthBy>
</Handler>
```

3.32.13. IgnoreIfMissing

Normally, if a user is not present in the user database, they will always be rejected. If this optional parameter is set, and a user is not in the database, the enclosing AuthBy ignores the request. If they are in the database, they will be accepted if and only if their check items pass in the normal way.

This option is usually only useful in conjunction with one or more following AuthBys.

IgnoreIfMissing does not change database failure behaviour. When a database lookup fails, database failure behaviour is triggered and ignore is returned while the database is deemed to be in failed state. Database failure behaviour is not triggered when *IgnoreIfMissing* sets AuthBy result to ignore.

Note

If user is not found and DEFAULT users are looked up and found, *IgnoreIfMissing* does not apply.

```
# Consider the second LDAP server only when the user is not found
# from the first LDAP server or the first LDAP server has failed.
<Handler>
      # This is also the default AuthByPolicy
      AuthByPolicy ContinueWhileIgnore
      <AuthBy LDAP2>
            IgnoreIfMissing
            # Other configuration parameters
      </AuthBy>
      <AuthBy LDAP2>
            # LDAP configuration parameters
      </AuthBy>
</Handler>
```

3.32.14. DefaultSimultaneousUse

This optional parameter defines a default value for Simultaneous-Use check items that will apply only if the user does not have their own user-specific Simultaneous-Use check item.

```
# Use sim-use of 2 unless there is a user-specific entry
DefaultSimultaneousUse 2
```

3.32.15. AuthenProto

AuthenProto specifies which authentication protocols are permitted for authentication. It is an optional parameter. *AuthenProto* is available for all *AuthBy* modules but its functionality depends on the specific *AuthBy*. It does not affect proxying or special *AuthBy* modules, such as *<AuthBy INTERNAL>* which do their own request handling.

If the authentication request is rejected because of this parameter setting, the failure is logged as a *WARNING* in Radiator log, and available for all *AuthLog* clauses as the failure reason.

Allowed values for *AuthenProto* are:

- **PAP**
- **CHAP**
- **MSCHAP**
- **MSCHAPv2**
- **SIPDigest**
- **EAP**
- **AuthorizeOnly**
- **Unknown**

The default value is **PAP, CHAP, MSCHAP, MSCHAPv2, EAP, AuthorizeOnly**. The value is **AuthorizeOnly** if the request does not match any of the other values but has Service-Type attribute set to Authorize-Only. The value is **Unknown** when the authentication protocol cannot be determined. The default for *AuthenProto* covers the usual user authentication protocols. Add **Unknown** to those *AuthBy*s that need to handle the authentication requests that do not have the correct combination of any of the following:

- CHAP attributes
- MSCHAP attributes
- MSCHAPv2 attributes
- User-Password attribute
- EAP-Message attributes
- Service-Type attribute set to Authorize-Only

Here is an example of using *AuthenProto*:

```
# Allow PAP only
AuthenProto PAP

# Allow all CHAP variants
AuthenProto CHAP,MSCHAP,MSCHAPv2

# Empty list allows nothing
AuthenProto

# Unknown allows anything else. This allows PAP and requests that are not
# CHAP, MSCHAP, MSCHAPv2, SIPDigest, EAP or AuthorizeOnly
AuthenProto PAP, Unknown
```

3.32.16. CaseInsensitivePasswords

This optional parameter permits case insensitive password checking for authentication methods that support plaintext password checks, such as FILE, SQL, DBFILE and some others. It has no effect on CHAP or MSCHAP passwords, or on password checking involving any encrypted passwords.

```
# Permit case insensitive password checks
CaseInsensitivePasswords
```

3.32.17. RejectEmptyPassword

This optional parameter forces any Access-Request with an empty password to be rejected. This is provided as a work around for some broken remote RADIUS servers (VMS RADIUS server in particular) that incorrectly accept requests with empty passwords.

```
# Reject anything with an empty password
RejectEmptyPassword
```

3.32.18. AuthenticateAccounting

This optional parameter forces Radiator to authenticate accounting requests (as well as the normal Access-Requests). It is very rarely required.

3.32.19. IgnoreAuthentication

This optional parameter causes the AuthBy to IGNORE all authentication requests. This can be useful for providing fine control over authentication with multiple AuthBy clauses.

3.32.20. IgnoreAccounting

This optional parameter causes the AuthBy to IGNORE all accounting requests. This can be useful for providing fine control over authentication with multiple AuthBy clauses.

3.32.21. RcryptKey

This optional parameter indicates that the passwords in the user database may have been reversibly encrypted using Rcrypt. Any password in the database read by this AuthBy and which is in the form `{rcrypt}anythingatall` will be interpreted as an Rcrypt password and the function `Radius::Rcrypt::decrypt()` will be used to decrypt it before any password comparisons are made. Rcrypt encrypted passwords are compatible with PAP, CHAP, and MS-CHAP.

Rcrypt reversible encryption allows you keep your user password database reasonably secure, but still support CHAP, MS-CHAP and other authentication methods that require access to the plaintext password. Rcrypt encryption is supported as an option by the RAdmin RADIUS user administration package from Radiator Software.

Tip

The value of RcryptKey must exactly match the key that was used to originally encrypt the passwords.

Tip

You can add Rcrypt encryption and decryption to other programs with the `Radius::Rcrypt` Perl module supplied with Radiator.

3.32.22. TranslatePasswordHook

This optional parameter specifies a Perl hook that can be used to convert, translate or transform plaintext passwords after retrieval from a user database and before comparison with the submitted password received from the client. The hook is passed the following arguments:

- $_[0] The correct password as retrieved from the user database.
- $_[1] A reference to the current AuthBy module object.
- $_[2] The user name being authenticated.
- $_[3] A flag saying whether the correct password is encrypted by crypt() or MSCHAP.
- $_[4] A reference to the current RADIUS request.

The hook is expected to return a single scalar value containing the transformed password. In the following example, all passwords are transformed to uppercase.

```
# Translate all passwords to UPPERCASE
TranslatePasswordHook sub {$_[0] =~ tr/a-z/A-Z/; return $_[0]}
```

3.32.23. CheckPasswordHook

This optional parameter specifies a Perl hook that can be used to compare password retrieved from a user database with the submitted password received from the client. The retrieved passwords must start with the leading prefix {OSC-pw-hook}. The hook must return true when the password is deemed correct. This hook is for proprietary hash formats and other custom password check methods. This hook runs after TranslatePasswordHook. The hook is passed the following arguments:

- $_[0] A reference to the current RADIUS request
- $_[1] The password submitted by the user
- $_[2] The correct password as retrieved from the user database without the prefix

3.32.24. AutoMPPEKeys

Some NASs, PPoE, VPDN, wireless controllers and wireless access points require MPPE key attributes in the Access-Accept message for setting up encryption. If this AuthBy is doing MS-CHAP V1 authentication, then setting this parameter will force Radiator to automatically reply with MS-CHAP-MPPE-Keys. If this AuthBy is doing MS-CHAP V2 or EAP authentication, then setting this parameter will force Radiator to automatically reply with MS-MPPE-Send-Key and MS-MPPE-Recv-Key.

This flag parameter is optional and defaults to not set. Keys may be computed from the plaintext password, password NT hash or derived by some other means that depends on the AuthBy or AuthBy's EAP method. For example AuthBy LSA and AuthBy NTLM derive the keys with the help of Active Directory. Not all EAP methods support MPPE keys.

When keys are computed based on the password, the password must be available in one of the previously mentioned formats and the user must have User-Password check item.

With TLS based EAP methods, such as EAP-FAST, EAP-TLS, EAP-TTLS and PEAP, MS-MPPE-Send-Key and MS-MPPE-Recv-Key are computed based on TLS handshake results and do not depend on password availability.

Tip

This option is almost always required with AuthBys that authenticate TLS based EAP methods or other EAP methods used with 802.1X authentication for wireless and wired networks.

3.32.25. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with `PacketTrace` set off or **`0`**.

`PacketTrace` is available for the following clauses:

- `Client`
- `Handler`
- `Realm`
- `AuthBy`
- `ServerDIAMETER`
- `ServerRADSEC`

- *ServerTACACSPLUS*

Here is an example of using *PacketTrace*:

```
# Debug any packets that pass through here
PacketTrace
```

3.32.26. CachePasswords

This parameter enables a user password cache in this AuthBy. It can be used to improve the performance of slow AuthBy clauses, or when large number of identical requests for the same user are likely to occur, or when multiple request might result from a one-time password (in a multi-link or wireless roaming environment) etc.

If this parameter is set, all Access-Requests will first be checked against a password cache that contains a copy of the last valid Access-Accept for that user. If the cache contains a matching password that has not exceeded its CachePasswordExpiry, the previous reply will be sent back, without looking up the user again in this AuthBy. Therefore the possibly slow process of consulting the user database or proxying the request can be sometimes avoided.

Tip

Not all AuthBy clauses support this parameter (or CachePasswordExpiry and CacheReplyHook), but the ones that do include UNIX, FILE, DBFILE, SQL, LDAP, ACE, RADMIN and RADIUS. Other AuthBy clauses may or may not support this parameter.

Tip

Use of this parameter with a large user population can cause large amounts of memory use by the Radiator process.

Tip

If Radiator is restarted, the password cache is lost.

Note

Matching of cached passwords can never succeed for CHAP or MS-CHAP authentication requests.

3.32.27. CachePasswordExpiry

If CachePasswords is enabled, this parameter determined the maximum age (in seconds) for cached passwords. Defaults to 86400 seconds (1 day). Cached passwords that are more than this number of seconds old will not be used.

3.32.28. CachePasswordKey

Sets the key used when *CachePasswords* is enabled. **%0** is replace by the current username which is also the default key when *CachePasswordKey* is not set.

```
# %0 expands to the current username which is the
```

```
# hardcoded default when CachePasswordKey is not set.
CachePasswordKey %0

# Use current username and an attribute from $p (current request)
CachePasswordKey %0:%{X-Device-Group}
```

3.32.29. CacheReplyHook

This optional parameter allows you to define a Perl hook that runs when a cached reply is about to be returned to the NAS because of CachePasswords.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

The hook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change or set up environment variables, umasks etc.

The arguments passed to the hook are:

- Argument 0 is the object handle for this AuthBy object.
- Argument 1 is the user name being authenticated.
- Argument 2 is the received request packet.
- Argument 3 is the cached reply packet that is about to be returned to the NAS.

3.32.30. ClearTextTunnelPassword

This optional parameter prevents Radiator from encrypting Tunnel-Password attributes in replies. This is provided in order to support older NASs that do not support encrypted Tunnel-Password.

```
# Old NAS does not support encrypted Tunnel-Password
ClearTextTunnelPassword
AddToReply Tunnel-Password=xxxxxx
```

3.32.31. NoCheckPassword

This optional parameter causes AuthBy not to check the password. This means that any password entered by the user will be accepted. This parameter is useful in conjunction with other authentication methods where the password check is done elsewhere.

3.32.32. ConsumePassword and LeavePassword

`ConsumePassword` and *`LeavePassword`* are optional parameters that allow an AuthBy to extract and use a portion of a password. This is typically required with multiple AuthBys to implement two-factor or one-time password authentication. When set for an AuthBy, only a portion of the current password is used by the AuthBy. The remaining portion becomes the new current password. This allows each AuthBy to check separate portions of a password.

`ConsumePassword` causes an AuthBy to extract a portion from the **beginning** of the current password. Possible values for *`ConsumePassword`* are:

- Positive number

  This defines how many characters are extracted from the beginning of the password. If the number is larger than password length, the whole password is used and the next AuthBy gets a zero length password.

- Negative number

  This defines how many characters are left at the end of the password while the rest are extracted from the beginning of the password. If the number is larger than password length, zero length password is used is used and the next AuthBy gets the whole password.

- One or more other characters

  This defines a separator. Portion of the password before the first occurrence of separator is used. The separator is discarded and the next AuthBy gets what remains after the first separator. If the separator is not a part of the password, the password is used fully and the next AuthBy gets a zero length password.

- Empty

  The whole password is used. The next AuthBy gets a zero length password.

LeavePassword causes an AuthBy to extract a portion from the **end** of the current password. Possible values for *LeavePassword* are:

- Positive number

  This defines how many characters are left at the beginning of password while the rest are extracted from the end of the password. If the number is larger than password length, zero length password is used and the next AuthBy gets the whole password.

- Negative number

  This defines how many characters are extracted from the end of the password. If the number is larger than password length, the whole password is used and the next AuthBy gets a zero length password.

- One or more other characters

  This defines a separator. Portion of the password after the last occurrence of separator is used. The separator is discarded and the next AuthBy gets the portion before the last separator. If the separator is not a part of the password, zero length password is used and the next AuthBy gets the whole password.

- Empty

  Zero length password is used. The next AuthBy gets the whole password.

Examples

Notice in the examples how *ConsumePassword* consumes the beginning of password and *LeavePassword* leaves it unchanged:

- If the password is **part1:part2:part3** and *ConsumePassword* is **:**, the next AuthBy gets **part2:part3**
- If the password is **987654variablepart** and *ConsumePassword* is **6**, the next AuthBy gets **variablepart**
- If the password is **variablepart987654** and *ConsumePassword* is **-6**, the next AuthBy gets **987654**
- If the password is **part1:part2:part3** and *LeavePassword* is **:**, the next AuthBy gets **part1:part2**
- If the password is **variablepart987654** and *LeavePassword* is **-6**, the next AuthBy gets **variablepart**
- If the password is **987654variablepart** and *LeavePassword* is **6**, the next AuthBy gets **987654**

For more configuration examples, see `goodies/duo.cfg` and `goodies/digipassStatic.cfg`. This example shows how to check a password, for example **variablepart987654**, that has a variable length static password followed by 6 digit TOTP code.

```
<Handler>
    AuthByPolicy ContinueWhileAccept
    <AuthBy SQL>
        # Current password is static password followed by a token code
        # SQL configuration to check static password and other check attributes
        # Fetch reply attribute configuration

        # Use and remove static password leaving the token code
        ConsumePassword -6
    </AuthBy>
    <AuthBy SQLTOTP>
        # Current password is just the token code
    </AuthBy>
</Handler>
```

3.32.33. AuthenticateAttribute

This optional parameter allows you to control which RADIUS attribute is used to look up the user database. Normally, Radiator uses the User-Name RADIUS attribute as the key to find a user in the user database. If the AuthenticateAttribute parameter is defined, it specifies the name of an alternative RADIUS attribute that will be used as the key during the lookup in the user database. This is useful in order to do authentication based on, say, the Calling-Station-Id:

```
# look up database based on Calling-Station-Id
AuthenticateAttribute Calling-Station-Id
```

3.32.34. UsernameMatchesWithoutRealm

This optional parameter forces authenticators to strip any realm from the user name before authenticating the name. This allows users to log in with user@realm, even though their user database user name is just user.

3.32.35. Blacklist

Reverses the sense of authentication checks, making it easier to implement blacklists. If the user name matches the user database, then they will be rejected. If there is no match they will be accepted.

3.32.36. EAPErrorReject

If an EAP error occurs, REJECT instead of IGNORE. The RFCs say that IGNORE is the correct behaviour, but REJECT can work better in some load balancing situations.

3.32.37. PasswordPrompt

For AuthBys that support the feature, specifies the prompt to be used when asking for a password.

3.32.38. PostAuthHook

This optional parameter allows you to define a Perl function that is called during packet processing. *`PostAuthHook`* is called for each request when the *`AuthBy`* is done processing the request and has returned.

The hook parameters and behaviour are the same as for Handler's *`PostAuthHook`*. For more information, see Section 3.31.11. PostAuthHook on page 154.

3.32.39. AllowNULInUsername

By default, user names with NUL octets cause a reject with user database lookups. This optional flag parameter allows user database lookups when the looked up user name contains a NUL octet. This is not set by default.

Here is an example of using *AllowNULInUsername*:

```
# User-Name attributes and EAP identities are allowed to have NUL octets
AllowNULInUsername
```

3.32.40. AddExtraCheck

This is a string that adds check items before passing to the authentication modules. The value is a list of comma-separated check items value pairs.

3.32.41. RejectReason

This optional parameter specifies a string that is used as the Reply-Message if *<AuthBy INTERNAL>* rejects a request. The enclosing Realm or Handler must also have *RejectHasReason* enabled.

Here is an example of using *RejectReason*:

```
RejectReason Your account has been disabled
```

3.32.42. Fork

The parameter forces the authentication module to fork(2) before handling the request. Fork should only be set if the authentication module or the way you have it configured is slow, so that it takes more than a fraction of a second to process the request.

If you do not understand what forking is for or how it can improve the performance of your Radiator server, talk about it with someone who does before using it. Not all authentication methods will benefit from forking. Fork has no effect on Windows platforms.

Note

In particular, it does not usually make sense to use Fork with AuthBy SQL, AuthBy FILE, AuthBy LDAP2 or any of the other common authentication methods provided with Radiator. Further, some SQL and LDAP client libraries are not robust across forks. You might want to consider using Fork with AuthBy EXTERNAL or a custom authentication module if it needs to do significant amounts of IO, or to communicate with a remote system.

```
# This AuthBy EXTERNAL program is very slow, and does lots of IO
Fork
```

3.32.43. UseAddressHint

This optional parameter forces Radiator to honour a Framed-IP-Address in an Access-Request request unless it is overridden by a Framed-IP-Address in the user's reply items. If you enable this, then users will get the IP Address they ask for. If there is a Framed-IP-Address reply item for a user, that will override anything they might request.

```
# Let users get addresses they ask for
UseAddressHint
```

3.32.44. DynamicReply

This optional parameter specifies a reply item that will be eligible for run-time variable substitution. That means that you can use any of the % substitutions in Section 3.3. Special formatters on page 21 in that reply item. You can specify any number of DynamicReply lines, one for each reply item you want to do replacements on. Any packet-specific replacement values will come from the Access-Accept message being constructed, and not from the incoming Access-Request. That means that special characters like %n will not be replaced by the received User-Name, because User-Name is in the request, but not the reply.

In the following example, substitution is enabled for USR-IP-Input-Filter in an AuthBy clause. When a user authenticates, the %a in the filter will be replaced by the users IP Address, which makes the filter an anti-spoof filter.

```
<AuthBy whatever>
      ......
      UseAddressHint
      DynamicReply USR-IP-Input-Filter
</AuthBy>
```

In the users file:

```
DEFAULT User-Password = "UNIX"
      Framed-IP-Address = 255.255.255.254,
      Framed-Routing = None,
      Framed-IP-Netmask = 255.255.255.255,
      USR-IP-Input-Filter = "1 REJECT src-addr != %a;",
      Service-Type = Framed-User
```

Note

This parameter used to be called *`Dynamic`*. That name is still recognised as a synonym for *`DynamicReply`*.

3.32.45. DynamicCheck

This optional parameter specifies a check item that will be eligible for run-time variable substitution prior to authentication. That means that you can use any of the % substitutions in table at Section 3.3. Special formatters on page 21 in that check item. You can specify any number of Dynamic- Check lines, one for each check item you want to do replacements on.

In the following example, substitution is enabled for the Group check item. When a user authenticates, the %{Shiva-VPN-Group} in the check item will be replaced with the value of the Shiva-VPN-Group attribute in the authentication request. You could use this mechanism to verify that the user is in the Unix group corresponding to their Shiva-VPN-Group.

```
<AuthBy whatever>
      ......
      DynamicCheck Group
</AuthBy>
```

In the users file:

```
DEFAULT Group=%{Shiva-VPN-Group}
      Framed-IP-Address = 255.255.255.254,
```

```
        Framed-Routing = None,
        Framed-IP-Netmask = 255.255.255.255,
        ........
```

3.33. <AuthBy ACE>

The *<AuthBy ACE>* module performs authentication directly to an RSA Security Authentication manager (formerly SecurID ACE/Server). For more information, see RSA website [https://www.rsa.com/en-us]. RSA Security Authentication Manager provides a token-based one-time password system. *<AuthBy ACE>* requires the `Authen::ACE4` Perl module from CPAN. Compile it for your chosen Perl distribution. For more information, see Section 2.1.2. CPAN on page 3. You can also contact Radiator Software in case you need help with your *Authen::ACE4* setup.

Tip

<AuthBy ACE> works with RSA Authentication Manager 7.1 and later. If you have AM 7.1 or later you might consider using *<AuthBy RSAAM>*, since it is more capable and more portable.

Before using this AuthBy method ensure that you have the following things:

- Installed and configured Authentication Manager
- Purchased tokens for each user from RSA Security
- Added the users that you wish to authenticate to Authentication Manager, and assigned each one a token
- In Authentication Manager, added an 'Agent Host' for each Radiator server host you intend to operate. If Radiator will run on the same host as Authentication Manager, make sure you add an Agent Host for that host.
- Followed the installation instructions in `goodies/ace.txt` in the Radiator distribution package

<AuthBy ACE> works also with EAP-Generic-Token-Card and EAP-PEAP-Generic-Token-Card authentication, as well as RADIUS PAP and TTLS-PAP.

Tip

There are more detailed installation and testing instructions in the `goodies/ace.txt` file in your distribution.

Tip

An alternative to using *<AuthBy ACE>* is to proxy requests to the optional RADIUS server that comes with Authentication Manager (although that RADIUS server has many fewer features and supported platforms than Radiator).

Tip

There is an example Radiator configuration file for *<AuthBy ACE>* in `goodies/ace.cfg` in your Radiator distribution.

Tip

<AuthBy ACE> uses the State reply item to get the RADIUS client to carry the context from one step of authentication to the next. If you wish to test *<AuthBy ACE>* with radpwtst, use the -interactive flag.

```
radpwtst -interactive -user fred -password 1234574424
```

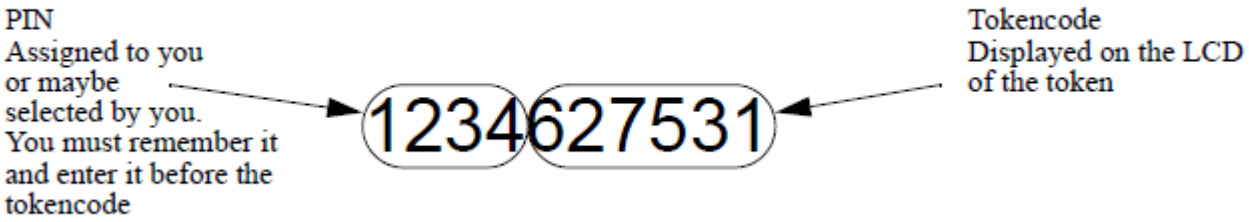
3.33.1. Using RSA Security token cards to log in with AuthBy ACE

RSA Security produce 2 main types of hardware tokens. 'Standard cards' and 'key fobs' tokens have an LCD display that changes every 60 seconds, but no other buttons or switches. 'Pinpad' cards have an LCD display, and also 0-9 keys, a diamond button and a 'P' button. The way you generate your password depends on the type of token you have.

In RSA terminology, the number displayed by a token is called the 'tokencode' The number is typically 6 or 8 digits long and changes every 60 seconds. The 'PIN' is a secret 4 digit or longer number that you will be assigned (or may have selected) and which you must remember. The 'passcode' is what you use as your password to log in, and consists of the PIN followed by the tokencode.

For standard tokens, the passcode is formed from the PIN followed by the tokencode displayed on the token at that time. So, for example, if the PIN you have remembered that was assigned to you is '1234', and the token is currently displaying the tokencode '627351', then the passcode that you will use as your password is '1234627351' (that is 10 digits). See Figure 1. Making a password from an RSA Security Token Code (not for Pinpads) on page 180.

Figure 1. Making a password from an RSA Security Token Code (not for Pinpads)



When a standard token is set to New PIN Mode by the ACE administrator, you must first login in with your PIN and the current tokencode, and you will then be prompted by Radiator for your new PIN. If the token is set to New PIN mode and also has a Cleared PIN, you must omit your PIN.

For Pinpad tokens, you have to enter the PIN into the token to generate the passcode. If for example, your remembered PIN is 1234, enter the PIN into the Pinpad one digit at a time (press 1 - 2 - 3 - 4), then press the diamond button. The token will then display a new tokencode, say '736284'. You would then use 736284 (that's only 6 digits) as your password. When a Pinpad token is set to New Pin Mode by the ACE administrator, you must create your passcode in the usual way and then the system will prompt them for their new PIN., When a Pinpad is in New Pin Mode and also has a cleared PIN, the user must enter the tokencode showing on the token (without using a PIN) first, and then the system will prompt them for their new PIN.

Note that some types of tokens display an 8 digit tokencode, rather than a 6 digit tokencode. AuthBy ACE understands the following parameters as well as those described in Section 3.32. <AuthBy xxxxxx> on page 165.

3.33.2. ConfigDirectory

This optional parameter specifies the location of the ACE Agent `sdconf.rec` file, which the ACE Agent client libraries use to find the location of the ACE server(s). It is also the directory where the node secret files will be saved. The user ID that runs Radiator must have read and write access to this directory. The file `sdconf.rec` must be present on the machine where AuthBy ACE is running. Defaults to the value of the *`VAR_ACE`* environment variable, if set, else `/var/ace/`. This parameter has no effect on Windows.

```
ConfigDirectory /opt/ace/data
```

3.33.3. Timeout

This optional parameter specifies the maximum time that a single ACE authentication is allowed to take. A typical ACE authentication will require several RADIUS transactions, involving multiple requests and challenges until the final Access-Accept is sent, and there is no guarantee that the user will complete the authentication process. If the total time for the authentication exceeds this number of seconds, the authentication will be abandoned. Defaults to 300 seconds (5 minutes).

3.33.4. EnableFastPINChange

Some NASs, notably some Juniper devices, have non-standard behaviour in New Pin Mode: when the user is asked whether they want to set their PIN, the NAS automatically gets the new PIN from the user and returns it to the Radiator server, which is expected to use it to set the PIN immediately. This flag enables compatibility with this behaviour if the user/device enters a PIN instead of ‘y’ or ‘n’. If EnableFastPINChange is enabled and the token is in New PIN mode and the users response does not look like ‘y’ or ‘n’, then the response will be used to set the new PIN, bypassing the PIN confirmation dialogues. Defaults to disabled.

3.34. <AuthBy TEST>

The AuthBy TEST module always accepts authentication requests, and ignores (but replies to) accounting requests. It is implemented in `AuthTEST.pm`. It is useful for testing purposes, but you should be sure not to leave them lying around in your configuration file, otherwise you might find that users are able to be authenticated when you really did not want them to.

AuthBy TEST can also serve as a useful template for developing your own AuthBy modules.
For more information, see file:/data/radiator-reference-manual/source/HighAvailabilityradiusd/HighAvailabilityradiusd.dita#HighAvailabilityradiusd.

3.34.1. Identifier

This allows you to assign a symbolic name to an AuthBy clause and its configuration. This allows you to refer to it by name in an Auth-Type check item when authenticating a user.

3.35. <AuthBy FILE>

AuthBy FILE authenticates users from a user database stored in a flat file. It ignores (but replies to) accounting requests. It is implemented in AuthFILE.pm. It understands standard Livingston user files. For more information about Livingston user files, see Section 9.2. Flat file user database on page 490.

For performance reasons, AuthBy FILE opens and reads the user database at start-up, reinitialisation and whenever the file’s modification time changes, (i.e. the database is cached within Radiator). Since the user database is cached in memory, large databases can require large amounts of memory.

AuthBy FILE supports a *`Nocache`* parameter that causes the user database to not be cached, and forces the file to be reread for every authentication. It will do a linear search for the user. You should be very careful about using this because it could be very slow for more than 1000 users or so. Also, authentication speed will depend

on the user's position in the file, and will be faster for users near the beginning of the file. If you need *Nocache* in a production setting, you should consider DBFILE instead.

When attempting to authenticate a user, AuthBy FILE will first compare all the check items associated with the user. It understands and handles check items. For more information, see Section 7.1. Check items on page 462.

If all the check items agree with the attributes in the Access-Request message, AuthBy FILE will reply with an Access-Accept message containing all the attributes given as reply attributes in the user database. Some reply attributes are given special handling. For more information, see Section 7.2. Reply items on page 480. If the user does not appear in the database, or if any check attribute does not match, an Access-Reject message is sent to the client.

<AuthBy FILE> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.35.1. Filename

This specifies the filename that holds the user database. The default value is **%D/users**, which means the file named `users` in DbDir. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21.

Here is an example of using *Filename*:

```
# user database is called rad_users in DbDir
Filename %D/rad_users
```

> **Tip**
>
> Do not use %U or %u (which are replaced by the user name) in *Filename*, otherwise the filename changes with every authentication.

3.35.2. GroupFilename

This defines the filename that holds the user group database. The default value is **%D/groups**, which means the file named `groups` in DbDir. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21.

For usage examples, see `goodies/authorize-group1.cfg` and `goodies/authorize-group2.cfg`.

3.35.3. Nocache

Disables caching of the user database, and forces Filename to be reread for every Authentication. If not set, AuthBy FILE will only reread the user database when the files modification time changes. Do not use this parameter unless you have to, because it can be very slow for any more than 1000 users or so. If you need think you need *Nocache*, you should consider DBFILE instead.

```
# Do not cache so we can do some simple testing
# without restarting the server all the time
Nocache
```

3.36. <AuthBy DBFILE>

AuthBy DBFILE authenticates users from a user database stored in a DBM file in the standard Merit DBM format. It is implemented in `AuthDBFILE.pm`. It does not log (but does reply to) accounting requests. For more

information about file formats, see Section 9.3. DBM user database on page 491. DBM files can be built from flat file user databases with the *builddbm* utility. For more information about *builddbm*, see Section 6.2. builddbm on page 458.

AuthBy DBFILE opens and reads the user database for every authentication. This means that if you change the user database DBM file, it will have an immediate effect. The DBM file is not locked when it is accessed.

When attempting to authenticate a user, AuthBy DBFILE will first compare all the check items associated with the user in the same way as <AuthBy FILE>. For more information, see Section 3.35. <AuthBy FILE> on page 181. If all those match the attributes in the Access-Request message, AuthBy DBFILE will reply with an Access-Accept message containing all the attributes given as reply attributes in the user database. If the user does not appear in the database, or if any check attribute does not match, an Access-Reject message is sent to the client.

<AuthBy DBFILE> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.36.1. Filename

Specifies the filename that holds the user database. Defaults to `%D/users`, i.e. files named `users.dir` and `users.pag` in DbDir. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21. Depending on the actual DBM module that Perl chooses, the database files may have other extensions.

```
# user database is called rad_users in DbDir
Filename %D/rad_users
```

3.36.2. DBType

By default, Radiator and Perl will choose the ‘best’ format of DBM file available to you, depending on which DBM modules are installed on your machine. You can override this choice by specifying DBType as the name of one of the DBM formats supported on your platform. The typical choices are:

- AnyDBM_File
- NDBM_File
- DB_File
- GDBM_File
- SDBM_File
- ODBM_File

but not may be available on your platform. The default is AnyDBM_File, which chooses the best available implementation.

```
# Force it to use DB_File
DBType DB_File
```

> **Tip**
>
> If you alter this, you will probably have to use the -t flag set to the same type on *builddbm* to force it to build a compatible database.

3.37. <AuthBy CDB>

<AuthBy CDB> authenticates users from a user database stored in a CDB database file. CDB is a fast, reliable, lightweight package for creating and reading constant databases. For more details about CDB, see cdb website [https://cr.yp.to/cdb.html]. It is implemented in `AuthCDB.pm`, which was contributed by Pedro Melo (melo@ip.pt). It does not log accounting requests but replies to them. To use <AuthBy CDB>, install the CDB_File package from CPAN. For more information, see Section 2.1.2. CPAN on page 3.

<AuthBy CDB> opens and reads the user database for every authentication. This means that if you change the user database CDB file, it has an immediate effect. The CDB file is not locked when it is accessed.

The CDB is indexed by user name and the value is the check items followed by a newline followed by the reply items.

When attempting to authenticate a user, <AuthBy CDB> first compares all the check items associated with the user in the same way as <AuthBy FILE> (for more information, see Section 3.35. <AuthBy FILE> on page 181). If all those match to the attributes in the Access-Request message, <AuthBy CDB> replies with an Access-Accept message containing all the attributes given as reply attributes in the user database. If the user does not appear in the database or if any check attribute does not match, an Access-Reject message is sent to the client.

<AuthBy CDB> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.37.1. Filename

Specifies the filename that holds the user database. Defaults to %D/users.cdb. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21.

```
# user database is called rad_users.cdb in DbDir
Filename %D/rad_users.cdb
```

3.38. <AuthBy GROUP>

<AuthBy GROUP> allows you to conveniently define and group multiple AuthBy clauses. It is implemented in `AuthGROUP.pm`. This is most useful where you need to be able to have multiple sets of authentication clauses, perhaps with different AuthByPolicy settings for each group. You can use an <AuthBy GROUP> (containing any number of AuthBy clauses) anywhere that a single AuthBy clause is permitted. <AuthBy GROUP> can be nested to any depth.

<AuthBy GROUP> will try each AuthBy method in turn until one of them either Accepts or Rejects the request. You can change this using AuthByPolicy. For more information, see Section 3.38.1. AuthByPolicy on page 185.

```
<AuthBy GROUP>
      AuthByPolicy ContinueUntilReject
      <AuthBy SQL>
            ...
      </AuthBy>
      <AuthBy DBM>
            ...
      </AuthBy>
      <AuthBy GROUP>
            AuthByPolicy ContinueUntilAccept
            RewriteUsername s/^(.+)$/cyb-$1/
            <AuthBy FILE>
```

```
                ...
            </AuthBy>
            <AuthBy FILE>
                ...
            </AuthBy>
        </AuthBy>
</AuthBy>
```

<AuthBy GROUP> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.38.1. AuthByPolicy

This parameter allows you to control the behaviour of multiple AuthBy clauses inside this *<AuthBy GROUP>*. This parameter is always available in *<Handler ...>* and *<Realm ...>* clauses. In particular, it allows you to specify under what conditions Radiator tries the next AuthBy clause. If you only have one AuthBy clause, *AuthByPolicy* is not relevant and is ignored.

You can specify more than one AuthBy clause for a single Realm, Handler, or *<AuthBy GROUP>*. The normal behaviour of Radiator is to try to authenticate with the first one. If that authentication method either Accepts or Rejects the request, then Radiator immediately sends a reply to the NAS. If the AuthBy ignores the request, then the next one is tried. That is the default behaviour, you can change it using **AuthByPolicy**. The permissible values of *AuthByPolicy* are:

- **ContinueWhileIgnore**

  This is the default. Continue trying to authenticate until either Accept, Challenge, or Reject.

- **ContinueUntilIgnore**

  Continue trying to authenticate until Ignore.

- **ContinueWhileAccept**

  Continue trying to authenticate as long as it is Accepted.

- **ContinueUntilAccept**

  Continue trying to authenticate until it is Accepted.

- **ContinueWhileChallenge**

  Continue trying to authenticate as long as it is Challenged.

- **ContinueUntilChallenge**

  Continue trying to authenticate until it is Challenged.

- **ContinueWhileReject**

  Continue trying to authenticate as long as it is Rejected.

- **ContinueUntilReject**

  Continue trying to authenticate until it is Rejected.

- **ContinueWhileAcceptOrChallenge**

  Continue trying to authenticate as long as it is either Accepted or Challenged.

- **ContinueUntilAcceptOrChallenge**

  Continue trying to authenticate until it is either Accepted or Challenged.

- **`ContinueUntilRejectOrChallenge`**

  Continue trying to authenticate until it is either Reject or Challenged.

- **`ContinueAlways`** Note: this is the same as any other value

  Always do every authentication method. Returns the result of the last one.

Here is an example of using *`AuthByPolicy`*:

```
# Authenticate with SQL, but if they are rejected
# fall back to a flat file
AuthByPolicy ContinueWhileReject
<AuthBy SQL>
      ....
</AuthBy>
<AuthBy FILE>
      ....
</AuthBy>
```

You can only have one *`AuthByPolicy`* parameter and it applies to all the AuthBy clauses. You cannot change it between AuthBy clauses.

Tip

`ContinueUntilAcceptOrChallenge` is the most useful one when using EAP requests in an *`<AuthBy GROUP>`* with multiple internal AuthBys.

3.38.2. RewriteUsername

This is an optional parameter. It enables you to alter the username in authentication and accounting requests. For more details and examples, see Section 8. Rewriting user names on page 484.

3.38.3. StripFromRequest

Strips the named attributes from the request before passing it to any lower authentication modules. The value is a comma separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.38.4. AddToRequest

Adds attributes to the request before passing it to any lower authentication modules. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.38.5. AddToRequestIfNotExist

Adds attributes to the request before passing it to any lower authentication modules. Unlike AddToRequest, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest and AddToRequestIfNotExist adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name if they are not there already
AddToRequestIfNotExist Calling-Station-Id=1,Login-IP-Host=%h
```

3.38.6. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *`HandleAcctStatusTypes`* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *`HandleAcctStatusTypes`*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.39. <AuthBy UNIX>

<AuthBy UNIX> authenticates users from a user database stored in a standard Unix password file or similar format. It is implemented in `AuthUNIX.pm`. It does not log (but does reply to) accounting requests. For more information about the file format, see Section 9.4. Unix password file on page 491. Since Unix password files only have encrypted passwords, <AuthBy UNIX> can not work with CHAP or MSCHAP authentication.

For performance reasons, <AuthBy UNIX> opens and reads the password and group files at start-up, reinitialisation and whenever the file modification times change, (i.e. they are cached within Radiator). Since these files are cached in memory, large password files can require large amounts of memory. If you set the Nocache parameter, the files will be reread for every authentication, and will not be cached internally (this can be slow if you have a large password or group files, and should rarely be necessary).

It is not necessary to be running on a Unix host in order to use <AuthBy UNIX>. It will work equally well on Windows and NT, but you are probably less likely to need it there.

By using the Match parameter you can also specify other file formats if you need to.

When attempting to authenticate a user, <AuthBy UNIX> will encrypt the password from the user and compare it to the one in the password file. If the encrypted passwords match, AuthBy UNIX will reply with an Access-Accept message. If the user does not appear in the password file, an Access-Reject message is sent to the client. <AuthBy UNIX> caches the password file and group file internally, and rereads the files when the modification time changes. If the Nocache parameter is set the password and group files will be reread for every authentication.

It is important to note that on its own, <AuthBy UNIX> does not implement check or reply items, and therefore can only be used for “Authenticate only” applications. However, you can use it in conjunction with another AuthBy module that does use check and reply items. For more information, see Section 7. Check and reply items on page 462. If you do this, you can also use the Group check item, which will check whether the user is a member of a group defined in the GroupFilename file.

Tip

You can use AddToReply to easily add standard reply items to all users authenticated by <AuthBy UNIX>. For more information see Section 3.14.20. AddToReply on page 106

<AuthBy UNIX> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.39.1. Filename

Specifies the filename of the password file. Defaults to `/etc/passwd`. The file name can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21.

Tip

On systems with shadow password files, such as Solaris, Filename can name the shadow file. In order to support this, Radiator must have permission to read the shadow file (this usually means it must run as root).

```
# password file is in /usr/local/etc/local_passwd
Filename /usr/local/etc/local_passwd
```

3.39.2. Match

This parameter allows you to use flat files with different formats to the standard Unix password format. Match is a regular expression that is expected to match and extract the user name, password and (optional) primary group ID fields from each line in the password file. The default extracts the first two colon separated fields as user name and password, followed by a UID, followed by an (optional) primary group ID (i.e. standard Unix password file format).

```
# fields are separated by vertical bar |
Match ^([^\|]*)\|([^\|]*)
```

Tip

The default Match expression is:

```
Match ^([^:]*):([^:]*):?[^:]*:?([^:]*)
```

3.39.3. GroupFilename

Specifies the name of the group file. The group file is in standard Unix group file format. Used to check "Group=" check items when authentication is cascaded from another module. Defaults to `/etc/group/`.

```
# group file is in /usr/local/etc/local_group
GroupFilename /usr/local/etc/local_group
```

3.39.4. Nocache

Disables caching of the password and group files, and forces them to be reread for every Authentication. If not set, AuthBy UNIX will only reread the files when their modification time changes. Do not use this parameter unless you have to, because it can be very slow for any more than 1000 users or so.

```
# Do not cache so we can do some simple testing
# without restarting the server all the time
Nocache
```

3.40. <AuthBy EXTERNAL>

<AuthBy EXTERNAL> passes all requests to an external program, which is responsible for determining how to handle the request. It is implemented in `AuthEXTERNAL.pm`.

3.40.1. Command

This parameter specifies the command to run. The command can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21. There is no default, and a Command must be specified. For more information about interpreting stdin, stdout, and exit status, see Section 3.40.2. Interpreting command stdin, stdout, and exit status on page 189.

```
# Interface to an external system
Command /usr/local/bin/doReq %T
```

3.40.2. Interpreting command stdin, stdout, and exit status

When the external command is run, all the attributes in the request will be formatted and passed to its standard input (stdin), one per line, in the format:

```
<tab> Attribute-Name = attribute_value
```

Each line output by the command on stdout is interpreted as a list of comma separated attribute-value pairs in the format:

```
Attribute-Name = attribute_value
```

and are returned in the reply message (if any). Any output lines that cannot be interpreted in that form are put in a Reply-Message attribute and returned in the reply message (if any). (This last behaviour is for backwards compatibility only and will not be supported indefinitely).

The exit status of the external command determines what type of reply is to be sent in response to the request:

- 0 means reply with an acceptance. For Access-Requests, an Access-Accept will be sent. For Accounting-Requests, an Accounting-Response will be sent.
- 1 means reply with a rejection. For Access-Requests, an Access-Reject is sent. For Accounting-Requests, no response is sent.
- 2 means do not send any reply. This will also make the Realm fall through to the next AuthBy module if you specified more than one for this Realm. For more information, see Section 3.38.1. AuthByPolicy on page 185.
- 3 means reply with an Access-Challenge for Access-Request. For Accounting- Requests, no response is sent.
- Any other value means that no reply is sent and no further action is taken.

<AuthBy EXTERNAL> will wait for the external process to complete before handling more requests, so you should use this carefully, and avoid using long-running commands. If you cannot avoid long-running EXTERNAL commands, you can use the Fork parameter to force <AuthBy EXTERNAL> to fork before calling the external command. This may improve performance.

<AuthBy EXTERNAL> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.40.3. DecryptPassword

This optional parameter makes AuthBy EXTERNAL decrypt the User-Password attribute before passing it to the external program. If you do not specify this, User-Password will be passed exactly as received in the request (i.e. encrypted by MD5 according to the RADIUS standard).

This is not able to decrypt CHAP or MSCHAP passwords.

```
# Pass plaintext passwords to the external program
DecryptPassword
```

3.40.4. ResultInOutput

If this optional parameter is set, AuthBy EXTERNAL will determine the type of the reply from the first line of the stdout output of the external program, rather than the exit code of the external program.

The following codes are recognised:

- ACCEPT

  Reply with an Access-Accept

- REJECT

  Reply with an Access-Reject

- IGNORE

  Do not reply

- CHALLENGE

  Reply with an Access-Challenge

- REJECT_IMMEDIATE

  Reply with an Access-Reject, and do not consider any other DEFAULT users in a chained Auth-Type request

3.41. <AuthBy SQL>

<AuthBy SQL> authenticates users from an SQL database, and stores accounting records to an SQL database. It is implemented in `AuthSQL.pm`. *<AuthBy SQL>* is very powerful and configurable, and it has many parameters in order to customise its behaviour. You need to have some familiarity with SQL and relational databases in order to configure and use *<AuthBy SQL>*.

<AuthBy SQL> uses the Perl DBI/DBD interface to connect to your database. You can therefore use *<AuthBy SQL>* with a large number of commercial and free SQL database systems. In order to use SQL, you will need to install your database software, install the matching Perl DBD module, and install the Perl DBI module before *<AuthBy SQL>* works.

When *<AuthBy SQL>* receives an Access-Request message, it tries to find a password and check and reply items for the user in a database table (you can change this behaviour with the *AuthColumnDef* parameter).

Radiator constructs an SQL select statement from the *AuthSelect* parameter. By changing *AuthSelect*, you can control the table it looks in, and the names of the columns for the password, check and reply columns. If the user is found, all the check items are compared with the attributes in the request, including *Expiration* and other special check items. For more information about check and reply items, see Section 7. Check and reply items on page 462.

If all the check items are satisfied by the attributes in the request, *<AuthBy SQL>* replies with an Access-Accept message containing all the attributes in the reply items. If the user does not appear in the database, or if any check attribute does not match, an Access-Reject message is sent to the client.

If your *AuthSelect* statement does not generate a simple password, check items, and reply items as a result, you can configure Radiator how to interpret the columns in the result with the *AuthColumnDef* parameter. If you do not specify any *AuthColumnDef* parameters, Radiator assumes that *AuthSelect* returns password, check items, and reply items in that order.

When *<AuthBy SQL>* receives an Accounting-Request message, it can store any number of the attributes from the request in an SQL table. You can control the table it stores in, and the names of the columns where the attributes are stored, and the attribute that is stored there. To enable SQL accounting you must define AccountingTable and you must define at least one *AcctColumnDef*. If you do not do both of these *<AuthBy SQL>* acknowledges Accounting-Request message but will not store them anywhere. The example `goodies/sql.cfg` in the Radiator distribution shows a typical setup that will work with the table schemas in `goodies/*Create.sql` files.

Some example scripts for constructing database tables for various RDBMSs can be found in the `goodies/*Create.sql` files in the Radiator distribution. Regard these as a starting point for constructing large scalable user and accounting databases.

<AuthBy SQL> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.41.1. AuthSelect

This is an SQL select statement that will be used to find and fetch the password and possibly check items and reply items for the user who is attempting to log in. You can use the special formatting characters. %0 is replaced with the quoted and escaped user name. The first column returned is expected to be the password; the second is the check items (if any) and the third is the reply items (if any) (you can change this expectation with the AuthColumnDef parameter). Defaults to "select PASSWORD from SUBSCRIBERS where USERNAME=%0", which does not return any check or reply items. You can make arbitrarily complicated SQL statements so that you will only authenticate users for example whose account status is OK or who have not exceeded their download limit etc. For information about how check items and reply items are used, see Section 7. Check and reply items on page 462. If the password (or encrypted password) column for a user is NULL in the database, then any password will be accepted for that user.

The password column may be in any of the formats described in Section 7.1.1. User-Password, Password on page 463.

```
# Check user status is current. No reply items in DB
# Note: The entire statement must be on one line
AuthSelect select PW, CHECK from USERS where NAME=%0 and STATUS = 1
```

If AuthSelect is defined as an empty string, SQL will not attempt to authenticate at all.

If one or more AuthSelectParam parameters are specified, they will be used in order to replace any fields marked with a question mark in AuthSelect.

Tip

By default, many SQL servers do case-insensitive string comparison. This means that AuthBy SQL, AuthBy RADMIN etc. would match, for example mikem, MIKEM and MiKeM as being the same user. Some SQL databases allow you to force case-sensitive comparisons. For example, In the case of MySQL, the 'BINARY' keyword forces the following comparison to be case-sensitive. Therefore you could force case-sensitive user names in an AuthSQL for MySQL with something like:

```
AuthSelect select PASSWORD from SUBSCRIBERS where BINARY USERNAME=%0
```

Tip

You can improve the performance of AuthSelect queries executed by the SQL server with AuthSelectParam.

Note

In the event of a SQL timeout, Radiator will reconnect to the database and the AuthSelect query will be tried again. This means there may be up to 2 Timeout intervals before the entire AuthSelect query fails.

3.41.2. AuthSelectParam

This optional parameter specifies a bind variable to be used with AuthSelect. See Section 3.8.1. SQL bind variables on page 47 for information about how to use bind variables.

3.41.3. AuthColumnDef

This optional parameter allows you to change the way Radiator interprets the result of the AuthSelect statement. If you do not specify any *AuthColumnDef* parameters, Radiator assumes that the first column returned is the password; the second is the check items (if any) and the third is the reply items (if any). If you specify any *AuthColumnDef* parameters, Radiator uses the column definitions you provide.

AuthColumnDef ignores the returned columns if their value is one of the following:

- NULL
- Empty value
- Single NULL octet value

You can specify any number of *AuthColumnDef* parameters, one for each interesting field returned by AuthSelect. The general format is:

```
AuthColumnDef n, attributename, type[, formatted]
```

- *n* is the index of the field in the result of AuthSelect. 0 is the first field.
- *attributename* is the name of the attribute to be checked or replied. The value of the attribute is in the nth field of the result. The special *attributename* "GENERIC" indicates that it is a list of comma separated *attribute=value* pairs.

- *type* indicates whether it is a *check* or *reply* item. A type of *request* sets the named attribute in the incoming request, from where it can be retrieved later in the authentication process with special formatting characters.
- *formatted*, if this keyword is present, the value retrieved from the database is subject to special character processing before its value is used, and can therefore contain *%{something}* forms which are replaced at authentication time. Reply items values are always formatted when they are added to a reply. Therefore there is typically no need to use this flag when the type is *reply*.

Example

The standard default AuthSelect statement is:

```
AuthSelect select PASSWORD from SUBSCRIBERS \
      where USERNAME=%0
```

This returns a single plain text password check item. The result can be interpreted with:

```
AuthColumnDef 0, User-Password, check
```

Example

Here is a more complicated AuthSelect statement:

```
AuthSelect select PASSWORD, CHECKATTR, REPLYATTR \
      from SUBSCRIBERS \
      where USERNAME=%0
```

The previous example returns 3 fields in the result. The first is a plain text password, the second is a string of check items like "Service-Type=Framed-User, Expiration="Feb 2 1999"", and the third field is a string of reply items like "Framed-Protocol=PPP,Framed-IP-Netmask = 255.255.255.0,OSC-Timestamp=%t,....". Special %-formats are expanded when a reply is constructed. This applies to all attributes, including those have special name "GENERIC". The result can be interpreted with:

```
AuthColumnDef 0, User-Password, check
AuthColumnDef 1, GENERIC, check
AuthColumnDef 2, GENERIC, reply
```

Note

If your PASSWORD column contains a Unix encrypted password and you are using *AuthColumnDef*, you need to set it like this:

```
AuthColumnDef 0, Encrypted-Password, check
```

Example

Here is an example AuthSelect statement:

```
AuthSelect select SERVICE, PASSWORD, MAXTIME
      from SUBSCRIBERS \
      where USERNAME=%0
```

This returns 3 fields in the result. The first is a Service-Type to check, the next is a plain text password and the last is the number of seconds to send back in Session-Timeout. The result can be interpreted with:

```
AuthColumnDef 0, Service-Type, check, formatted
AuthColumnDef 1, User-Password, check
AuthColumnDef 2, Session-Timeout, reply
```

In this example, column 0 is interpreted for special characters before being used as a check item for the Service-Type parameter.

Example

Here is an example of using NULL values for customising user authorisation:

```
AuthColumnDef 1, NAS-IP-Address, check
AuthColumnDef 2, Framed-IP-Address, reply
```

This allows you to restrict certain users so that they can only log in from a certain NAS. The unrestricted users have column 1 set to NULL. Likewise, users with static IP address have non-NULL value in column 2.

3.41.4. AccountingTable

This is the name of the table that will be used to store accounting records. Defaults to “ACCOUNTING”. If AccountingTable is defined to be an empty string, all accounting requests will be accepted and acknowledged, but no accounting data will be stored. You must also define at least one AcctColumnDef before accounting data will be stored.

The AccountingTable table name can contain special formatting characters: table names based on the current year and/or month might be useful, so you can rotate your accounting tables.

```
# store accounting records in RADUSAGEyyyymm table
AccountingTable RADUSAGE%Y%m
```

3.41.5. EncryptedPassword

This parameter should be set if and only if your AuthSelect statement will return a bare Unix encrypted password, and you are not using AuthColumnDef. Encrypted passwords cannot be used with CHAP or MSCHAP authentication. If the encrypted password column for a user is NULL in the database, then any password will be accepted for that user. If the encrypted password column for a user is set to the empty string (as opposed to NULL), then no password will be accepted for that user.

EncryptedPassword is a hint to Radiator about how to deal with passwords that have no explicit password type prefix. If EncryptedPassword is not set, bare passwords (i.e. without a prefix) are treated as plaintext passwords. If EncryptedPassword is set, they are treated as a Unix crypt password. If a password has a prefix, then the password will be interpreted according to that type of password, independent of the setting of EncryptedPassword. For more information, see Section 7.1.1. User-Password, Password on page 463.

Note

This parameter is ignored if you have defined your own AuthSelect column definitions with AuthColumnDef.

```
# unix Encrypted password are in CRYPTPW
AuthSelect select CRYPTPW from USERS where N = %0
EncryptedPassword
```

3.41.6. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *`HandleAcctStatusTypes`* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *`HandleAcctStatusTypes`*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.41.7. AccountingStartsOnly

If this parameter is defined, it forces AuthBy SQL to only log Accounting Start requests to the database. All other Accounting requests are accepted and acknowledged, but are not stored in the SQL database.

Note

This parameter is now deprecated and will not be supported in the future. Use Section 3.38.6. HandleAcctStatusTypes on page 187 instead.

Note

It does not make sense to set AccountingStartsOnly and AccountingStopsOnly.

3.41.8. AccountingAlivesOnly

If this parameter is defined, it forces AuthBy SQL to only log Accounting Alive requests to the database. All other Accounting requests are accepted and acknowledged, but are not stored in the SQL database.

Note

This parameter is now deprecated and will not be supported in the future. Use Section 3.38.6. HandleAcctStatusTypes on page 187 instead.

3.41.9. AccountingStopsOnly

If this parameter is defined, it forces AuthBy SQL to only log Accounting Stop requests to the database. All other Accounting requests are accepted and acknowledged, but are not stored in the SQL database.

Note

this parameter is now deprecated and will not be supported in the future. Use Section 3.38.6. HandleAcctStatusTypes on page 187 instead.

You may want to use this parameter if your accounting system does not use or need Accounting Start to do its billing.

```
# We only want Stops
AccountingStopsOnly
```

Note

It does not make sense to set AccountingStartsOnly and AccountingStopsOnly.

3.41.10. DateFormat

This optional parameter specifies the format to be used to format dates for insertion into the AccountingTable. All date formatting characters are permitted. For more information, see Section 3.4. Date formatting on page 26. Defaults to `%b %e, %Y %H:%M` which formats to, for example, `Sep 3, 1995 13:37`.

```
# Include seconds in dates in a way that MS-SQL likes
DateFormat %b %e, %Y %H:%M:%S
```

3.41.11. AcctColumnDef

AcctColumnDef is used to define which attributes in accounting requests are inserted into AccountingTable. It also specifies which column they are inserted into, and optionally the data type of that column. The general form is:

```
AcctColumnDef Column,Attribute[,Type][,Format]
```

`Column` is the name of the SQL column where the data is inserted. **`Attribute`** is the name of the RADIUS attribute to store there. **`Type`** is an optional data type specifier, which specifies the data type of the SQL column. **`Format`** is an optional format string that can be used to format the value. Columns and their values are included in accounting SQL statements in alphabetical order by column name.

The following **`Type`** names are recognised:

- **`integer`**

  The insertion is done as an integer data type. RADIUS attributes that have VALUE names defined are inserted as their integer RADIUS value.

- **`integer-date`**

  The attribute value is converted from Unix seconds to an SQL datetime string using the date formatting characters. For more information, see Section 3.4. Date formatting on page 26. The **`Format`** field is used as the date format (if it is present), otherwise the standard *`DateFormat`* on page 196 parameter for this AuthBy SQL is used (which defaults to the format 'Sep 3, 1995 13:37'). This is useful for inserting the Timestamp attribute as an SQL datetime type. The default is compatible with at least Microsoft SQL and Sybase datetime columns. If it is not suitable for your database, consider defining your own *`DateFormat`* parameter for this AuthBy SQL. The resulting value is quoted after conversion.

- **`formatted-date`**

  formatted-date is now deprecated, and new installations should use **`integer-date`** instead. It has a much wider range of formatting and date options.

  The attribute is converted by Perl `TimeDate` module `Date::Format` according to the format string. `TimeDate` is available from CPAN. For more information, see Section 2.1.2. CPAN on page 3. It

is most useful for SQL databases with unusual date formats, like Oracle. **formatted-date** is now only provided for historical reasons, and new installations should probably use **integer-date** in conjunction with *DateFormat* instead. The resulting value is not quoted after conversion.

- **formatted**

  The attribute field is processed looking for the special characters described in Section 3.3. Special formatters on page 21. If the resulting string is empty it is not inserted. This is useful for inserting data from other places besides the current request, such as a GlobalVar you have defined elsewhere, or from a data item that the previous AuthBy put in the current reply packet. The resulting data is quoted. See **literal** below to generate unquoted data.

- **literal**

  Similar to formatted, except that the resulting value is not quoted.

- **inet_aton**

  Converts a dotted quad IP address (such as 10.1.1.5) to a 32 bit unsigned integer.

- Anything else

  Any other type string causes the named RADIUS attribute to be inserted literally as a string. The resulting value is quoted.

You can use **formatted-date** to create date formats to suit your SQL database. This example inserts the Timestamp into an Oracle DATE or TIMESTAMP type column called TIME_STAMP:

```
AcctColumnDef TIME_STAMP,Timestamp,formatted-date,\
      to_date('%e %m %Y %H:%M:%S', 'DD MM YYYY HH24:MI:SS')
```

The insert statement is this:

```
insert into ACCOUNTING (TIME_STAMP, ......) values
(to_date('16 02 1999 16:40:02', 'DD MM YYYY HH24:MI:SS'), ....)
```

For types other than **formatted-date** and **integer-date**, the **format** field can be used to build custom values in your insert statement. This can be very useful to call SQL conversion functions on your data. If you specify a format, it is used as a sprintf-style format, where %s is replaced by your value.

If any named attribute is not present in the accounting request, nothing is inserted in the column for that value. The attribute doe not appear in the insert statement at all, and the SQL server's default value (usually NULL) is used for that column. With some SQL servers, you can change the default value to be used when a column is not specified in an insert statement.

You can have 0 or more *AcctColumnDef* lines, one for each attribute you want to store in the accounting table. If there are no *AcctColumnDef* lines, then the accounting table is not updated.

The attribute Timestamp is always available for insertion, and is set to the time the packet was received, adjusted by value of `Acct-Delay-Time` attribute (if present), as an integer number of seconds since midnight Jan 1, 1970 UTC. The Timestamp attribute is added by Radiator to all received Accounting requests, and is set to the current time according to the host on which the Radiator is running.

Here is an example column configuration:

```
AcctColumnDef USERNAME,User-Name
AcctColumnDef TIME_STAMP,Timestamp,integer
AcctColumnDef ACCTSTATUSTYPE,Acct-Status-Type
AcctColumnDef ACCTDELAYTIME,Acct-Delay-Time,integer
AcctColumnDef ACCTINPUTOCT,Acct-Input-Octets,integer
```

```
AcctColumnDef ACCTOUTPUTOCT,Acct-Output-Octets,integer
AcctColumnDef ACCTSESSIONID,Acct-Session-Id
AcctColumnDef ACCTSESSTIME,Acct-Session-Time,integer
AcctColumnDef ACCTTERMINATECAUSE,Acct_Terminate-Cause
AcctColumnDef NASIDENTIFIER,NAS-Identifier
AcctColumnDef NASPORT,NAS-Port,integer
# Insert date-time without and with seconds
DateFormat %Y-%m-%d %H:%M
AcctColumnDef DATE_TIME,Timestamp, integer-date
AcctColumnDef DATE_TIME_SEC,Timestamp, integer-date, %Y-%m-%d %H:%M:%S
```

Note

If your accounting table inserts are not working, run Radiator at a trace level of 4, and you see each insert statement logged before it is executed. This helps you determine if your *AcctColumnDef* lines are correct.

Note

If there are multiple definitions for the same column with non-null values, the last one in the configuration file is used.

Note

SQL table and column names are generally case sensitive, and usually can consist only of letters, digits or the underscore character '_'.

Note

You can further customise the accounting insert query with *AcctInsertQuery* on page 199.

Note

The **formatted** type is useful for inserting values set up in GlobalVars, or to get values from the current reply (possibly put there by a preceding AuthBy).

```
AcctColumnDef ACCOUNTTYPE,%{Reply:accounttype},formatted
AcctColumnDef SERVERNAME,%{GlobalVar:servername},formatted
```

Note

You can get SQL to calculate the start time of an accounting packet with something like:

```
AcctColumnDef START_TIME,%b-0%{Acct-Session-Time},literal
```

3.41.12. AuthSQLStatement

This parameter allows you to execute arbitrary SQL statements each time an authentication request is received, but before authentication is done.

You can have as many AuthSQLStatement parameters as you like (i.e. 0 or more). Each one will have its special formatting macros replaced at run time (the ones of the format %{attribute-name} are probably the most useful). %0 is replaced with the quoted and escaped user name. They are executed in the order they appear in the configuration file, before AuthSelect is run. They are run even if AuthSelect is empty.

3.41.13. AcctSQLStatement

This parameter allows you to execute arbitrary SQL statements each time an accounting request is received. You might want to do it to handle processing in addition to the normal inserts defined by AcctColumnDef, or you might want to construct a much more complicated SQL statement than AcctColumnDef can handle. You only need this if the accounting definitions provided by AcctColumnDef are not powerful enough.

You can have as many AcctSQLStatement parameters as you like (i.e. 0 or more). Each one will have its special formatting macros replaced at run time (the ones of the format %{attribute-name} are probably the most useful). %0 is replaced with the quoted and escaped user name. They are executed in the order they appear in the configuration file.

```
AcctSQLStatement delete from ONLINE where SessionID='%{Quote:%{Acct-Session-Id}}'
```

Note

By having multiple AuthBy SQL clauses, and by using AccountingStartsOnly and AccountingStopsOnly, in conjunction with AcctSQLStatement, you could implement a "who is online" table.

3.41.14. AcctInsertQuery

This optional parameter allows you to customise the exact form of the insert query used to insert accounting data. All date formatting characters are permitted. For more information, see Section 3.4. Date formatting on page 26. %0 is replaced with the value of the AccountingTable parameter. %1 is replaced with the comma separated list of column names to be inserted (derived from Acct- ColumnDef), and %2 is replaced by the data values to be inserted for each column in the same order as %1. The columns and values will be in alphabetical order by column name. The default is ‘insert into %0 (%1) values (%2)’.

You can use this to use special features in your SQL server, such as this for MySQL:

```
# maybe update if this is a duplicate
AcctInsertQuery update or insert into %0 (%1) values (%2)
```

3.41.15. AcctFailedLogFileName

The name of a file used to log failed Accounting-Request messages in the standard radius accounting log format. If the SQL database insert fails, the accounting message will be logged to the named file. For more information about file format, see Section 9.5. Accounting log file on page 491. If no AcctFailedLogFileName is defined, failed accounting messages will not be logged. The default is no logging. The file name can include special formatting characters as described in Section 3.3. Special formatters on page 21, which means that, for example, using the %c specifier, you can maintain separate accounting log files for each Client. The AcctFailedLogFileName file is always opened written and closed for each failed insertion, so you can safely rotate it at any time.

The user name that is recorded in the log file is the rewritten user name when RewriteUsername is enabled.

Note

You can change the logging format with AcctLogFileFormat.

Note

You may want to make the filename depend on the date, so you get one missed accounting file per day.

```
# Log all accounting to a single log file in LogDir
AcctFailedLogFileName %L/misseddetails
```

3.41.16. AcctLogFileFormat

This optional parameter is used to alter the format of the accounting log file from the standard radius format when AcctLogFileFormatHook is not defined. AcctLogFileFormat is a string containing special formatting characters. It specifies the format for each line to be printed to the accounting log file. A newline will be automatically appended. It is most useful if you use the %{attribute} style of formatting characters (to print the value of the attributes in the current packet).

```
AcctLogFileFormat %{Timestamp} %{Acct-Session-Id}\
%{User-Name}
```

3.41.17. AcctLogFileFormatHook

Specifies an optional Perl hook that will is used to alter the format of the failed accounting log file from the standard radius format when defined. The hook must return the formatted accounting record. A newline will be automatically appended. By default no hook is defined and AcctLogFileFormat or the default format is used. The hook parameter is the reference to the current request.

3.41.18. PostAuthSelectHook

This optional parameter allows you to define a Perl function that will be run during the authentication process. The hook will be called after the AuthSelect results have been received, and before Radiator has processed the attributes it is interested in. This hook runs only once even if the DB returns multiple rows. If there are no rows, the hook is not run.

The first argument passed to the hook is a handle to the current AuthBy SQL object. The second argument is the name of the user being authenticated. The third argument is a pointer to the current request. The fourth argument is a pointer to the User object being constructed to hold the check and reply items for the user being authenticated. The fifth argument ($_[4]) is a reference to the @row resulting from AuthSelect.

```
# Change the flag in the first field to 'Accept' or 'Reject'
PostAuthSelectHook sub{my($self,$name,$p,$user,$row)=@_;\
      my $flag = $row->[0];\
      $row->[0] = $flag ? 'Reject’:Accept';\
      }
```

3.41.19. GroupMembershipQuery

This optional parameter defines an SQL query which will be used to determine whether a user is a member of a group in order to implement the Group check item. Special characters are supported. %0 is replaced by the user name being checked. %1 is replaced by the group name being checked. You can use bound variables with GroupMembershipQueryParam. GroupMembershipQuery is expected to return a single row, where the first field is the name of the group the user belongs to.

3.41.20. GroupMembershipQueryParam

You can also use GroupMembershipQueryParam to provide bound variables for Group- MembershipQuery. %0 is replaced by the user name being checked. %1 is replaced by the group name being checked.

3.41.21. AcctTotalQuery

This optional parameter defines an SQL query which will be used to determine the total of all session times for a given user. Special characters are supported. %0 is replaced by the user name being checked. It is expected to return a single field containing the total session time in seconds.

It is used to get a count for the following check items:

- Max-All-Session

```
SELECT SUM(AcctSessionTime) FROM radacct WHERE UserName=%0
```

3.41.22. AcctTotalSinceQuery

This optional parameter defines an SQL query which will be used to determine the total of session times from a certain time until now for a given user. Special characters are supported. %0 is replaced by the user name being checked. %1 is replaced by the Unix epoch time in seconds in the start time of the query. It is expected to return a single field containing the total session time in seconds.

It is used to get a count for the following check items:

- Max-Hourly-Session
- Max-Daily-Session
- Max-Monthly-Session

3.41.23. AcctTotalOctetsSinceQuery

This optional parameter defines an SQL query which will be used to determine the total of octets from a certain time until now for a given user. Special characters are supported. %0 is replaced by the user name being checked. %1 is replaced by the Unix epoch time in seconds of the start time of the query. It is expected to return a single field containing the total octets.

It is used to get a count for the following check items:

- Max-Hourly-Octets
- Max-Daily-Octets
- Max-Monthly-Octets

3.41.24. AcctTotalOctetsQuery

This optional parameter defines an SQL query which will be used to determine the total of octets for a given user. Special characters are supported. %0 is replaced by the user name being checked. It is expected to return a single field containing the total octets.

It is used to get a count for the following check items:

- Max-All-Octets

3.41.25. AcctTotalGigawordsSinceQuery

This optional parameter defines an SQL query which will be used to determine the total of gigawords from a certain time until now for a given user. Special characters are supported. %0 is replaced by the user name being checked. %1 is replaced by the Unix epoch time in seconds of the start time of the query. It is expected to return a single field containing the total gigawords.

It is used to get a count for the following check items:

- Max-Hourly-Gigawords
- Max-Daily-Gigawords
- Max-Monthly-Gigawords

3.41.26. AcctTotalGigawordsQuery

This optional parameter defines an SQL query which will be used to determine the total of gigawords for a given user. Special characters are supported. %0 is replaced by the user name being checked. It is expected to return a single field containing the total gigawords.

It is used to get a count for the following check items:

- Max-All-Gigawords

3.41.27. NullPasswordMatchesAny

Normally, a NULL password in the SQL table matches any submitted password. If you disable this option, NULL passwords does not match any submitted password, causing every authentication request for that user to be rejected.

NullPasswordMatchesAny does not have any effect if one or more *AuthColumnDef*s are defined.

3.42. <AuthBy RADIUS>

<AuthBy RADIUS> forwards all authentication and accounting requests for this Realm to another (possibly remote) RADIUS server. This is called proxying. When the remote RADIUS server replies, the reply is forwarded back to the client that originally sent the request to the server.

This allows Radiator to act as a proxy RADIUS server, possibly running on the firewall of your organisation. You can also use it to set up roaming realms, or to make your RADIUS server act as a multiplexer for multiple realms. You can forward certain realms to other servers within your organisation in order to improve performance or redundancy.

By specifying multiple *Host* lines the requests can be forwarded to primary/secondary RADIUS server pairs. The normal behaviour is to try to forward the requests to the remote hosts in the order the hosts are listed in the configuration file. This means that it initially tries to forward a request to the first host listed (unless that host was marked as failed recently). If no reply is received from the first host (after *Retries*), it is marked as failed for *FailureBackoffTime* seconds, and the request is forwarded to the next *Host* in the list. This continues until a reply is received, or until all the hosts are tried. This way, if a primary remote server fails, the secondary takes over. After *FailureBackoffTime* seconds, the primary is tried again. To change this behaviour, choose other load balancing versions of *<AuthBy RADIUS>*. For more information, see Section 3.54. <AuthBy ROUNDROBIN>, <AuthBy VOLUMEBALANCE>, <AuthBy LOADBALANCE>, <AuthBy HASHBALANCE>, <AuthBy EAPBALANCE> on page 244.

Attention

By default, an <AuthBy RADIUS> clause is completed as soon as the request has been forwarded to the remote RADIUS server. In this case, <AuthBy RADIUS> returns IGNORE for AuthByPolicy, so take care when using sequences of AuthBy clauses that include <AuthBy RADIUS>. <AuthBy RADIUS> does not wait for a reply before moving on to other AuthBy clauses, or handling new requests. You can change this behaviour with one of these flag parameters: Asynchronous and Synchronous. Be cautious if you enable the Synchronous flag. It can have a significant impact on performance. If you need to continue processing the reply with another AuthBy clause, the correct way to do that is with a ReplyHook or Asynchronous flag that was added to Radiator 4.17.

Tip

If it is required to proxy RADIUS requests to a remote Radiator across an insecure or unreliable network such as the internet, consider using the RadSec protocol, and use <AuthBy RADSEC> instead of <AuthBy RADIUS>. RadSec provides encrypted, reliable transport of RADIUS requests to a remote Radiator. For more information, see Section 3.71. <AuthBy RADSEC> on page 288, Section 3.121. <ServerRADSEC> on page 418, and Section 16. RadSec (RFC 6614) on page 516.

Tip

Perform load balancing as well as simple proxying by using <AuthBy ROUNDROBIN>, <AuthBy VOLUMEBALANCE> or <AuthBy LOADBALANCE> instead of <AuthBy RADIUS>. For more information, see Section 3.54. <AuthBy ROUNDROBIN>, <AuthBy VOLUMEBALANCE>, <AuthBy LOADBALANCE>, <AuthBy HASHBALANCE>, <AuthBy EAPBALANCE> on page 244.

Tip

You can specify your remote RADIUS hosts either with Host parameters or <Host xxxxxx> clauses. For more information, see Section 3.42.2. Host on page 205 and Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220. The Host parameter is simpler but less flexible than the <Host xxxxxx> clause. For ease of understanding, do not mix formats in the same <AuthBy RADIUS>.

Tip

Be cautious when using Fork with <AuthBy RADIUS>. It usually results in requests being sent to the remote host, but the replies not received.

Example

In case it is not obvious from the above, there are 2 ways of specifying which remote RADIUS hosts to proxy to. The simplest, but the least flexible, way is to use Host parameters, and have the same secrets, ports etc. for all hosts:

```
<AuthBy RADIUS>
      # Same secret and timeout for all hosts
      Secret xyzzy
```

```
      RetryTimeout 2
      Host host1.bigco.com
      Host host2.bigco.com
</AuthBy>
```

Example

The other way allows you to customise some parameters on a host-by-host basis by using Host clauses:

```
<AuthBy RADIUS>
      # Same secret for all hosts
      Secret xyzzy
      <Host host1.bigco.com>
            # But a custom Timeout for this one
            RetryTimeout 2
      </Host>
      <Host host2.bigco.com>
            # And custom ports for this one
            AuthPort 1001
            AcctPort 1002
      </Host>
</AuthBy>
```

<AuthBy RADIUS> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.42.1. Failure algorithm

AuthBy RADIUS implements a configurable algorithm to detect failed RADIUS hosts, and to temporarily disregard failed hosts. The algorithm uses the MaxFailedRequests, MaxFailedGraceTime and FailureBackoffTime parameters to customise the operation of the algorithm. It also uses KeepaliveTimeout and UseStatusServerForFailureDetect in order to use only Status-Server requests for failure detection, instead of any request.

AuthBy RADIUS initially assumes that each Host is not failed. After a request is sent to a RADIUS server, if no reply is received after the ReplyTimeout, it is reset up to Retries times. If there is still no reply, that request is deemed to have failed for that Host. AuthBy RADIUS keeps track of how many consecutive requests failed for each Host since the last time a reply was heard from that Host. If more than MaxFailedRequests consecutive requests are deemed to have failed within MaxFailedGraceTime seconds of that last reply heard from that Host, that Host is deemed to have failed.

When the Host is deemed to be failed, AuthBy RADIUS will not attempt to send any requests to it until FailureBackoffTime seconds have elapsed. It will also skip sending requests to that host, and will instead attempt to send to the next Host in its list of Hosts (if any).

The default values for these parameters are:

```
Retries 3
RetryTimeout 5
MaxFailedRequests 1
MaxFailedGraceTime 0
FailureBackoffTime 0
```

These values mean that by default AuthBy RADIUS will declare the Host failed after a 3 retries packet transmission failure, but that it will always try to transmit the next request to the Host. This means that AuthBy

RADIUS will always try to send every request to the first Host, and if nothing is heard from that Host within (Retries * Retry-Timeout) seconds, it will attempt to send to the next Host.

Tip

Judicious use of these parameters allows you to implement a RADIUS Host fallback policy, where if one RADIUS Host fails to respond to requests, then it will automatically temporarily fall back to the next RADIUS Host and so on.

3.42.2. Host

This parameter specifies the host names where the destination RADIUS server is running. It can be either a DNS name or an IP address. Multiple comma-separated host names can be specified in one `Host` parameter, or you can use multiple `Host` lines. Radiator tries up to Retries times to contact each host that you specify. If no response it heard it tries the next host in the list and so on until a reply is received or the list is exhausted. The Host name can contain special formatting characters, which are resolved at startup.

Tip

If the DNS name for any Host resolves to multiple IP addresses, Radiator forwards to those addresses in a round-robin fashion. DNS names are resolved at startup time.

```
# Send all requests for this realm to 198.51.100.2, if no reply
# try the secondary at 198.51.100.3, if no reply from that,
# try all the addresses that radiushosts.example.com resolves to
# in round-robin fashion.
Host 198.51.100.2,198.51.100.3
Host radiushosts.example.com
```

Tip

For greater flexibility, you can also specify the hosts with the `<Host xxxxxx>` clause, which allows you to customise details for each host. For more information, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220.

```
# These are equivalent to the lines in the example above:
<Host 198.51.100.2>
      # Put host-specific values for Secret, ports,
      # retries etc. in here
</Host>
<Host 198.51.100.3>
</Host>
<Host radiushosts.example.com>
</Host>
```

Tip

In order to proxy to an IPv6 address, the first IPv6 address listed in LocalAddress is used as the source address. If LocalAddress does not contain any IPv6 address, then the default IPv6 source address for that host is used. A LocalAddress of '::' is equivalent to a locally allocated IPv6 address.

```
<AuthBy RADIUS>
      # send via IPv6. Packets will appear to come from
      # the default IPv6 source address for this host
      LocalAddress ::
      Host 2001:db8:100:f101::1
      Secret xxxxxx
      .....
</AuthBy>
```

Tip

IPv6 addresses are not required to be prefixed with 'ipv6:' with Radiator 4.13 or later.

3.42.3. Secret

The default value of the secret we share with the destination radius servers. Radiator acts like a RADIUS client when it forwards RADIUS request to another RADIUS server.

You must define a shared secret for each Host in AuthBy RADIUS, and it must match the secret configured into the destination RADIUS server. There is no default. The secret can be any number of ASCII characters. Any ASCII character except newline is permitted, but it might be easier if you restrict yourself to the printable characters. For a reasonable level of security, the Secret should be at least 16 characters, and a mixture of upper and lower case, digits and punctuation. You should not use just a single recognisable word. Can be overridden for an individual host inside its Host clause.

```
# This better agree with the server at
# eric.open.com.au or they wont understand us
<AuthBy RADIUS>
      Host eric.open.com.au
      Secret 666obaFGkmRNs666
</AuthBy>
```

CAUTION

Some NASs, notably Enterasys Smart Switch Routers support a maximum shared secret length of 16 characters.

3.42.4. AuthPort

Specifies the default UDP port on the destination Hosts to which Radiator will send authentication requests. The argument may be either a numeric port number or an alphanumeric service name as specified in /etc/services (or its moral equivalent on your system). The default port is 1645. Note that the officially assigned port number for RADIUS accounting has been changed to 1812. AuthPort may contain special formatting characters. Can be overridden for an individual host inside its Host clause.

```
# Send authentication to port 1812 on the remote server
AuthPort 1812
```

3.42.5. AcctPort

Specifies the default UDP port on the destination Hosts to which Radiator will send accounting requests. The argument may be either a numeric port number or an alphanumeric service name as specified in /etc/services (or its moral equivalent on your system). The default port is 1646. Note that the officially assigned port number for RADIUS accounting has been changed to 1813. AcctPort may contain special formatting characters. Can be overridden for an individual host inside its Host clause.

```
# Send accounting to port 1813 on the remote server
AcctPort 1813
```

3.42.6. OutPort

If this optional parameter is set, it forces a particular port number to be used for the forwarding port. Therefore if you have, for example:

```
OutPort 1001
```

Then all requests forwarded will appear to come from port 1001 on this Radiator host. This can be useful for implementing strict firewall rules. The argument may be either a numeric port number or an alphanumeric service name as specified in /etc/services (or its moral equivalent on your system). The default is to use a port number determined by your operating system, which will typically be different every time you restart your Radiator. OutPort may contain special formatting characters. A typical use of special formatting characters is with GlobalVar and command line arguments.

Tip

Many operating systems require root or administrator privileges to use a socket with a port number less than 1024.

3.42.7. Retries

If Radiator does not get a reply from the RADIUS server within defined time, it retransmits the request up to this number of retries. The default value is **3**, which means maximum of 4 transmissions. This can be overridden for an individual host inside its *`Host`* clause.

Here is an example of using *`Retries`*:

```
# It is a poor link, so lots of retries
Retries 10
```

3.42.8. RetryTimeout

Specifies the default number of seconds to wait for a reply before retransmitting. The default is 5 seconds, which is a common value for most RADIUS clients. If the destination RADIUS server is at the end of a distant or saturated link, you may want to set this to 10 or 20 seconds. Can be overridden for an individual host inside its Host clause.

```
# It is a poor link, wait 15 seconds before retransmission
RetryTimeout 15
```

3.42.9. KeepaliveTimeout

This optional integer specifies the maximum time in seconds that a RADIUS connection can be idle before a Status-Server request is sent. The default value is **0** and keepalives are not used. When *UseStatusServerForFailureDetect* is enabled, *KeepaliveTimeout* together with *MaxFailedRequests* and other related parameters defines the minimum time it takes to notice the next hop has failed.

3.42.10. KeepaliveNoreplyTimeout

This is an optional integer that specifies the waiting time, in seconds, for a Status-Server request. If no reply is received within the time *KeepaliveNoreplyTimeout* defines, the Status-Server request is marked as lost.

If *KeepaliveNoreplyTimeout* is not defined, the waiting time value depends on the AuthBy you are using:

- *<AuthBy RADCSEC>*: *NoreplyTimeout* value is used instead.
- *<AuthBy RADIUS>*: *RetryTimeout* value is used instead.

It is recommended to have a smaller value for *KeepaliveNoreplyTimeout* and a larger value for *NoreplyTimeout* or *RetryTimeout*. The Status-Server responder is always the next hop host and a reply is received quickly. With a short *KeepaliveNoreplyTimeout*, a possible failure situation is discovered quickly and the request is rerouted to another server. The final destination of an Access-Request or an Accounting-Request message may be located many hops away and for this reason a long *NoreplyTimeout* may be needed.

Here is an example of using both *KeepaliveNoreplyTimeout* and *NoreplyTimeout* in *<AuthBy RADSEC>*:

```
<AuthBy RADSEC>
      NoreplyTimeout 10
      KeepaliveNoreplyTimeout 3
</AuthBy>
```

To use this example with *<AuthBy RADIUS>*, you must use *RetryTimeout* instead of *NoreplyTimeout*.

3.42.11. KeepaliveRequestType

This string defines the type of RADIUS request that is sent as a keep-alive request. Any RADIUS request type is allowed. The default value is Status-Server.

Here is an example of using *KeepaliveRequestType* with *AddToKeepaliveRequest*:

```
# Send Access-Request as keepalive probe
KeepaliveRequestType Access-Request
AddToKeepaliveRequest User-Name=mikem,User-Password=fred
```

3.42.12. AddToKeepaliveRequest

This string adds attributes to the keep-alive request before sending it to Host. The value is a comma-separated list of attribute value pairs on one line, exactly as for any reply item. Special formatting characters are allowed, for more information, see Section 3.3. Special formatters on page 21.

Here is an example of using *AddToKeepaliveRequest* with *KeepaliveRequestType*:

```
# Send Access-Request as keepalive probe
KeepaliveRequestType Access-Request
AddToKeepaliveRequest User-Name=mikem,User-Password=fred
```

3.42.13. UseStatusServerForFailureDetect

If this optional flag is enabled, use only Status-Server requests (if any) to determine that the target server is failed when there is no reply. If this is not enabled, use no-reply to any type of request. This parameter uses *`NoreplyTimeout`*, *`MaxFailedRequests`*, and *`MaxFailedGraceTime`* during failure detection. This is not enabled by default.

`FailureBackoffTime` has no effect when *`UseStatusServerForFailureDetect`* is enabled. Once a Host has been marked as failed, it remains marked as failed until a reply is received from it (typically in response to a subsequent Status-Server request).

If you enable this, ensure that *`KeepaliveTimeout`* is set to a sensible interval to balance between detecting failures early and loading the target server.

3.42.14. FailureBackoffTime

This optional parameter specifies for how long time a failed remote server is removed from the forwarding list and marked as failed. The number of retries is defined by *`MaxFailedRequests`*. If no reply is received when all retries are used, the host is marked as failed and no requests are sent to it. After *`FailureBackoffTime`* time has expired, the Host is again eligible for forwarding. The unit is seconds, and the default value is **`0`**, which means that the host is always regarded as working.

3.42.15. MaxFailedRequests

This optional parameter specifies how many requests must fail to receive a reply before the remote radius server is marked as failed, and the FailureBackoffTime will be applied. The default is 1, which means that one ignored request will cause the Host to be marked as failed for FailureBackoffTime seconds.

3.42.16. MaxFailedGraceTime

This optional parameter specifies the time period in seconds over which *`MaxFailedRequests`* failures cause the target host to be assumed to be failed. The default value is **`0`**. After a host is declared to be failed, no request are forwarded to it until *`FailureBackoffTime`* seconds have elapsed. The default value lets the AuthBy to choose a reasonable value which is typically very large. Very short values require a high number of failures to set target host as failed. This value should only be changed in special cases.

3.42.17. HostStateChangeHook

This optional parameter allows you to define a Perl function that will be called when the reachability of a next hop Host changes. See section Section 3.42.1. Failure algorithm on page 204 for more about reachability detection. The following arguments are passed in the following order:

- Reference to the request that was sent to the Host but received no reply and triggered the change state. Request type is `Status-Server` when this request type is used for reachability detection.
- Reference to this `Radius::AuthRADSEC` structure
- Reference to the `Radius::Host` structure for the remote host
- New host state. One of integer constant values:
  - `$Radius::AuthRADIUS::HOST_STATE::REACHABLE`
  - `$Radius::AuthRADIUS::HOST_STATE::UNREACHABLE`
- IP address of the Host that is now reachable or unreachable
- Port on the Host that is now reachable or unreachable

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

HostStateChangeHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. The following example shows how to use the state constants, store the state over hook invocations and log information from the hook.

```
HostStateChangeHook sub { \
    my ($p, $self, $host, $new_state, $ip, $port) = @_; \

    my $old_state = $host->{state_change_hook_current_state}; \
    $old_state = $Radius::AuthRADIUS::HOST_STATE::REACHABLE
        unless defined $old_state; \

    return if $old_state == $new_state; \
    $host->{state_change_hook_current_state} = $new_state; \

    my $st = ($new_state == $Radius::AuthRADIUS::HOST_STATE::UNREACHABLE) ?
             'Unreachable' : 'Reachable'; \
    my $msg = "HostStateChangeHook: IP '$ip', Port '$port', New State: '$st'"; \
    $self->log($main::LOG_INFO, "$self->{log_class_identifier} $msg", $p); \
    return; }
```

3.42.18. MaxTargetHosts

This parameter limits the number of different hosts a request is proxied to in the case of no reply. The default value is **0**, which means there is no limit. If Radiator does not receive a reply from a host, it keeps trying until all hosts are exhausted.

3.42.19. StripFromRequest

Strips the named attributes from the request before forwarding it to any Host. The value is a comma separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.42.20. AddToRequest

Adds attributes to the request before forwarding to any Host. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.42.21. AddToRequestIfNotExist

Adds attributes to the request before forwarding to any Host. Unlike *AddToRequest*, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs. You can use any of the special % formats in the attribute values. There is no default.

```
# Possibly add our default Operator-Name
AddToRequestIfNotExist Operator-Name=1example.com
```

3.42.22. RequireMessageAuthenticator

AuthBy RADIUS clause checks the value of any *Message-Authenticator* attribute in incoming EAP or other replies. An incorrect authenticator causes the request to be ignored.

The optional *RequireMessageAuthenticator* flag causes the clause to require a correct *Message-Authenticator* attribute to be present in all incoming replies that support *Message-Authenticator* attribute. Most of the message types support *Message-Authenticator*. Accounting requests and responses do not support *Message-Authenticator*.

3.42.23. NoForwardAuthentication

Stops AuthBy RADIUS forwarding Authentication-Requests. They are ACCEPTED, but no further action is taken with them. This is different in meaning to IgnoreAuthentication, which IGNOREs them.

```
# Just ACCEPT Authentication-Requests, do not forward them
NoForwardAuthentication
```

3.42.24. NoForwardAccounting

Stops AuthBy RADIUS forwarding Accounting-Requests. They are ACCEPTED, but no further action is taken with them. This is different in meaning to IgnoreAccounting, which IGNOREs them.

```
# Just ACCEPT Accounting-Requests, do not forward them
NoForwardAccounting
```

3.42.25. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *HandleAcctStatusTypes* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *HandleAcctStatusTypes*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.42.26. LocalAddress

This optional parameter specifies the local address(es) to bind the proxy forwarding socket. This in turn specifies what the IP source address will be in forwarded requests. Defaults to BindAddress (which defaults to 0.0.0.0, i.e. the default source address). If no appropriate IPv4 or IPv6 address is found in LocalAddress, the default IPv4 or IPv6 source address for the host will be used. This parameter is usually only useful for

multi-homed hosts. If you do not understand what this is for, do not set it: the default behaviour is fine for most situations. If BindAddress or LocalAddress are set to multiple comma separated addresses, only the first IPv4 or IPv6 address (as appropriate) will be used for outgoing IPv4 or IPv6 addresses.

Tip

IPv6 addresses are not required to be prefixed with 'ipv6:' with Radiator 4.13 or later.

```
# We are multi-homed, bind the proxy port so forwarded requests
# appear to come from 203.53.154.27
LocalAddress 203.53.154.27
```

Tip

You can make forwarded IPv4 requests appear to come from one address, and IPv6 requests appear to come from a different address. In this example, requests will usually be forwarded to 2002:721:1500:1::a101, and will have a source address of 2001:720:1500:1::a100. If 2002:721:1500:1::a101 does not reply, then requests will be forwarded to 210.1.1.5 with a source address of 203.63.154.27.

```
LocalAddress 203.63.154.27,2001:720:1500:1::a100
Host 2002:721:1500:1::a101
Host 210.1.1.5
```

Note

AuthBy RADIUS will create and use a separate UDP network socket for each distinct source address used from the LocalAddress list.

3.42.27. ReplyHook

This optional parameter allows you to define a Perl function that will be called after a reply is received from the remote RADIUS server and before it is relayed back to the original client. The following arguments are passed in the following order:

- Reference to the reply received from the remote RADIUS server
- Reference to the reply packet being constructed for return to the NAS
- Reference to the original request from the NAS
- Reference to the request that was sent to the remote RADIUS server
- Reference to the Radius::Host structure for the remote host
- Reference to the redirected flag. If redirected is set by the hook, the ReplyHook has redirected the request to another AuthBy.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

The response type can be enforced when needed. For example, when the remote RADIUS server has rejected the request, the ReplyHook can do any local processing required for rejects and then change the response type to accept.

```
# Change RadiusResult in the 3rd argument, the original request
ReplyHook sub { ${$_[2]}->{RadiusResult} = $main::ACCEPT; }
```

ReplyHook Can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Fake a new attribute into the reply going back to the client
ReplyHook sub { ${$_[1]}->add_attr('test-attr', \
      'test-value');}
```

3.42.28. NoReplyHook

This optional parameter allows you to define a Perl function that will be called if no reply is received from any remote RADIUS server. A reference to the original request received from the NAS is passed as the first argument. A reference to the request that was forwarded to the remote RADIUS server is passed as the second argument. A reference to the reply packet being constructed for return to the NAS is passed as the third argument (note that the normal behaviour in case of no reply, is for no reply to be sent to the NAS).

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook will also be reported to the log file when your hook executes. Multiline hooks (i.e. with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore you should not use trailing comments in your hook.

NoReplyHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Call an AuthBy SQL to handle accounting that
# failed to get to the remote server
NoReplyHook sub { Radius::AuthGeneric::find('SQL')\
      ->handle_request(${$_[0]}, ${$_[2]});}
```

Tip

See NoReplyReject on page 214 parameter in case you need to return a reject back to the NAS. Setting this parameter may be quicker than creating a hook to trigger a reject.

3.42.29. ForwardHook

This is a Perl hook that is called once for each request before the request is forwarded to a remote RADIUS or RadSec server. It allows you to modify the forwarded request without changing the current one. *ForwardHook* receives the following arguments:

- Current request
- Forwarded request

Here is an example of using *ForwardHook*:

```
ForwardHook sub { my $p = $_[0]; my $fp = $_[1]; \
                  $fp->add_attr('OSC-AVPAIR', 'Added by ForwardHook'); }
```

3.42.30. ReplyTimeoutHook

This optional parameter allows you to define a Perl function that will be called if no reply is received from the currently tried remote RADIUS server. A reference to the original request received from the NAS is passed as the first argument. A reference to the request that was forwarded to the remote RADIUS server is passed as the second argument.

3.42.31. IgnoreReject

This optional parameter causes Radiator to ignore (i.e. not send back to the original NAS) any Access-Reject messages received from the remote RADIUS server. This is sometimes useful for authenticating from multiple RADIUS servers. However, you should note that if all the remote radius servers reject the request, then the NAS will receive no reply at all.

```
# If we get a reject from the remote, do not send it to the NAS
IgnoreReject
```

3.42.32. NoReplyReject

This is an optional flag parameter. When this parameter is set, it forces *<AuthBy RADIUS>* and its subclasses to return with result REJECT to trigger an Access-Reject when a proxied request times out. This parameter is not set by default.

When *NoReplyReject* is enabled, the reject reason is set to 'Upstream timeout'.

```
# Trigger an Access-Reject back to the NAS
NoReplyReject
```

Tip

NoReplyReject allows rejecting timed out requests without hooks such as *NoReplyHook*. For more information, see Section 3.42.28. NoReplyHook on page 213

3.42.33. Asynchronous

The default behaviour for *<AuthBy RADIUS>* is to return IGNORE as soon as the request has been forwarded to the remote RADIUS server. It does not wait for a reply before moving on to other AuthBy classes, or handling new requests. You can change this behaviour with the *Asynchronous* flag. This flag is recommended for all new configurations.

When you enable the *Asynchronous* flag, *Handler* continues to evaluate its AuthBy policy after a reply or timeout from the remote proxy. Other requests are processed while the reply is pending. *Asynchronous* is supported by *<AuthBy RADIUS>* and its subclasses.

For more information about how proxied requests are waited for by Radiator, see Section 3.42. <AuthBy RADIUS> on page 202 and Section 3.42.34. Synchronous on page 215.

```
# Auth to server1 and continue to AuthBy FILE if server 1 accepts.
# Process other requests while the reply from server 1 is pending.
<Handler>
      AuthByPolicy ContinueWhileAccept
      <AuthBy RADIUS>
            # Evaluate the policy when we get a reply or a timeout
            Asynchronous
```

```
            Host server1
            Secret xxxxxx
      </AuthBy>
      <AuthBy FILE>
            Filename %D/users
      </AuthBy>
</Handler>
```

3.42.34. Synchronous

The default behaviour for *<AuthBy RADIUS>* is to return IGNORE as soon as the request has been forwarded to the remote RADIUS server. It will not wait for a reply before moving on to other AuthBy classes, or handling new requests. You can change this behaviour with the *Synchronous* flag, but make sure you understand what you are doing before enabling the *Synchronous* flag.

If you enable the *Synchronous* flag, Radiator will wait for either a reply, or a timeout from the remote radius server before processing any following AuthBy clauses, or before handling any further requests. This means that handling requests will stop until a reply is received or the reply times out (which might take 15 seconds or more, depending on the settings of your *RetryTimeout* and retries parameters). This can seriously affect the performance of your RADIUS server, especially if the remote radius server is slow, stopped, or at the end of a slow or unreliable link. You should therefore be very cautious about setting this flag, and you should understand the consequences of remote server failure. The performance impact of the *Synchronous* flag can be alleviated by use of the *Fork* parameter (see Section 3.32.42. Fork on page 177) at the cost of significant increase in memory requirements.

Nevertheless, *Synchronous* can be very handy if you wish, for example, to forward a request to remote server only if another server REJECTS the request. See the example below for sample configuration.

```
# Auth to server2 only if server 1 rejects. Caution:
# accounting will normally go to server1, unless it rejects
<Realm xxxxxx>
      AuthByPolicy ContinueWhileReject
      <AuthBy RADIUS>
            # Wait here until we get a reply or a timeout
            Synchronous
            Host server1
            Secret xxxxxx
      </AuthBy>
      <AuthBy RADIUS>
            Host server2
            Secret yyy
      </AuthBy>
</Realm>
```

3.42.35. UseExtendedIds

This optional flag can be used to work around various problem that might arise with remote RADIUS servers in some circumstances.

In the standard RADIUS protocol;, the packet identifier is only 8 bits (0 to 255), which means that a RADIUS server can only have 256 requests pending from a given client at any time. This flag forces AuthBy RADIUS to use a much larger range of identifiers (at least 32 bits) carried in the Proxy-State attribute, meaning that many more requests can be pending at a given time, and that replies from a remote RADIUS server are more accurately matched to their original requests.

One such problem is flooding of remote servers by large number of new requests occurring at the same time, such as after a power failure in a large part of the city, resulting in lots of requests being proxied all at the same time.

Another problem is in the case of some types of remote server which do not send their replies from the same port and address to which they were sent.

Tip

The correct operation of this parameter requires that the remote RADIUS server honours the Proxy-State attribute correctly by replying it back to the sender exactly as it was sent. Most modern RADIUS server (including Radiator) behave correctly in this respect.

3.42.36. UseOldAscendPasswords

Attention

Deprecated. See Section 3.42.35. UseExtendedIds on page 215 instead.

This optional parameter tells Radiator to encode all passwords sent by this AuthBy using the old style (non RFC compliant) method that Ascend used to use on some NASs. The symptom that might indicate a need for this parameter is that passwords longer than 16 characters are not decoded properly. Can be overridden for an individual host inside its Host clause.

3.42.37. ServerHasBrokenPortNumbers

Attention

Deprecated. See Section 3.42.35. UseExtendedIds on page 215 instead.

Some RADIUS servers (GoRemote (GRIC) on NT in particular) exhibit broken behaviour in that the reply does not come from the same UDP port that the request was sent to! This broken behaviour would normally cause Radiator to ignore replies from such broken servers. The optional ServerHasBrokenPortNumbers flag will permit interoperation with such broken servers. Can be overridden for an individual host inside its Host clause.

CAUTION

Enabling this for two remote servers at the same host will cause difficult to trace problems.

```
# We are proxying to a GRIC server on NT
ServerHasBrokenPortNumbers
```

3.42.38. ServerHasBrokenAddresses

Some RADIUS servers (some rare accounting proxies) exhibit broken behaviour in that the reply does not come from the same address that the request was sent to! This broken behaviour would normally cause Radiator to ignore replies from such broken servers. The optional ServerHasBrokenAddresses flag will permit interoperation with such broken servers. Can be overridden for an individual host inside its Host clause.

WARNING

Enabling this for _two_ remote servers at the same host will cause difficult to trace problems.

```
ServerHasBrokenAddresses
```

3.42.39. CachePasswords

This parameter enables a user password cache in this AuthBy RADIUS. It allows proxying to be more robust when the remote server is not available. It can be very useful if the remote server is unreliable, or at the end of a slow, saturated or unreliable link. The exact behaviour of when the password cache is consulted is controlled by the CacheOnNoReply parameter.

Tip

Use of this parameter with a large user population can cause large amounts of memory use by the Radiator process.

Tip

If Radiator is restarted, the password cache is lost.

Note

Matching of cached passwords can never succeed for CHAP or MS-CHAP authentication requests.

3.42.40. CacheOnNoReply

This flag controls when the password cache created by the CachePasswords parameter is consulted.

If CacheOnNoReply is set (the default), then the Access-Request will always be proxied to the remote RADIUS server, and password cache will only be consulted if there is no reply from of any of the remote RADIUS servers. If no reply is received from any of the remote RADIUS servers, and If there is a cached reply that matches the password and has not exceeded the CachePasswordExpiry time limit, then the request will be accepted.

If CacheOnNoReply is not set, then the password cache will consulted before proxying. If there is a cached reply that matches the password and has not exceeded the CachePasswordExpiry time limit, then the request will be accepted immediately without being proxied to any remote RADIUS server.

3.42.41. IgnoreReplySignature

Deprecated. Normally, if a reply from a remote RADIUS server is received with a bad authenticator, the reply will be logged and then ignored. This optional parameter tells AuthBy RADIUS to ignore incorrect signatures in replies from remote RADIUS servers. Some RADIUS servers implement incorrect signature algorithms, and this flag will prevent problems when interoperating with such servers. Can be overridden for an individual host inside its Host clause.

CAUTION

Use of this flag can cause incorrect handling of replies in unusual circumstances.

You know you need this flag if you see an error like this, even when you are sure the shared secret for the RADIUS server is correct.

```
"Bad authenticator received in reply ....."
```

3.42.42. IgnoreAccountingResponse

This optional flag causes AuthBy RADIUS to ignore replies to accounting requests, instead of forwarding them back to the originating host. This can be used in conjunction with the AccountingHandled flag in a Handler or Realm (see Section 3.31.6. AccountingHandled on page 152) to ensure that every proxied accounting request is replied to immediately, and the eventual reply from the remote RADIUS server is dropped.

3.42.43. AcctFailedLogFileName

The name of a files used to log failed Accounting-Request messages in the standard radius accounting log format. If no reply is ever received from any of the remote hosts, the accounting message is logged to the named file. For more information about log file format, see Section 9.5. Accounting log file on page 491. If no *AcctFailedLogFileName* is defined, failed accounting messages are not logged. The default is no logging. The file name can include special formatting characters as described in Section 3.3. Special formatters on page 21, which means that, for example, using the %c specifier, you can maintain separate accounting log files for each Client. The *AcctFailedLogFileName* file is always opened, written and closed for each failure, so you can safely rotate it at any time.

The user name that is recorded in the log file is the rewritten user name when *RewriteUsername* is enabled.

Tip

You can change the logging format with *AcctLogFileFormat*.

Tip

You may want to make the filename depend on the date, so you get one missed accounting file per day.

```
# Log all accounting to a single log file in LogDir
AcctFailedLogFileName %L/misseddetails
```

3.42.44. AcctLogFileFormat

This optional parameter is used to alter the format of the failed accounting log file from the standard radius format when *AcctLogFileFormatHook* is not defined. *AcctLogFileFormat* is a string containing special formatting characters. It specifies the format for each line to be printed to the accounting log file. A new line is automatically appended. It is most useful if you use the **%{attribute}** style of formatting characters (to print the value of the attributes in the current packet.

```
AcctLogFileFormat %{Timestamp} %{Acct-Session-Id}\
%{User-Name}
```

3.42.45. AcctLogFileFormatHook

This specifies an optional Perl hook that is used to alter the format of the failed accounting log file from the standard radius format when defined. The hook must return the formatted accounting record. A new line is automatically appended. By default no hook is defined and *`AcctLogFileFormat`* or the default format is used. The hook parameter is the reference to the current request.

3.42.46. AccountingStartsOnly

If this parameter is defined, it forces AuthBy RADIUS to only forward Accounting Start requests. All other Accounting requests are accepted and acknowledged, but are not forwarded.

Tip

It does not make sense to set AccountingStartsOnly and AccountingStopsOnly.

3.42.47. AccountingAlivesOnly

If this parameter is defined, it forces AuthBy RADIUS to only forward Accounting Alive requests. All other Accounting requests are accepted and acknowledged, but are not forwarded.

3.42.48. AccountingStopsOnly

If this parameter is defined, it forces AuthBy RADIUS to only forward Accounting Stop requests. All other Accounting requests are accepted and acknowledged, but are not forwarded.

You may want to use this parameter if your accounting system does not use or need Accounting Start to do its billing.

```
# We only want to forward Stops
AccountingStopsOnly
```

Tip

It does not make sense to set AccountingStartsOnly and AccountingStopsOnly.

3.42.49. ClearTextTunnelPassword

This optional parameter prevents Radiator decrypting and re-encrypting Tunnel-Password attributes in replies during proxying. This is provided in order to support older NASs that do not support encrypted Tunnel-Password.

3.42.50. AllowInRequest

This optional parameter specifies a list of attribute names that are permitted in forwarded requests. Attributes whose names do not appear in this list will be stripped from the request before forwarding.

```
# Strip everything except username and password
AllowInRequest User-Name,User-Password
```

3.42.51. DisableMTUDiscovery

If this optional parameter is set, it disables MTU discovery on platforms that support that behaviour (currently Linux only). This can be used to prevent discarding of certain large RADIUS packet fragments on supporting operating systems.

3.42.52. Gossip

The *Gossip* flag parameter enables this AuthBy to send and listen for notifications (aka. gossip) to/from other Radiator servers when any remote Radius server fails to reply. This is disabled by default.

See Section 3.134. <GossipRedis> on page 439 for more about the Gossip framework and `goodies/farmsize.cfg` for a configuration sample.

```
# Use the configured Gossip implementation for notifications
Gossip
```

3.42.53. NoKeepaliveTimeoutForChildInstances

If this optional flag is enabled, only the first instance of this server farm will send Status-Server requests. Recommended when Gossip is used to distribute reachability information. This is not set by default.

3.42.54. GossipNoReply

If the *GossipNoReply* flag parameter is set, then a notification is sent when a remote Radius server fails to reply. Radiator server also increases a counter for failed requests when a notification is received. This is enabled by default.

3.42.55. GossipHostDown

If the *GossipHostDown* flag parameter is set, then a notification is sent when a remote Radius server is marked down. Radiator server also marks the remote Radius server down when a notification is received. This is enabled by default.

3.42.56. GossipHostUp

If the *GossipHostUp* flag parameter is set, then a notification will be send when a remote Radius server is marked up and alive again. Radiator server also marks the remote Radius server healthy when a notification is received. This is enabled by default.

3.43. <Host xxxxxx> within <AuthBy RADIUS>

This clause can be used to specify the name and details of remote RADIUS servers inside *<AuthBy RADIUS>* and its subtypes. The *<Host xxxxxx>* clause further allows you to customise details for individual hosts. *<AuthBy RADIUS>* permits one or more Host clauses.

In a Host clause header, the *xxxxxx* is the Host name or IP address of the remote RADIUS host to proxy to. The Host name can contain special formatting characters, which are resolved at startup.

```
<AuthBy RADIUS>
      <Host server1.test.com>
            Secret xyzzy
            AuthPort 1645
            AcctPort 1646
      </Host>
```

```
      <Host server2.test.com>
            Secret xyzzy
            AuthPort 1645
            AcctPort 1646
      </Host>
</AuthBy>
```

The following parameters can be used within a Host clause. They have the same meaning and default values as the parameter of the same name in the enclosing *<AuthBy RADIUS>*:

- Secret on page 206
- AuthPort on page 206
- AcctPort on page 207
- Retries on page 207
- RetryTimeout on page 207
- UseExtendedIds on page 215
- RequireMessageAuthenticator on page 211
- IgnoreReplySignature on page 217
- FailureBackoffTime on page 209
- MaxFailedRequests on page 209
- MaxFailedGraceTime on page 209
- LocalAddress on page 211
- OutPort on page 207
- UseOldAscendPasswords on page 216
- ServerHasBrokenPortNumbers on page 216
- ServerHasBrokenAddresses on page 216

3.43.1. BogoMips

<AuthBy LOADBALANCE> and *<AuthBy VOLUMEBALANCE>* use *BogoMips* to determine which target RADIUS server to use. The default value is **1**. If set to **0**, this *Host* is not used for forwarding by *<AuthBy RADIUS>* or any of its sub-types.

3.44. <AuthBy RADMIN>

<AuthBy RADMIN> provides authentication and accounting using the RAdmin User Administration package from Radiator Software website [https://radiatorsoftware.com/products/radmin/]. RAdmin is a complete web-based package that allows you to maintain your RADIUS user and accounting details in an SQL database. You can add, change and delete users, examine connection history, control simultaneous login, get reports on modem usage and many other functions. The combination of Radiator and RAdmin provides a complete solution to your RADIUS user administration requirements.

Tip

RAdmin is not a billing or invoicing system.

During authentication, Radiator checks the password in the RAdmin “RADUSERS” table. Accounting details are added to the RADUSAGE table.

There is an example Radiator configuration file for RAdmin in `goodies/radmin.cfg`.

<AuthBy RADMIN> understands also the same parameters as <AuthBy SQL>. For more information, see Section 3.41. <AuthBy SQL> on page 190.

3.44.1. AuthSelect

This SQL query is used to select details of users who are attempting to log in. %0 is replaced by the quoted and (possibly rewritten) User-Name. Other special formatting characters may be used.

Defaults to

```
select PASS_WORD, STATICADDRESS, TIMELEFT, MAXLOGINS, \
SERVICENAME, BADLOGINS, VALIDFROM, VALIDTO from RADUSERS \
where USERNAME=%0
```

Tip

You can force AuthBy RADMIN to honour additional fields in your AuthSelect statement by using AuthColumnDef. For example, you might add 3 new columns to your RADUSERS table and wish to use them as reply items. You could do that something like this:

```
# Honour FRAMED_NETMASK,FRAMED_FILTER_ID,MAXIDLETIME too
AuthSelect select PASS_WORD,STATICADDRESS,TIMELEFT,\
                  MAXLOGINS,SERVICENAME, BADLOGINS, VALIDFROM, \
                  VALIDTO, FRAMED_NETMASK,FRAMED_FILTER_ID,MAXIDLETIME \
           from RADUSERS where USERNAME=%0
AuthColumnDef 0,Framed-IP-Netmask,reply
AuthColumnDef 1,Filter-Id,reply
AuthColumnDef 2,Idle-Timeout,reply
```

Note that the numbering of AuthColumnDef 0 starts with the field following the first 8 minimum and required fields.

3.44.2. LogQuery

This optional parameter allows you to control the SQL query that is used to insert log messages into the database.

The default is:

```
insert into RADMESSAGES (TIME_STAMP, TYPE, MESSAGE)
values (%t, %0, %1)
```

Where %t is translated as a special character to the current time, %0 is converted to the message priority (in integer in the range 0 to 4 inclusive), and %1 is converted to the log message, quoted and escaped. MESSAGE will be truncated to MaxMessageLength characters prior to insertion.

3.44.3. MaxMessageLength

The optional parameter sets the maximum length of message that will be inserted by LogQuery. All messages longer than MaxMessageLength characters will be truncated to MaxMessageLength. Defaults to 200, which is the default size of the MESSAGE column in the RADMIN.RADMESSAGES table.

3.44.4. UserAttrQuery

This optional parameter allows you to control the query used to get user-specific RADIUS check and reply items. %0 is replaced by the (possibly rewritten) User-Name. Other special formatting characters may be used.

Defaults to

```
select ATTR_ID, VENDOR_ID, IVALUE, SVALUE, ITEM_TYPE \
from RADCONFIG where NAME=%0 order by ITEM_TYPE
```

3.44.5. ServiceAttrQuery

This optional parameter allows you to control the query used to get service-specific RADIUS check and reply items. %0 is replaced by the Service Profile name from the SERVICENAME column in the user's database record. Other special formatting characters may be used. ServiceAttrQuery will be run after UserAttrQuery if ServiceAttrQuery is non-empty, and if a non-empty servicename was found in the 5th field returned from AuthSelect.

Defaults to

```
select ATTR_ID, VENDOR_ID, IVALUE, SVALUE, ITEM_TYPE \
from RADSTCONFIG where NAME=%0 order by ITEM_TYPE
```

3.44.6. AttrQueryParam

This optional parameter enables the use of bound variables (where supported by the SQL server) and query caching in the UserAttrQuery and ServiceAttrQuery strings. If you specify one or more AttrQueryParam parameters, they will be used in order to replace parameters named with a question mark (“?”) in the UserAttrQuery and Service- AttrQuery queries, and the query will be cached for future reuse by the SQL server. %0 is replaced by the appropriate user name or service name. For more information, see Section 3.8.1. SQL bind variables on page 47.

3.44.7. IncrementBadloginsQuery

This optional parameter specifies the SQL query to issue if AuthBy RADMIN detects a bad password. It is intended to increment a count of the number of bad logins, which can then be checked during authentication. %0 is replaced with the name of the user being authenticated. Other special formatting characters may be used.

Defaults to:

```
update RADUSERS set BADLOGINS=BADLOGINS+1 where USERNAME=%0
```

3.44.8. ClearBadloginsQuery

This optional parameter specifies the SQL query to issue if AuthBy RADMIN detects a good password. It is intended to clear a count of the number of bad logins, which can then be checked during authentication. %0 is replaced with the name of the user being authenticated. Other special formatting characters may be used.

Defaults to:

```
update RADUSERS set BADLOGINS=0 where USERNAME=%0
```

3.44.9. MaxBadLogins

AuthBy RADMIN compares the bad login count in the RAdmin database with Max- BadLogins. If it is exceeded, it is assumed that password guessing has been attempted and the user will be disabled until their bad login count is reset. Defaults to 5. If set to 0, the bad login count is ignored.

3.45. <AuthBy EMERALD4>

<AuthBy EMERALD4> authenticates users from the Emerald version 4 database from IEA website [http://www.iea-software.com/]. It also stores accounting and logs to the Emerald 4 SQL database.

Tip

There is a sample Radiator configuration file for Emerald 4 in `goodies/emerald4.cfg` in your Radiator distribution.

NOTICE

<AuthBy EMERALD> for Emerald versions older than Emerald 4 is still distributed with Radiator but will be removed in a future Radiator release.

<AuthBy EMERALD4> understands also the same parameters as *<AuthBy SQL>*. For more information, see Section 3.41. <AuthBy SQL> on page 190.

3.45.1. AuthSelect

AuthSelect This optional parameter from AuthBy SQL is pre-configured or Emerald 4 and need not be altered. Defaults to

```
AuthSelect exec RadGetUser %0, NULL
```

3.45.2. ConcurrencyControl

This optional flag controls whether to apply Simultaneous-Use limits to each user. Defaults to false.

3.45.3. TimeBanking

This optional flag control whether Time Banking limits are to be enforced. Defaults to false.

3.45.4. HonourServers

This optional flag controls whether this module will use the Emerald Servers list as an additional source of Radius Client addresses. ClientQuery is used to fetch the client list. Defaults to false.

3.45.5. HonourServerPortAccess

This optional flag controls whether this module will enforce time-based access limits for certain NAS ports. PortAccessQuery is used to fetch details of port restrictions. Defaults to false.

3.45.6. HonourDNISGroups

This optional flag controls whether this module will enforce limits on Called-Station-Id. DNISGroupQuery is used to fetch details of permitted groups. Defaults to false.

3.45.7. HonourRoamServers

This optional flag Controls whether RoamQuery will be used to get roaming restrictions. Defaults to false.

3.45.8. DNISGroupQuery

SQL query used to fetch DNIS Groups. This parameter is preconfigured for Emerald 4, and should not need to be changed. Defaults to:

```
DNISGroupQuery select dn.DNISNumber from AccountTypes a, \
DNISNumbers dn where a.AccountType="%0" and \
a.DNISGroupID=dn.DNISGroupID \
and dn.DNISNumber="%1"
```

3.45.9. PortAccessQuery

SQL query used to fetch permitted ports This parameter is preconfigured for Emerald 4, and should not need to be changed. Defaults to:

```
PortAccessQuery select sa.StartTime, sa.StopTime, \
sa.MaxSessionLength from Servers s, ServerAccess sa, \
AccountTypes at\
where s.IPAddress="%{Client:Name}" \
and s.ServerID = sa.ServerID \
and (sa.Port=%0 or sa.Port=9214328)\
and sa.AccountTypeID=at.AccountTypeID\
and at.AccountType="%1"\
order by sa.Port
```

3.45.10. RadUserQuery

SQL query used to fetch user attributes This parameter is preconfigured for Emerald 4, and should not need to be changed. Defaults to:

```
RadUserQuery exec RadGetConfigs %0
```

3.45.11. ClientQuery

SQL query used to fetch RADIUS client details This parameter is preconfigured for Emerald 4, and should not need to be changed. Defaults to:

```
ClientQuery select IPAddress, Secret, Community, ServerID \
from Servers
```

3.45.12. RoamQuery

SQL query used to fetch roaming restrictions This parameter is preconfigured for Emerald 4, and should not need to be changed. Defaults to:

```
RoamQuery exec RadRoamCache
```

3.46. <AuthBy PLATYPUS>

<AuthBy PLATYPUS> provides authentication and accounting using the popular Platypus ISP billing package from Platypus website [http://www.ispbilling.com/]. The combination of Radiator and Platypus provides a very powerful and easy to use ISP billing and user management system.

Important

<AuthBy PLATYPUS> is not compatible nor needed with Platypus 7. See `platypus7.cfg` in `goodies` directory for a configuration sample to use with Platypus 7.

NOTICE

<AuthBy PLATYPUS> is obsolete and it will be removed in a future Radiator release. This does not affect Platypus 7 support in Radiator.

3.47. <AuthBy LDAP2>

<AuthBy LDAP2> module authenticates by issuing requests to an LDAP server. When the LDAP server replies, Radiator fetches a number of attributes and looks in them for the password, check items and reply items in order to authenticate the user. It does not log (but does reply to) accounting requests. You need to have a basic understanding of LDAP servers and databases in order to configure *<AuthBy LDAP2>*.

When an *<AuthBy LDAP2>* module receives its first authentication request, it attempts to connect to the LDAP server specified by Host. Optionally you can authenticate Radiator as a valid user of the LDAP server by specifying *AuthDN* and *AuthPassword*. This is not the same thing as authenticating a user. It happens before authenticating a user, and proves that this *radiusd* is allowed to talk to the LDAP database.

The *<AuthBy LDAP2>* module tries then to fetch some attributes for the user. Specify the base DN to start looking in, and the attribute name with which to filter. Also specify the attributes that contain the password, and (optionally) the names of the attributes containing an encrypted password, RADIUS check items and RADIUS reply items. This scheme allows you to work with almost any LDAP schema. All you have to do is identify the right LDAP attribute names.

If all the check items are satisfied by the attributes in the request, the *<AuthBy LDAP2>* module replies with an Access-Accept message containing all the attributes in the reply items attribute (if any). If the user does not appear in the LDAP database, or if any check attribute does not match, an Access-Reject message is sent to the client.

At present, *<AuthBy LDAP2>* modules do synchronous connections and searches. This can mean significant delays if your LDAP server is reached by a slow network connection, or your LDAP server is slow. In this case, consider putting the *<AuthBy LDAP2>* realm in a sub-server, and having your main Radiator forward requests for that realm to the RADIUS sub-server.

This clause supports all the common LDAP configuration parameters. For more information about the LDAP configuration parameters, see Section 3.9. LDAP configuration on page 53.

<AuthBy LDAP2> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.47.1. Using <AuthBy LDAP2> with Microsoft Active Directory

When using *<AuthBy LDAP2>* together with Microsoft Active Directory (AD), you may need to try the following:

- Use *ServerChecksPassword* when the user must be authenticated. AD does not provide password to LDAP. In this case, do not use *HoldserverConnection*. For more information, see Section 3.47.10. ServerChecksPassword on page 231.

- Leave *BaseDN* empty if you use Global Catalog. For more information, see Global Catalog and LDAP Searches [https://technet.microsoft.com/en-us/library/cc978012.aspx].
- Global Catalog contains all users but not necessarily not all the attributes. Use port 3268 for LDAP and port 3269 for LDAPS to access to Global Catalog.
- Use *AttrsWithBaseScope* if you need to get a constructed attribute, such as *tokenGroups*, for a certain user. For more information, see Section 3.47.8. AttrsWithBaseScope on page 230.

3.47.2. BaseDN

This is the base DN, where searches are made. It is used in similar way as with all LDAP modules. For more information, see Section 3.9.1. BaseDN on page 53.

Special formatting characters are permitted. When using *<AuthBy LDAP2>*, **%0** is replaced by *UsernameAttr* and **%1** by the user name.

Tip

On some LDAP servers, you can get a significant performance increase by narrowing the search to the exact entry you are interested in. With AuthBy LDAP2 default settings, this example restricts the search to **uid=username,ou=foo,o=bar,c=au**:

```
BaseDN      %0=%1,ou=foo,o=bar,c=au
Scope       base
```

3.47.3. SearchFilter

This is the LDAP filter to use when searching for the user. It is used in similar way as with all LDAP modules. For more information, see Section 3.9.2. SearchFilter on page 54.

Usually, the search filter that is used to find a matching user name is

```
(uid=name)
```

uid is the name of the LDAP attribute defined by the *UsernameAttr* parameter, and *name* is the name of the user currently being authenticated. For advanced applications, you can completely alter the search filter that Radiator uses by using the optional *SearchFilter* parameter. It allows you to use arbitrarily complicated LDAP search filters to select or exclude users based on attributes other than their user name. Special formatting characters are permitted, **%0** is replaced by *UsernameAttr* and **%1** by the user name. For example, this *SearchFilter* matches only users with the appropriate setting of their ‘current’ attribute:

```
SearchFilter (&(current=1)(uid=%1))
```

In *SearchFilter*, you an use any special formatting character. For backwards compatibility, Perl variables used to be interpolated, but this has been removed. The default setting for *SearchFilter* is **(%0=%1)**, which matches the user name against the LDAP attribute defined by the *UsernameAttr* parameter (usually *uid*). Therefore, the default search string is **(uid=name)**.

3.47.4. EncryptedPasswordAttr

This is the optional name of the LDAP attribute that contains a Unix crypt(3) encrypted password for the user. If you specify EncryptedPasswordAttr, it will be used instead of PasswordAttr, and PasswordAttr will not be fetched. You must specify either PasswordAttr or EncryptedPasswordAttr. There is no default.

Tip

If your passwords are in the form `{crypt}1xMKc0GIVUNbE`, `{SHA}0DPiKuNIrrVmD8IUCuw1hQxNqZc=` or `{SSHA}0DPiKuNIrrVmD8IUCuw1hQxNqZc=` you should be using PasswordAttr, not EncryptedPasswordAttr. Only use EncryptedPasswordAttr if the your password are plain old Unix crypt format, like: `1xMKc0GIVUNbE`.

```
# Get the encrypted password from cryptpw
EncryptedPasswordAttr cryptpw
```

3.47.5. PasswordAttr

This is the name of the LDAP attribute that contains the password for the user. The password may be in any of the formats supported by User-Password. For more information, see Section 7.1.1. User-Password, Password on page 463. Most LDAP servers will only have a plaintext password if they are secured in another way, and probably not even then. You must specify either PasswordAttr or EncryptedPasswordAttr. There is no default.

Note

OpenLDAP's userPassword is (a) encrypted and (b) only retrievable via an appropriately authenticated binding to the slapd.

```
# Plaintext passwords. Gasp
PasswordAttr passwd
```

Tip

If there is no password to be checked (e.g. Wireless MAC Addresses) you should specify PasswordAttr without a value, otherwise you will get a warning log message.

CAUTION

The value of currently configured PasswordAttr is printed as **obscured** when the attributes received from the LDAP server are logged. If the PasswordAttr is not present, nothing is logged for it. That is, only the value is obscured not the information about the presence of the attribute in the reply. To debug the password, use the Debug configuration parameter and see the console output. Another option is to configure “PasswordLogFileName” for the enclosing Handler. For more information, see Section 9.6. Password log file on page 492.

3.47.6. UsernameAttr

This is the name of the LDAP attribute that is required to match the user name in the authentication request (possibly after user name rewriting by RewriteUsername). Defaults to “uid”. The LDAP search filter is constructed from UsernameAttr and the name of the user in the Access-Request like this: “(UsernameAttr = user name)” For example, if you UsernameAttr is “uid”, and user “mikem” tries to log in, Radiator will use the LDAP filter “(uid=mikem)”. This behaviour can be customised with the Search-Filter parameter.

```
# Use the uid attribute to match usernames
```

```
UsernameAttr uid
```

3.47.7. AuthAttrDef

This optional parameter allows you to specify LDAP attributes to use as general check and reply items during authentication. *AuthAttrDef* is more general and useful than *CheckAttr* and *ReplyAttr*, and should be used in preference to them.

Using AuthAttrDef you can specify multiple LDAP attributes and tell Radiator to use them as check or reply items during authentication.

You can specify any number of *AuthAttrDef* parameters, one for each check or reply attribute in your LDAP database. The general format is:

```
AuthAttrDef ldapattributename[, radiusattributename, type[, formatted]]
```

- **ldapattributename** is the name of the LDAP attribute that is used as the check or reply item. If it is multi-valued, and this is a check item, then Radiator permits a match with any one of the values.
- **radiusattributename** is the name of the RADIUS attribute that is used as the check or reply item. The special radiusattributename ‘GENERIC’ indicates that it is a list of comma separated attribute=value pairs, similar to ReplyAttr or CheckAttr.
- **type** indicates whether it is a check or reply item. It consists of the word “check” or the word “reply”. If type is “request” the value is saved in the current request, from where it can be later collected with a special formatting macro like: *%{attributename}*.
- **formatted** indicates that the LDAP attributes are to be subject to special character processing before being used.

Tip

The **radiusattributename** and **type** fields are optional. If they are not specified, then the **ldapattributename** attribute is fetched from LDAP, but the fetched value of that attribute is not used. This can be helpful avoid receiving all attributes from the server, or with for some types of LDAP query where they can be used with *PostSearchHook* [PostSearchHook_AuthByLDAP2.dita#PostSearchHook_AuthByLDAP2] on page 230.

Example

Fetch the LDAP attribute called *calledstationid*, and use it as a check item against the RADIUS Called-Station-Id

```
AuthAttrDef       calledstationid,Called-Station-Id,check
```

Example

Check the RADIUS Service-Type matches the LDAP attribute called *servicetype*, and return the LDAP attribute called *address* as a static IP address (after special character replacements):

```
AuthAttrDef       servicetype,Service-Type,check
AuthAttrDef       address,Framed-IP-Address,reply,formatted
```

Example

During LDAP authentication, save 2 LDAP attributes into the current request:

```
# Put poolhint attribute into the request:
AuthAttrDef radiusUserPoolHint, X-userPoolHint, request
# Put Group Name attribute into the Request:
AuthAttrDef radiusSimultaneousUseGroupName,X-GroupName, request
```

Then use those attributes in a later AuthBy FILE:

```
fred        Group=%{X-GroupName}
            PoolHint=%{X-userPoolHint}
```

Example

Check items handle multi-valued LDAP attributes in a special way: by permitting a match with any one of the multiple values. For example, suppose you had *callingstation* LDAP attribute that could be multi-valued, and into which you put all the numbers the user was permitted to call from (as a separate value for each number), then you would use:

```
AuthAttrDef callingstation,Calling-Station-Id,check
```

3.47.8. AttrsWithBaseScope

Tells Radiator to search first for the user DN then do a search with scope base to fetch the attributes. Required, for example, to get access to Windows AD constructed attributes, such as tokenGroups, which are only returned when the search scope is set to base. Defaults to off.

3.47.9. PostSearchHook

This optional parameter allows you to define a Perl function that is run during the authentication process. The hook is called after the LDAP search results have been received, and after Radiator has processed the attributes it is interested in. Hook authors can use LDAP library routines to extract other attributes and process them in any way. *PostSearchHook* is called once for each LDAP result, as governed by *MaxRecords* parameter. If there are no results, the hook is not run. See Section 3.47.13. MaxRecords on page 232.

PostSearchHook has the following arguments:

- Handle to the current AuthBy object
- User name
- Pointer to the current request
- Pointer to the User object being constructed to hold the check and reply items for the user being authenticated
- Search result entry
- Pointer to the reply packet currently being constructed

Here is an example of *PostSearchHook*:

```
# this example for LDAP2 gets an additional attribute,
# multiplies it by 60 and uses it for Session-Timeout
# as a reply attribute for the user
PostSearchHook sub {my $attr = $_[4]->get('someldapattr');\
            $_[3]->get_reply->add_attr('Session-Timeout',\
            $attr * 60);}
```

Tip

In order to get any attributes you may want to access in the *PostSearchHook*, you also need to add this to the *<AuthBy LDAP>* clause:

```
AuthAttrDef someldapattr
```

someldapattr is the name of the LDAP attribute you are going to access in the *PostSearchHook*.

3.47.10. ServerChecksPassword

Normally, Radiator fetches the user's password attribute from the LDAP server using the *PasswordAttr* parameter and checks the password internally. This optional parameter causes the LDAP server to check the password instead. This is useful with LDAP servers that implement proprietary encryption algorithms in their passwords, or do not provide access to password attribute. For example, Microsoft Active Directory does not provide read access to password information over LDAP.

When *ServerChecksPassword* is specified, the password checking is performed using an LDAP `bind` operation.

Here is an example of using *ServerChecksPassword*:

```
# We are using Active Directory
ServerChecksPassword
```

CAUTION

ServerChecksPassword is compatible with PAP, EAP-TTLS/PAP, and other authentication methods that provide a plain text password. *ServerChecksPassword* does not work with CHAP, MSCHAP, and most EAP methods since these do not provide a password Radiator can use with an LDAP `bind` operation.

Note

In some cases, using *ServerChecksPassword* with *HoldServerConnection* may cause failure situations. This is due to some LDAP servers' behaviour when the password check fails but the connection is not closed. A failure situation may also occur when the password check succeeds but the user is not allowed to perform searches in the server. If your users experience unexpected authentication failures, try testing your system without using these 2 parameters together.

3.47.11. UnbindAfterServerChecksPassword

This optional parameter works around problems with some LDAP servers. Normally, when ServerChecksPassword is set, after Radiator checks a users password the LDAP connection is not unbound. This can cause problems with some LDAP servers (notably Oracle ID and Novell eDirectory), where they unexpectedly cause the following LDAP query to fail with LDAP_INAPPROPRIATE_AUTH. Setting this flag causes an unbind after each ServerChecksPassword bind.

3.47.12. AuthCheckDN

This optional parameter allows you to specify an alternate DN to use to check a user's password, instead of the one returned by the search result. Special formatting characters are permitted, and %0 is replaced by the DN returned by the search.

Note

This is an advanced parameter. Most installations will not need to use it. Available in AuthBy LDAP2 only.

3.47.13. MaxRecords

This optional parameter specifies the maximum number of matching LDAP records to use for check and reply items. Default is 1 to be backwards compatible with earlier versions of AuthBy LDAP2. Only the first match (if any) is used for ServerChecksPassword. This parameter can be used to extract additional check and reply items from LDAP records that define user roles.

3.47.14. GetNovellUP

This optional parameter can be used with the Novell eDirectory LDAP server to fetch the user's Universal Password and use it to authenticate the user. The eDirectory Universal Password is a single password for each user that can be used to authenticate a range of Unix and Windows services. Normally it is not possible to fetch the users password from eDirectory, but GetNovellUP uses a special Novell API to fetch the users plaintext password.

GetNovellUP will fetch the password if ServerChecksPassword is not set, and if PasswordAttr and EncryptedPasswordAttr are either not set or are not present in the user's LDAP record.

Passwords retrieved with GetNovellUP are in plaintext and are compatible with PAP, CHAP, MSCHAP, MSCHAPV2, TLS, TTLS-*, PEAP-MSCHAPV2, EAP-MD5 etc.

The eDirectory server must be configured correctly before it will supply Universal Passwords to Radiator. The following conditions must be met.

- eDirectory Password Policy must be created and assigned to the group, organisational unit or organisation that holds the users to be authenticated.
- Password Policy must have Universal Passwords enabled.
- Password Policy must have ‘Allow password retrieval by admin’ enabled.

See `goodies/edirectory.txt` for more details about how to install and configure eDirectory so that Radiator can use GetNovellUP successfully.

3.47.15. GroupSearchFilter

For advanced applications, you can specify a search filter that Radiator will use to find which user groups a user belongs to by using the optional GroupSearchFilter parameter. It allows you to use arbitrarily complicated LDAP search filters to find the names of user groups the user belongs to. Special formatting characters are permitted, and %0 is replaced by UsernameAttr and %1 by the user name.

3.47.16. GroupNameCN

When GroupSearchFilter is specified and Radiator looks for the user groups the user is a member of, this parameter specifies the name of the Group name attribute in the LDAP records. Defaults to "cn".

3.47.17. GroupBaseDN

When GroupSearchFilter is specified and Radiator looks for the user groups the user is a member of, this parameter specifies an alternate LDAP base DN for the group search. Defaults to the value of BaseDN.

3.47.18. CheckAttr

This is the optional name of the LDAP attribute that contains the RADIUS check items for the user. During authentication, all the check items in this LDAP attribute (if specified) will be matched against the RADIUS attributes in the authentication request in the same was as for AuthBy FILE and AuthBy SQL, including Expiration in the format “Dec 08 1998”. Defaults to undefined. For more information, see Section 7.1. Check items on page 462.

Tip

If there are multiple instances of the LDAP attribute for the user, they are concatenated together with commas. This means that you can have each RADIUS check attribute in its own LDAP attribute for easy reading and maintenance.

Tip

AuthAttrDef is a more general and easy to use method of defining check and reply items. Support for CheckAttr will be discontinued sometime in the future.

```
# Check the radius items in checkitems
CheckAttr checkitems
```

3.47.19. ReplyAttr

This is the optional name of the LDAP attribute that contains the RADIUS reply items for the user. If the user authenticates successfully, all the RADIUS attributes named in this LDAP attribute will be returned to the user in the Access-Accept message. Defaults to undefined. For more information, see Section 7.2. Reply items on page 480.

Tip

If there are multiple instances of the LDAP attribute for the user, they are concatenated together with commas. This means that you can have each RADIUS reply attribute in its own LDAP attribute for easy reading and maintenance.

Tip

AuthAttrDef is a more general and easy to use method of defining check and reply items. Support for ReplyAttr will be discontinued sometime in the future.

```
# Reply with all the items in replyitems
ReplyAttr replyitems
```

3.48. <AuthBy SYSTEM>

<AuthBy SYSTEM> provides authentication with your getpwnam, getgrgid, and getgrnam system calls. On most Unix hosts, that means authentication from the same user database that normal user logins occur from, whether that be `/etc/passwd`, NIS, YP, NIS+, or the like. It is implemented in `AuthSYSTEM.pm`. This allows you to hide whether its password files, NIS+, PAM or whatever else might be installed on your system. It is not supported on Windows, or on systems with shadow password files unless Radiator runs with root or other permissions that allow it to access them.

<AuthBy SYSTEM> honours the Group and GroupList check item. These check items can match the numeric or symbolic group number of the primary or any secondary group.

<AuthBy SYSTEM> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.48.1. UseGetspnam and UseGetspnamf

These parameters are deprecated and will be removed from the future Radiator versions.

3.49. <AuthBy TACACSPLUS>

<AuthBy TACACSPLUS> provides authentication via a TacacsPlus server. It supports authentication only, not accounting or authorisation. It requires the Authen::TacacsPlus module. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3. Use at least version `TacacsPlus-0.15.tar.gz`. Earlier versions do not work properly. Version 0.15 supports PAP authentication only. Later versions support both PAP and CHAP.

<AuthBy TACACSPLUS> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.49.1. Host

This optional parameter specifies the name of the host where the TacacsPlus server is running. It can be a DNS name or an IP address. Defaults to **`localhost`**.

```
Host oscar.example.com
```

3.49.2. Key

This mandatory parameter specifies the encryption key to be used to encrypt the connection to the TacacsPlus server. You must specify this. There is no default. It must match the key specified in the TacacsPlus server configuration file.

```
# There is a line saying key = mytacacskey in my tac_plus
# config file
Key mytacacskey
```

3.49.3. Port

This optional parameter specifies the TCP port to be used to connect to the TacacsPlus server. It can be a service name as specified in /etc/services or an integer port number. Defaults to ‘tacacs’ (TCP port 49). You should not need to change this unless your TacacsPlus server is listening on a non-standard port.

3.49.4. Timeout

This optional parameter specifies the number of seconds timeout. Defaults to 15. You would only need to change this under unusual circumstances.

3.49.5. AuthType

This optional parameter allows you to force the type of authentication to be used in the Tacacs+ request sent to the Tacacs+ server. Options are 'PAP' and 'ASCII'. The default is to choose PAP if the version of Authen::TacacsPlus is 0.16 or greater, otherwise ASCII.

```
# Force ASCII auth regardless of the version of
# Authen::TacacsPlus installed
AuthType ASCII
```

3.50. <AuthBy PAM>

<AuthBy PAM> provides authentication via any method supported by PAM (Pluggable Authentication Modules) on your host. It is implemented in AuthPAM.pm. It requires that PAM be installed and configured on your host, and it also requires the Perl module `Authen::PAM` 0.04 or later. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3.

<AuthBy PAM> asks PAM to authenticate the user using the PAM service specified with the Service parameter (defaults to **`login`**).

AuthBy PAM has not been tested on Windows platforms.

Tip

Make sure PAM is configured on your host before building and testing the Authen-PAM Perl module, otherwise **`make test`** will report errors. This will usually require configuring `/etc/pam.conf`, or perhaps /etc/pam.d/login for the login service. For example, on our Red Hat Enterprise Linux 5.2, we found that we had to remove the **`pam_securetty`** from our `/etc/pam.d/login` file to enable testing from other than a secure TTY. Consult your system documentation for details on configuring PAM.

<AuthBy PAM> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.50.1. Service

This optional parameter specifies the PAM service to be used to authenticate the user name. If not specified, it defaults to **`login`**

```
# We want to use the PAM "ppp" service to authenticate our users
Service ppp
```

3.50.2. UsePamEnv

This optional parameter allow you to get UID, GID etc. if your PAM supports it, and your Authen::PAM was compiled with -DHAVE_PAM_ENV_FUNCTIONS. This can be useful with PAM authenticators which can supply UID, GID or other values of the user.

If this parameter is set, AuthBy PAM will gather PAM Environment strings and use them to set RADIUS reply attributes according to the following table:

Table 9. How PAM Environment strings are converted to RADIUS reply attributes

PAM Env string	RADIUS reply attribute
UID	OSC-Uid
GID	OSC-Gid
HOME	OSC-Home
SHELL	OSC-Shell

3.50.3. PasswordPrompt

This optional parameter allows you to specify the prompt string which PAM uses to ask for a password. Defaults to ‘password’. You may need to change this if your PAM module asks for data with a different prompt, such as ‘Code’.

3.51. <AuthBy ADSI>

<AuthBy ADSI> authenticates from Windows Active Directory, which is the user information database on Windows 2000 and later servers. It uses ADSI (Active Directory Service Interface) to get user information from any Active Directory service provider available to your Windows server. It is only available on Windows 2000 and later server platforms. It is implemented in AuthADSI.pm.

ADSI is a unified interface to Windows user information that was introduced in Windows 2000. Active Directory can access user information from a range of provider types:

- NT Primary or Secondary domain controller (WinNT:)
- Active Directory LDAP database (LDAP:)
- Novell Directory Services (NDS:)

You can configure AuthBy ADSI to use any of these service providers.

For more information, see Active Directory Service Interfaces website [https://msdn.microsoft.com/en-us/library/aa772170(v=vs.85).aspx].

During authentication, <AuthBy ADSI> check and honours AccountDisabled, IsAccount-Locked and LoginHours for the user being authenticated. It also checks the users password (by attempting to change it). Because Active Directory does not make the plaintext password available, <AuthBy ADSI> only supports PAP, not CHAP or MSCHAP authentication.

<AuthBy ADSI> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.51.1. BindString

BindString is the string that defines what ADSI object will be bound in order to get user details. You can bind to any Active Directory provider supported on your Radiator host, but WinNT or LDAP will be the usual choices. BindString must specify which provider to use and how to match the user. Use %0 to specify the user name to match.

WinNT means to use an NT 4.0 primary or backup domain controller, e.g. `WinNT:MyDomain/%0`,User means to match Users in the Windows NT domain called MyDomain. If the domain is omitted, the “best” domain controller in the default domain will be used.

Other acceptable variants are:

- `BindString WinNT://%0,User`

- *BindString WinNT://domain/%0,User*
- *BindString WinNT://domain/controller/%0,User*

LDAP means to use an LDAP server, including Microsoft Exchange and Windows 2000 Active Directory e.g. *LDAP://ldapsvr/cn=%n,cn=Users,dc=yourdomain, dc=com* means to match a user with the given common name (cn), in the AD domain yourdomain.com. If ldapsvr is omitted, the default AD server will be used.

Other acceptable variants are:

- *BindString LDAP://cn=%0.......*
- *BindString LDAP://controller/cn=%0.......*
- *BindString LDAP://msexchangeserver.bigco.com/cn=%0.......*
- *BindString LDAP://msexchangeserver:390/cn=%0.......*

NDS means use Novell Directory Services. e.g. NDS://MarsTree/O=MARS/OU=MARTIANS/CN=%0

The default is *WinNT://%0,User* which means a user with the given user name in the default domain

```
# Get users from the OSC domain in NT
BindString WinNT://OSC/%0,User
# Get user details from the Users folder in Active Directory
# for the AD domain open.com.au
BindString LDAP://cn=%0,cn=Users,dc=open,dc=com,dc=au
```

3.51.2. AuthUser

This parameter defines how to construct the Active Directory user name to be authenticated by Active Directory. You can choose whether to use standard NTLM user names or AD Distinguished Names. This is a different concept to BindString, which specifies what AD object to get account details from.

The default is %0, which will try to authenticate the user name sent by the NAS (after RewriteUsernames have been applied).

This example will authenticate the user from an AD user record in the ‘csx users’ Organizational Unit, and get account details from the same AD record. Unlike NTLM user names, it will even work for user names with spaces in them. Note that you need to specify AuthFlags of 0 in order to use an Active Directory DN in AuthUser.

```
BindString LDAP://cn=%0,ou=csx users,dc=open,dc=com,dc=au
AuthUser cn=%0,ou=csx users,dc=open,dc=com,dc=au
AuthFlags 0
```

3.51.3. AuthFlags

This optional parameter specifies flags to be passed to OpenDSObject. The default is 1, which means NTLM secure authentication. For more information, see AuthFlags website [https://msdn.microsoft.com/en-us/library/ms524513(v=vs.90).aspx]. You need to specify 0 to use an Active Directory DN in AuthUser.

3.51.4. AuthAttrDef

This optional parameter allows you to use additional ADSI user information as RADIUS check or reply items. This is most useful when you define new user attributes in your Active Directory schema. It is beyond the scope of this document to describe how to add new attributes to an Active Directory schema.

The general format is

```
AuthAttrDef adsiname,radiusattr,type
```

- *adsiname* is the name of an attribute in your Active Directory User schema. The value of that attribute will be fetched using ADSI during authentication.
- *radiusattr* is the name of the RADIUS attribute that the adsiname will be converted to. check or reply item. The special radiusattr 'GENERIC' indicates that it is a list of comma separated attribute=value pairs, similar to ReplyAttr or CheckAttr.
- *type* specifies whether to use the value as a check or reply item. type may be check, reply or request. If type is "request" the value is saved in the current request, from where it can be later collected with a special formatting macro like: %{attributename}.

For example,

```
AuthAttrDef address,Framed-IP-Address,reply
```

would get an attribute called 'address' from the ADSI user record, and put it into Framed-IP-Address attribute in the RADIUS reply. If address was not defined in your schema, or there was no value defined for the user being authenticated, then Framed-IPAddress would not be set in the reply.

Multi-valued AD attributes can be used as check items, which results in Radiator passing the authentication if one of the multiple items matches. For example if you have this in your AuthBy ADSI:

```
AuthAttrDef otherHomePhone,Calling-Station-Id,check
```

and multiple entries in Home, 'Other....' tab of the Telephones tab. Then Radiator will let the user log in if they call from any one of the Other Home Phone numbers.

3.51.5. GroupBindString

This optional parameter is used to generate an ADSI group identifier when checking group membership through a Group= check item. Defaults to 'WinNT://%0,Group' (i.e. the named group in the default domain). Special characters can be used, and %0 is replaced with the name of the group being checked, and %1 with the name of the user whose group membership is being checked.

3.51.6. GroupUserBindString

This optional parameter is used to generate an ADSI user name identifier when checking group membership through a Group= check item. Defaults to 'WinNT://%1' (i.e. the named user). Special characters can be used, and %0 is replaced with the name of the group being checked, and %1 with the name of the user whose group membership is being checked.

This example checks whether an NT user in the OSC domain is in an NT Group in the OSC domain:

```
GroupBindString WinNT://OSC/%0,Group
GroupUserBindString WinNT://OSC/%1
```

This example checks whether the active directory user identified by GroupUserBindString is in the group defined by GroupBindString.

```
GroupBindString LDAP://cn=%0,dc=open,dc=com,dc=au
GroupUserBindString LDAP://cn=%1,cn=Users,dc=open,dc=com,dc=au
```

Tip

With AD, do not confuse Organizational Unit with group membership. They are different ideas. A user can be in one OU, but be a member of multiple groups. Use GroupBindString, GroupUserBindString and the Group= check item to check for AD group members.

3.51.7. CheckGroupServer

This optional parameter, in conjunction with CheckGroup, allows you to set a Class reply attribute that depends on which NT group the user is a member of. CheckGroupServer is the name of an NT domain controller that contains group information.

```
<AuthBy ADSI>
      CheckGroupServer brbaaa01
      CheckGroup USVPN,ou=tcgic
      CheckGroup UKVPN,ou=tcgic
      ....
</AuthBy>
```

3.51.8. CheckGroup

This optional parameter, in conjunction with CheckGroupServer, allows you to set a Class reply attribute that depends on which NT group the user is a member of. Recall that if you set the Class in an access reply, then subsequent accounting requests sent by the NAS for that session will contain exactly the same Class attribute. This is useful for remembering which accounting group or rate to charge the user.

CheckGroup is a comma-separated pair of names. The first is an NT group name, the second is an arbitrary string. During authentication, if the user is a member of the NT group, then the Class attribute in the reply will be set to the arbitrary string. The first match found will be used.

For example:

```
<AuthBy ADSI>
      CheckGroupServer romeo
      CheckGroup USVPN,premium
      CheckGroup UKVPN,standard
      ....
</AuthBy>
```

In this example, if a user is a member of the NT group USVPN, then the reply will contain Class=premium. If they are in UKVPN group, then the reply will contain Class=standard. If they are in neither group, then no Class will be set in the reply.

3.52. <AuthBy PORTLIMITCHECK>

<AuthBy PORTLIMITCHECK> can apply usage limits for arbitrary groups of users. It is implemented in `AuthPORTLIMITCHECK.pm`. It requires that you have a <SessionDatabase SQL> defined in your Radiator configuration.

This module allows you to specify for example that up to 20 ports can be used by a customer with a certain DNIS, or 10 ports in one POP and 20 ports in another. Users can be grouped by any attribute stored in your SQL Session Database.

Furthermore, you can arrange to set the Class attribute for the session depending on bands of port usage. This could allow you to charge different amounts for the first 10 and the second 10 ports, for example, or to have a

premium price for excessive port occupancy. (The Class attribute set this way will be sent back by the NAS in all accounting requests for that session. You could then use the value of the Class attribute to tell your billing system how much to charge for the session. The ability to actually charge differently depends on the functions of your billing system.)

This module must be used in conjunction with some other module that actually performs the authentication of the user. PORTLIMITCHECK should be considered as a pre-check to make sure that the login would be within the port occupancy limits you have specified.

<AuthBy PORTLIMITCHECK> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.52.1. CountQuery

This parameter specifies an SQL query that will be used to count the users currently online according to the SQL Session Database. Defaults to “select COUNT(*) from RADONLINE where DNIS=’%{Called-Station-Id}’”. AuthBy PORTLIMITCHECK will compare the results of this query with SessionLimit in order to determine whether the user will be permitted to log in at all.

Tip

You must have a <SessionDatabase SQL> configured into Radiator. The Session Database must be configured to save the columns that you plan to use to group users in your CountQuery. So, if you are using the default CountQuery, your SQL Session Database must be configured to save the Called-Station-Id attribute to the DNIS column, with something like:

```
<SessionDatabase SQL>
      # We want to save the DNIS as well as the usual things.
      # Requires a different schema to the example RADONLINE provided
      AddQueryinsert into RADONLINE (USERNAME,\
      NASIDENTIFIER, NASPORT, ACCTSESSIONID, TIME_STAMP,\
      FRAMEDIPADDRESS, NASPORTTYPE, SERVICETYPE, DNIS) \
      values (%0, '%N',\
      %{NAS-Port}, '%{Acct-Session-Id}', %{Timestamp},\
      '%{Framed-IP-Address}', '%{NAS-Port-Type}', \
      '%{Service-Type}', '%{Called-Station-Id}')
</SessionDatabase>
```

3.52.2. SessionLimit

This parameter specifies the absolute upper limit to the number of current logins permitted to this group of users. Defaults to 0. For example if SessionLimit is set to 10, then up to 10 concurrent sessions are permitted. If an 11th user attempts to log in through this AuthBy, they will be rejected. If LimitQuery is defined, and if it successfully gets an integer from the database, then the result of the query will be used instead of Session-Limit. SessionLimit may contain special formatting characters.

```
# They have paid for 20 ports
SessionLimit20
```

3.52.3. ClassForSessionLimit

This optional parameter allows you to set up different charging bands for different levels of port occupancy in this group of users. You can have one or more ClassForSession- Limit lines. If the current level of port usage is below a ClassForSessionLimit, then the class name will be applied as a Class attribute to that session. Your

NAS will then tag all accounting records for that session with the Class attribute. If your billing system records and uses the Class attribute in accounting records, then you could use this to charge differently for different levels of port occupancy.

```
# The first 2 users will be tagged with a Class of "normal"
# the next 2 with "overflow". No more than 4 concurrent users
# permitted
SessionLimit 4
ClassForSessionLimit normal,2
ClassForSessionLimit overflow,4
```

3.52.4. LimitQuery

This optional parameter can be used to override the fixed session limit defined by SessionLimit. LimitQuery is an SQL query that is expected to return an integer that will be used as the limit instead of SessionLimit. If LimitQuery fails to execute, or if it does not return any rows, then SessionLimit will be used as the limit instead.

```
LimitQuery select maxsessions from customers where\
            dnis=%{Called-Station-Id}
```

3.52.5. IgnoreErrors

This optional parameter causes AuthBy PORTLIMITCHECK to IGNORE rather than REJECT if the SQL query or SQL connection fails. It can be useful for recovering from or working around SQL server failures.

3.53. <AuthBy DYNADDRESS>

<AuthBy DYNADDRESS> is used to dynamically allocate IP address information in conjunction with *<AddressAllocator xxxxxx>* clauses. It is implemented in `AuthDYNADDRESS.pm`. At present, there are three Address Allocation engines provided:

- AddressAllocator SQL can allocate IPv4 addresses and IPv6 prefixes out of an SQL database. For more information, see Section 3.115. <AddressAllocator SQL> on page 383.
- AddressAllocator DHCP can allocate addresses from a DHCP server. For more information, see Section 3.116. <AddressAllocator DHCP> on page 389.
- AddressAllocator DHCPv6 can allocate addresses and prefixes from a DHCPv6 server. For more information, see Section 3.117. <AddressAllocator DHCPv6> on page 393.

IP address and prefix allocation is usually the responsibility of your NAS, and most organisations have allocation done by their NASs. There are sometimes special requirements that mean allocation must be done by a central authority, and *<AuthBy DYNADDRESS>* allows you to make your RADIUS server the address allocator.

When using *<AuthBy DYNADDRESS>*, the usual arrangement is to make *<AuthBy DYNADDRESS>* the last AuthBy clause in a *<Handler>*. Some previous AuthBy clauses are responsible for authenticating the password of the user. Only if the authentication succeeds is *<AuthBy DYNADDRESS>* run and an address allocated.

It is common practice to maintain multiple IP address pools, with each pool used for a different class of users, perhaps with different access controls. The features of *<AuthBy DYNADDRESS>* allow you to control which pools to allocate from, and also how to translate the allocated information into RADIUS reply attributes.

See `goodies/addressallocator*` in the Radiator distribution for configuration samples for different allocation engines.

<AuthBy DYNADDRESS> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.53.1. AddressAllocator

This is the Identifier of an <AddressAllocator xxxxxx>. It specifies which Address Allocation engine will be used to allocate the addresses. It must match the Identifier parameter of an <AddressAllocator xxxxxx> clause. Special formatting characters are permitted. The named AddressAllocator will be used to allocate and deallocate addresses of all users authenticated through this AuthBy.

```
<AddressAllocator SQL>
      Identifier SQLAllocator
      DBSource ....
      .....
</AddressAllocator>
<Handler>
      AuthByPolicy ContinueWhileAccept
      # This does the authentication
      <AuthBy FILE>
            Filename xxxxxx
      </AuthBy>
      # If authentication succeeds, this allocates an
      # address, using the AddressAllocator above
      <AuthBy DYNADDRESS>
            AddressAllocator SQLAllocator
      </AuthBy>
</Handler>
```

Note

This parameter used to be called Allocator. This word is still supported but deprecated. Support will be removed some time in the future.

3.53.2. PoolHint

This optional parameter specifies how the pool hint is derived. A pool hint is generally used by an AddressAllocator to determine which pool to allocate an address from. The value of the pool hint will therefore depend on what type of Address Allocator you are using, and usually which pools are available.

The default PoolHint (and the most common requirement) is '%{Reply:PoolHint}', which means the pool hint is an attribute called 'PoolHint' in the current reply. Presumably the PoolHint will have been set in the reply by some previous AuthBy clause. This is easy to do for example in an AuthBy FILE, by adding the pool hint attribute to the users file:

```
user1       Password=x
            PoolHint=pool1

user2       Password=y
            PoolHint=pool2
```

The PoolHint attribute will then be used by the Address Allocator to indicate which pool to allocate from. The exact way the pool hint is used depends on the type of Address Allocator you are using, so refer to the documentation for the Address Allocator.

Tip

If the pool hint resolves to an empty string, a DEBUG message will be issued, no address will be allocated, but the request will be ACCEPTED. This allows you to arrange for only some users to get an address allocated (and the NAS will allocate addresses for the others).

3.53.3. MapAttribute

This optional parameter allows you to specify how the results of the address allocation are to be placed in the reply.

If the yiaddr attribute (usually Framed-IP-Address) is already set in the reply, then AuthBy DYNADDRESS will not allocate an address, and will just ACCEPT the request. This means that if a user record has a fixed IP address in it, then AuthBy DYNADDRESS will not allocate an address for that user.

The above also applies to IPv6 iaaddr and iaprefix. However, if there is additional configuration information, such as DNS server addresses, the allocator may still make a query to fetch this information. For example, DHCPv6 allocator can send a stateless Information-request.

Each MapAttribute specifies the name of the allocation variable, and the name of the RADIUS attribute in which to place it (if it is not set already). The following allocation variables are normally available for IPv4 allocation. Some AddressAllocator clauses may not provide all of these or may also make others available.

- yiaddr, usually the allocated IP address in dotted quad format (e.g. 1.2.3.4)
- subnetmask, usually the IP Subnet Mask in dotted quad format (e.g. 255.255.255.255)
- dnsserver, usually the address of the DNS server to use in dotted quad format (e.g. 1.2.3.4) Only one DNS server address is currently permitted and supported for IPv4

The default behaviour is to place the allocated IPv4 address in Framed-IP-Address, and the IPv4 subnetmask in Framed-IP-Netmask. This is equivalent to:

```
MapAttribute yiaddr, Framed-IP-Address
MapAttribute subnetmask, Framed-IP-Netmask
```

There are no default MapAttribute parameters for IPv6 information. For DHCPv6 allocator, the name of the allocation variables must be the case insensitive name of the desired DHCPv6 option. For example, the following requests and maps OPTION_IAADDR and OPTION_DNS_SERVERS. Multiple DNS servers are mapped to multiple instances of DNS-Server-IPv6-Address

```
MapAttribute iaaddr, Framed-IPv6-Address
MapAttribute dns_servers, DNS-Server-IPv6-Address
```

For more information about the details of the available allocation variables, see Section 3.117. <AddressAllocator DHCPv6> on page 393 and Section 3.115. <AddressAllocator SQL> on page 383.

3.53.4. MapResultHook

MapResultHook allows modifying the allocation results after they have been received, and before Radiator has processed the MapAttribute definitions.

This example complies with how some vendors require uplink addresses as host routes with a /128 prefix for the requesting routers.

```
MapAttribute iaprefix, Delegated-IPv6-Prefix
MapAttribute iaaddr, Framed-IPv6-Prefix
```

```
MapResultHook sub { my $result = $_[0]; \
      $result->{iaaddr} .= "/128" if $result->{iaaddr}; }
```

3.53.5. RunWhenMissing

This optional flag parameter controls if confirm and deallocate operations are run even if the address, as defined by MapAttribute yiaddr, is missing from the accounting requests that have Acct-Status-Type set to `Start`, `Alive` or `Stop`. The default value is enabled and confirm, and deallocate are always run.

```
# If Framed-IP-Address is missing from the request do nothing
MapAttribute yiaddr, Framed-IP-Address
RunIfMissing false
```

Tip

When IPv4 and IPv6 allocators are chained, yiaddr is typically not set for one of the allocators. In this case, set RunWhenMissing to off.

3.54. <AuthBy ROUNDROBIN>, <AuthBy VOLUMEBALANCE>, <AuthBy LOADBALANCE>, <AuthBy HASHBALANCE>, <AuthBy EAPBALANCE>

These authentication methods are sub-types of *<AuthBy RADIUS>* but they also do load distribution and balancing. They allow you to proxy RADIUS requests to any number of remote RADIUS servers and to distribute the RADIUS load amongst those servers. Further, if any remote server fails to reply to a proxied request, the remote server is marked as unavailable until the *FailureBackoffTime* expires. During that period, no requests are proxied to that remote RADIUS server.

These authentication methods understand all the same parameters as *<AuthBy RADIUS>*. For more information, see Section 3.42. <AuthBy RADIUS> on page 202. The only difference between them and *<AuthBy RADIUS>* is the algorithm for choosing which remote host to proxy to and how the received and dropped responses provide feedback for the host selection algorithm.

Note

See an example config file showing how to use various load balancing modules in `goodies/proxyalgorithm.cfg` in the Radiator distribution.

3.54.1. <AuthBy ROUNDROBIN>

<AuthBy ROUNDROBIN> distributes RADIUS requests to the servers. The first incoming RADIUS request is proxied to the first server listed, the next to the second listed and so on, until the list is exhausted. Then it starts again at the top of the list. If at any time a proxied request does not receive a reply from a remote server, that server is marked as unavailable until *FailureBackoffTime* seconds has elapsed. Meanwhile that request is retransmitted to the next host due to be used.

The result of this algorithm is to distribute the load equally amongst all the currently operational remote RADIUS servers.

<AuthBy ROUNDROBIN> supports the same parameters as Section 3.42. <AuthBy RADIUS> on page 202.

3.54.2. <AuthBy VOLUMEBALANCE>

When using *<AuthBy VOLUMEBALANCE>*, the incoming RADIUS requests are distributed between all the listed hosts according to the relative values of their BogoMips attributes. BogoMips is a relative measure of the processing power of that host. A *Host* with a BogoMips rating of 2 receives twice as many request as one with the default BogoMips rating of 1. If at any time a proxied request does not receive a reply from a remote server, that server is marked as unavailable until *FailureBackoffTime* seconds has elapsed. Meanwhile, that request is retransmitted to the next host due to be used.

The result of this algorithm is to distribute the load amongst all the currently operational remote RADIUS servers, according to the relative values of their BogoMips values. For more information about configuring the BogoMips parameter, see Section 3.43.1. BogoMips on page 221.

<AuthBy VOLUMEBALANCE> supports the same parameters as Section 3.42. <AuthBy RADIUS> on page 202.

3.54.3. <AuthBy LOADBALANCE>

When using *<AuthBy LOADBALANCE>*, the incoming RADIUS requests are distributed between all the listed hosts according to the relative values of their BogoMips attributes and the time the remote server takes to process requests. BogoMips is intended to be a relative measure of the processing power of that host. A *Host* with a BogoMips rating of 2 receives twice as many requests as the one with the default BogoMips rating of 1. Every request that is proxied and receives a reply has its turnaround time, which is measured in microseconds. A sliding average response time over the last 10 request is calculated. Requests are proxied to the server that is currently responding fastest. If at any time a proxied request does not receive a reply from a remote server, that server is marked as unavailable until *FailureBackoffTime* seconds has elapsed. Meanwhile, that request is retransmitted to the next host due to be used.

The result of this algorithm is to distribute the load until the BogoMips-weighted response time is the same for all servers. This allows the load balancing to adapt to the changes in the available processing capacity of a remote host.

For more information about configuring the BogoMips parameter, see Section 3.43.1. BogoMips on page 221.

Example

In this example, Radiator running by itself on a host distributes RADIUS requests to 3 remote servers. The second host has twice the processing power of the others, and therefore gets twice as many requests as the other 2.

```
....
# Proxy all requests to remote hosts inside our network:
<Handler>
      <AuthBy VOLUMEBALANCE>
            # If a host fails, do not send to it again
            # for 100 seconds
            FailureBackoffTime 100
            # Default values for all hosts. You can change
            # them for a single host in a Host clause
            Secret mysecret
            RetryTimeout 1
            Retries 1
            # Hosts to send to are listed below
            <Host 203.63.154.2>
            </Host>
```

```
            <Host 203.63.154.3>
                  BogoMips 2
            </Host>
            # This host has non-standard ports
            <Host 203.63.154.4>
                  AuthPort 1647
                  AcctPort 1648
            </Host>
      </AuthBy>
</Handler>
```

3.54.4. <AuthBy HASHBALANCE>

<AuthBy HASHBALANCE> distributes RADIUS requests among a set of RADIUS servers so that multiple related requests from the same user go to the same server unless that server fails. This prevents the requests being distributed among multiple servers, which is prone to problems with authentication methods that use multiple messages, such as EAP.

When using *<AuthBy HASHBALANCE>*, a hash number is calculated for each incoming request. That hash number is based on number of attributes in the incoming request. It is used to select which host to use. If the selected Host is unavailable, then the next one on the Host list is used in sequence until the Host list is exhausted. This results in a random distribution of requests among the available server, and the same server handles all the requests related to a single user.

<AuthBy HASHBALANCE> supports the same parameters as Section 3.42. <AuthBy RADIUS> on page 202.

The *HashAttributes* parameter specifies which attributes in the incoming request is used to select the server. The default value is shown below. It is suitable for most use cases.

```
HashAttributes %{Request:Calling-Station-Id}:%n
```

3.54.5. <AuthBy EAPBALANCE>

Note: Use *<AuthBy HASHBALANCE>* with new configurations and consider migrating current configurations from *<AuthBy EAPBALANCE>* to *<AuthBy HASHBALANCE>*. *<AuthBy HASHBALANCE>* has better interoperability with different RADIUS implementations.

<AuthBy EAPBALANCE> is similar to *<AuthBy HASHBALANCE>* in the aspect that it is intended to ensure all EAP requests relating to a single session always go to the same target RADIUS server. It uses the RADIUS State attribute to track EAP sessions, and therefore relies on all the RADIUS clients supporting the State attribute similarly. The target server for the first EAP request in a session is chosen in the same was as *<AuthBy HASHBALANCE>*. If you are unsure, try *<AuthBy HASHBALANCE>* with new configurations first.

<AuthBy EAPBALANCE> is useful in installations where all RADIUS clients are known to support the RADIUS State attribute similarly. This usually covers the network managed by a single organisation, but is likely to not work with Eduroam or other roaming services where remote or intermediate proxies may add, change, or remove the State attribute. With these services, *<AuthBy HASHBALANCE>* with the default HashAttributes setting should ensure correct operation.

CAUTION

When EAPBALANCE is used in a ServerFarm architecture to proxy requests to a set of back end RADIUS servers, the duplicate detection in the back end servers can be defeated by changes to requests

made by the server farm. It is therefore essential that all the backend servers in such an architecture have the *UseContentsForDuplicateDetection* flag set in the receiving Client clauses.

Note

See an example config file showing how to use EAPBALANCE in `goodies/eapbalance.cfg` in the Radiator distribution.

<AuthBy EAPBALANCE> supports the same parameters as Section 3.42. <AuthBy RADIUS> on page 202.

3.55. <AuthBy LDAPRADIUS>

This clause proxies requests to one or more target RADIUS servers. The target host is determined by a lookup in an LDAP database. This allows the easy management of large numbers of downstream radius servers, such as in a wholesale ISP. It inherits from both LDAP and *<AuthBy RADIUS>*.

<AuthBy LDAPRADIUS> runs the SearchFilter query to determine the details of the target RADIUS server until either an acknowledgment is received from the target or Num-Hosts is exceeded. This permits fallback RADIUS servers to be configured.

SearchFilter can be configured to select the target RADIUS server based on any attribute in the incoming request. The default is the user's Realm, but other possibilities, such as Called-Station-Id may be more useful for your organisation.

Tip

There is a sample LDAP schema for OpenLDAP in `goodies/radiator-ldap.schema` in your Radiator distribution. This schema is compatible with the default behaviour of SearchFilter and HostAttrDef allowing the selection of a target host primary based on Realm.

Tip

If SearchFilter fails to find any matching LDAP records, *<AuthBy LDAPRADIUS>* attempts to proxy according any *<Host xxxxxx>* clauses contained within the *<AuthBy LDAPRADIUS>* clause. For more information, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220. This permits unknown realms to be proxied to a catchall target server, such as GoRemote (GRIC) and IPASS.

This clause supports all the common LDAP configuration parameters. For more information about the LDAP configuration parameters, see Section 3.9. LDAP configuration on page 53.

<AuthBy LDAPRADIUS> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.55.1. BaseDN

This is the base DN, where searches are made. It is used in similar way as with all LDAP modules. For more information, see Section 3.9.1. BaseDN on page 53.

Special formatting characters are permitted. **%0** is replaced by the host counter. It is an integer that starts at **1** and counts the searches made for a given request. **%1** is replaced by the realm of the user name in the current request.

3.55.2. SearchFilter

This parameter specifies the LDAP search filter that is used to find the LDAP records containing remote RADIUS server data. The default value is **(oscRadiusTarget=%1)**, which is compatible with the example schema provided in `goodies/radiator-ldap.schema`, and selects a record where *oscRadiusTarget* matches the user's realm.

SearchFilter can contain any of the special characters. For more information, see Section 3.3. Special formatters on page 21. **%0** is replaced by the host counter. It is an integer that starts at **1** and counts the searches made for a given request. **%1** is replaced by the realm of the user name in the current request. Use **%0** to select a different record each time *HostSelect* is run for a given record, allowing you to choose, for example, primary or secondary server.

3.55.3. NumHosts

This parameter defines the maximum number of times that SearchFilter will be called for as given request. If NumHosts is exceeded for a given request, the proxying of the request fails. Defaults to 1. The current count is available as %0 in SearchFilter and HostAttrDef.

3.55.4. HostAttrDef

This optional parameter specifies which parameters to get from an LDAP record and how they are to be used to set the parameters of the Radiator Host clause for proxying. Format is

```
HostAttrDef ldapattrname,hostparamname
```

where *ldapattrname* is the name of the LDAP attribute to fetch and *hostparamname* is the name of the Radiator Host clause parameter it will be used to set. For more information about the available *hostparamname*, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220. If *hostparamname* is ‘failurePolicy’ it will be used to specify how AuthBy LDAPRADIUS will reply to the originating NAS if no reply is heard from any remote server for this request. The following values are supported:

- 0 ACCEPT
- 1 REJECT
- 2 IGNORE
- 3 CHALLENGE
- 4 REJECT_IMMEDIATE

The default behaviour if no reply is heard from any remote server is to not reply to the NAS. This will usually cause the NAS to re-send the request to its secondary RADIUS server.

In HostAttrDef, the *ldapattrname* may contain special characters, and %0 is replaced by hostCounter, an integer which starts at 1 and increases by one each time a search is made for a given request. You can use that mechanism to fetch different LDAP attributes for the primary, secondary etc. RADIUS servers.

If no HostAttrDef lines are specified, defaults to the equivalent of the following, which is compatible the sample OpenLDAP schema in `goodies/radiator-ldap.schema`. Note that not all LDAP parameters are required to be present. The minimum set required are Host and Secret. Host can be an IPv4 or IPv6 address.

```
HostAttrDef oscRadiusHost,Host
HostAttrDef oscRadiusSecret,Secret
HostAttrDef oscRadiusAuthPort,AuthPort
HostAttrDef oscRadiusAcctPort,AcctPort
```

```
HostAttrDef oscRadiusRetries,Retries
HostAttrDef oscRadiusRetryTimeout,RetryTimeout
HostAttrDef oscRadiusUseOldAscendPasswords,UseOldAscendPasswords
HostAttrDef oscRadiusServerHasBrokenPortNumbers,ServerHasBrokenPortNumbers
HostAttrDef oscRadiusServerHasBrokenAddresses,ServerHasBrokenAddresses
HostAttrDef oscRadiusIgnoreReplySignature,IgnoreReplySignature
HostAttrDef oscRadiusFailurePolicy,failurePolicy
```

3.56. <AuthBy SQLRADIUS>

This clause proxies requests to a target RADIUS server. The target host is determined by a table lookup in an SQL database. This allows the easy management of large numbers of downstream RADIUS servers, such as in a wholesale ISP. It inherits from both common SQL module and *<AuthBy RADIUS>*.

<AuthBy SQLRADIUS> runs the *HostSelect* query to determine the details of the target RADIUS server until either an acknowledgment is received from the target or *NumHosts* is exceeded. This permits fallback radius servers to be configured.

HostSelect can be configured to select the target RADIUS server based on any attribute in the incoming request. The default is the user's Realm, but other possibilities, such as Called-Station-Id may be more useful for your organisation.

Tip

There are example SQL table definitions in the `goodies/*.sql` scripts. These tables work with the default *HostSelect* allowing the selection of a target host primary and secondary based on Realm.

Tip

If *HostSelect* fails to select any rows, *<AuthBy SQLRADIUS>* attempts to proxy according any *<Host xxxxxx>* clauses contained within the *<AuthBy SQLRADIUS>* clause. For more information, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220. This permits unknown realms to be proxied to a catchall target server.

<AuthBy SQLRADIUS> understands the same parameters as *<AuthBy RADIUS>* for configuring any *<Host xxxxxx>* clauses. These parameters configure only *<Host xxxxxx>* clauses and do not set defaults or otherwise affect target hosts configured from the SQL.

For more information, see Section 3.42. <AuthBy RADIUS> on page 202. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.56.1. HostSelect

This parameter defines the SQL statement that is run to determine the details of the target RADIUS server. It is run for each request that is handled by the AuthBy. If no reply is received by the target RADIUS server for a given request, it is rerun to find a secondary server, and so on until either *HostSelect* returns no more rows, or the number of times exceeds *NumHost*s.

If *HostSelect* returns no rows, and if *<AuthBy SQLRADIUS>* contains *<Host xxxxxx>* clauses, then the request is proxied according to the *<Host>* clauses in order, the same as with *<AuthBy RADIUS>*. This is a useful catchall for unknown realms, and could be used to proxy to a GoRemote (GRIC) server or similar.

HostSelect is expected to return at least the target host name/address and the shared secret in that order. Optionally, you can also fetch a number of other columns to control the proxying process, including *RetryCount*, target ports and such. The columns fetched by *HostSelect* are used to determine the following *<AuthBy RADIUS>* Host parameters in this order. Any column that is NULL is ignored.

- Target host name or IP address
- *Secret*
- *AuthPort*
- *AcctPort*
- *Retries*
- *RetryTimeout*
- *UseOldAscendPasswords*
- *ServerHasBrokenPortNumbers*
- *ServerHasBrokenAddresses*
- *IgnoreReplySignature*
- *failurePolicy*

  This is an integer in the range **0** to **4** inclusive that indicates what sort of reply to send to the NAS in the event that proxying fails. You can use this to determine how to handle the failure of a downstream RADIUS server. The integers mean the following replies:

  - **0**: ACCEPT
  - **1**: REJECT
  - **2**: IGNORE
  - **3**: CHALLENGE
- *FailureBackoffTime*
- *MaxFailedRequests*
- *MaxFailedGraceTime*

For more information about how these attributes are used to control proxying, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220.

HostSelect can contain any of the special characters. For more information, see Section 3.3. Special formatters on page 21. Also, **%0** is replaced by the current host counter for this request. The counter starts with the value of *StartHost* which defaults to **1**. You can therefore use **%0** to select a different column each time *HostSelect* is run. **%1** is replaced with SQL quoted realm.

The default value is:

```
HostSelect select HOST%0, SECRET, AUTHPORT, ACCTPORT,\
RETRIES, RETRYTIMEOUT, USEOLDASCENDPASSWORDS, \
SERVERHASBROKENPORTNUMBERS,SERVERHASBROKENADDRESSES, \
IGNOREREPLYSIGNATURE, FAILUREPOLICY from RADSQLRADIUS \
where TARGETNAME=%1
```

The default value works with the example tables supplied in `goodies/*.sql`. Note that this allows for up to 2 target hosts per Realm, primary and secondary, and that the Realm to match goes in the TARGETNAME column.

Note

Details about failure history, backoff times and such are cached within Radiator memory, not in the SQL database.

Example

If you have a simple SQL table with one target host per Realm, *<AuthBy SQLRADIUS>* contains:

```
HostSelect select HOST%0, SECRET, AUTHPORT, ACCTPORT, RETRIES,\
RETRYTIMEOUT, USEOLDASCENDPASSWORDS, \
SERVERHASBROKENPORTNUMBERS, SERVERHASBROKENADDRESSES, \
IGNOREREPLYSIGNATURE, FAILUREPOLICY from RADSQLRADIUS where TARGETNAME=?
HostSelectParam %1
NumHosts 1
```

Example

If you want to choose the target RADIUS server based on Called- Station-Id and Realm, and multiple Called-Station-Ids can map to the same target RADIUS servers, and if the target has a primary and a secondary RADIUS server, you can use the example RADSQLRADIUS and RADSQLRADIUSINDIRECT tables, plus an *<AuthBy SQLRADIUS>* containing:

```
HostSelect select R.HOST%0, R.SECRET, R.AUTHPORT, \
R.ACCTPORT, R.RETRIES, R.RETRYTIMEOUT, \
R.USEOLDASCENDPASSWORDS, R.SERVERHASBROKENPORTNUMBERS, \
R.SERVERHASBROKENADDRESSES, R.IGNOREREPLYSIGNATURE, \
R.FAILUREPOLICY from RADSQLRADIUS R, RADSQLRADIUSINDIRECT I \
where I.SOURCENAME=? and I.TARGETNAME=R.TARGETNAME
HostSelectParam %{Called-Station-Id}
NumHosts 2
```

3.56.2. HostSelectParam

This optional parameter specifies a bind variable to be used with *HostSelect.* **%1** is replaced with unquoted realm. For more information, see Section 3.8.1. SQL bind variables on page 47.

Here is an example of using *HostSelectParam*:

```
# Use bound parameters to improve performance in SQL
HostSelect select HOST%0, SECRET, AUTHPORT, ACCTPORT, RETRIES,\
RETRYTIMEOUT, USEOLDASCENDPASSWORDS, \
SERVERHASBROKENPORTNUMBERS, SERVERHASBROKENADDRESSES, \
IGNOREREPLYSIGNATURE, FAILUREPOLICY from RADSQLRADIUS where TARGETNAME=?
HostSelectParam %1
NumHosts 1
```

3.56.3. NumHosts

This parameter defines the maximum number of times that HostSelect will be called for as given request. If NumHosts is exceeded for a given request, the proxying of the request fails. Defaults to 2. The current counter is available as %0 in HostSelect.

3.56.4. StartHost

StartHost sets the initial host number. Defaults to 1.

3.56.5. HostColumnDef

This optional parameter allows you to specify an alternate mapping between the fields returned by *HostSelect* and the parameters used to define the Host. If *HostColumnDef* is not specified, the mapping is the default as described in Section 3.56.1. HostSelect on page 249.

The format of *HostColumnDef* is:

```
HostColumnDef n,paramspec
```

Where **n** is the column number of the fields as returned by *HostSelect* (starting at **0**), and *paramspec* may be one of the following:

- *Host*
- *failurePolicy*
- any valid Host parameter as described in the Host clause. For more information, see Section 3.43. <Host xxxxxx> within <AuthBy RADIUS> on page 220. Also *StripFromRequest*, *RewriteUsername*, and *AddToRequest* are usable

In the following example, *HostSelect* returns five fields. The first defines the Host name or address, the second is the shared secret for that host, the third is the maximum retry count, and the fourth is the failure policy. The last is a comma-separated list of reply items that is added to the reply.

```
HostSelect select HOST%0, SECRET, RETRIES, FAILUREPOLICY, ADDTOREQUEST \
           from RADSQLRADIUS where TARGETNAME=?
HostSelectParam %1

HostColumnDef 0, Host
HostColumnDef 1, Secret
HostColumnDef 2, Retries
HostColumnDef 3, failurePolicy
HostColumnDef 4, AddToRequest
```

Tip

If a Host has *failurePolicy* defined, and *NoReplyHook* is defined, then *NoReplyHook* is run before the automatic replies are sent.

3.57. <AuthBy INTERNAL>

This clause allows you permanently pre-define how to reply to a request, depending only on the type of request. You can specify whether to **ACCEPT**, **REJECT**, **IGNORE** or **CHALLENGE** each type of request. The default behaviour is to **IGNORE** all requests.

The following result codes are recognised. They are not case sensitive, and may be embedded within a longer string:

- ACCEPT
- REJECT

- IGNORE
- CHALLENGE

This clause can be useful in a number of cases:

- As a fallback at the end of an AuthBy chain, so that if all your authentication methods are failing due to internal problems, you can let all users on, irrespective of password.
- As a step in a chain of AuthBy clauses to update requests or do other processing between AuthBys.
- As a trap for certain types of requests, either in a distinct handler, or at the beginning of a chain of AuthBy clauses.

Tip

The RADIUS protocol does not define an accounting reject message. For accounting requests, REJECT and CHALLENGE are the same as IGNORE.

This example clause will ACCEPT all Access Requests, ACCEPT Accounting Starts and Stops, and REJECT everything else:

```
<AuthBy INTERNAL>
      AuthResult ACCEPT
      AcctStartResult ACCEPT
      AcctStopResult ACCEPT
      DefaultResult REJECT
</AuthBy>
```

<AuthBy INTERNAL> also supports a number of hooks. You can define a Perl hook to handle some or all requests. Requests that are not handled by a hook will be handled according to the result code defined for that type of request. See `goodies/internalhook.cfg` for sample hooks.

Hooks are passed information about the request and the hook is expected to return a list of two values.The values are result code, one of:

```
$main::ACCEPT
$main::REJECT
$main::IGNORE
$main::CHALLENGE
$main::REJECT_IMMEDIATE
```

to indicate the result of the request, and textual reason message providing information about the result. All hooks in *<AuthBy INTERNAL>* are passed the same arguments in this order:

- Reference to the current request, $p
- Reference to the partly formed reply packet for $p
- Reference to additional check items. This can safely be ignored in most circumstances.

Tip

<AuthBy INTERNAL> cannot be used to authenticate any EAP-TLS, TTLS or PEAP protocols directly, but it can be used in conjunction with AuthBy FILE to achieve the same thing:

```
<AuthBy INTERNAL>
```

```
      Identifier myinternal
      .....
</AuthBy>
<Realm DEFAULT>
      <AuthBy FILE>
            Filename %D/users
            EAPType TLS
            .....
```

and in the users file:

```
DEFAULT Auth-Type=myinternal
```

This has the effect of using *<AuthBy FILE>* to do the EAP authentication handling, certificates etc., and the *<AuthBy INTERNAL>* to just authenticate the user name.

<AuthBy INTERNAL> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.57.1. DefaultResult

Specifies how to reply to any request for which there is no more specific result. The default is to IGNORE.

```
# Accept everything not otherwise specified
DefaultResult ACCEPT
```

3.57.2. AuthResult

Specifies how to reply to all Access Requests. There is no default.

3.57.3. AcctResult

Specifies how to reply to all Accounting Requests for which there is no more specific parameter. There is no default.

3.57.4. AcctStartResult

Specifies how to reply to all Accounting Start Requests. There is no default.

3.57.5. AcctStopResult

Specifies how to reply to all Accounting Stop Requests. There is no default.

3.57.6. AcctAliveResult

Specifies how to reply to all Accounting Alive Requests. There is no default.

3.57.7. RequestHook

This optional parameter allows you to define a Perl program that will handle all requests passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.8. AuthHook

This optional parameter allows you to define a Perl program that will handle all Access-Request requests passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.9. AcctHook

This optional parameter allows you to define a Perl program that will handle all Accounting-Request requests passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.10. AcctStartHook

This optional parameter allows you to define a Perl program that will handle all Accounting-Request requests with an Acct-Type of Start passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.11. AcctStopHook

This optional parameter allows you to define a Perl program that will handle all Accounting-Request requests with an Acct-Type of Stop passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.12. AcctAliveHook

This optional parameter allows you to define a Perl program that will handle all Accounting-Request requests with an Acct-Type of Alive passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.13. AcctOtherHook

This optional parameter allows you to define a Perl program that will handle all Accounting-Request requests with an Acct-Type other than Start, Stop and Alive passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.14. OtherHook

This optional parameter allows you to define a Perl program that will handle all requests other than Access-Request and Accounting-Request passed to this AuthBy INTERNAL. For more information about the arguments and the required return value, see Section 3.57. <AuthBy INTERNAL> on page 252.

3.57.15. StripFromRequest

Strips the named attributes from the request before it's processed by the AuthBy. The value is a comma separated list of attribute names. *`StripFromRequest`* removes attributes from the request before *`AddToRequest`* adds any to the request. There is no default.

```
# Remove any NAS-IP-Address and NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.57.16. AddToRequest

Adds attributes to the request before it's processed by the AuthBy. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. *`StripFromRequest`* removes attributes from the

request before *AddToRequest* and *AddToRequestIfNotExist* adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.57.17. AddToRequestIfNotExist

Adds attributes to the request before it's processed by the AuthBy. Unlike *AddToRequest*, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. *StripFromRequest* removes attributes from the request before *AddToRequest* and *AddToRequestIfNotExist* adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name if they are not there already
AddToRequestIfNotExist Calling-Station-Id=1,Login-IP-Host=%h
```

3.58. <AuthBy POP3>

This clause authenticates from a POP3 server, according to RFC1939. It requires the Mail::POP3Client Perl module version 2.9 or better. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3. It supports both plaintext and APOP authentication in the POP server. There is an example configuration file in `goodies/pop3.cfg`. <AuthBy POP3> was mostly contributed by Karl Gaissmaier.

<AuthBy POP3> can support SSL or non-SSL connections to the POP3 server. Use of SSL connections requires IO::Socket::SSL from CPAN and OpenSSL. For more information about CPAN, see Section 2.1.2. CPAN on page 3.

<AuthBy POP3> only supports PAP authentication in incoming RADIUS requests. CHAP and MS-CHAP are not supported, since the plaintext password is not available within Radiator.

<AuthBy POP3> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.58.1. Host

This parameter specifies the host name of the POP server. Defaults to 'pop3'.

```
Host your.pop.server.com
```

3.58.2. Port

This optional parameter specifies the port number to contact on the POP server. Defaults to 110, the standard pop3 port.

```
Port 9000
```

3.58.3. LocalAddr

This optional parameter allows you to specify what local internet address (an optionally port) to bind to. Format is 'xxx.xxx.xxx.xxx[:xx]'.

```
LocalAddr 203.63.154.2
```

3.58.4. AuthMode

This optional parameter specifies what types of POP authentication to permit.

- PASS means use plaintext passwords
- APOP means use APAP (MD5 encrypted) passwords
- BEST means use APOP if possible, else PASS

Defaults to BEST.

```
AuthMode APOP
```

3.58.5. Timeout

This optional parameter specifies a timeout in seconds. If the connection to the POP server is not complete within this time, the authentication will fail with REJECT. Defaults to 10 seconds.

```
Timeout 2
```

3.58.6. Debug

If this optional parameter is set, Mail::POP3Client prints details of its transactions to stdout.

3.58.7. SSLVerify

This optional parameter specifies what sort of SSL server verification that AuthBy POP3 will demand from the POP3 server. The options are ‘none’, ‘optional’ or ‘require’. Defaults to IO::Socket:SSL default.

```
SSLVerify require
```

3.58.8. SSLCAFile

Use this option to locate the file containing the certificates of the trusted certificate authorities. Thus, you can verify that the server certificate has been signed by a reputable certificate authority. Special characters are permitted.

Here is an example of using *`SSLCAFile`*:

```
SSLCAFile %D/certificates/demoCA/cacert.pem
```

3.58.9. SSLCAPath

`SSLCAPath` parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS client certificates. Radiator looks for root certificates first in *`SSLCAFile`*, then in *`SSLCAPath`*, so there usually is no need to set both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files.

Special characters are supported.The certificates and CRLs must be in PEM format, one per file. The file name has a special format. Setting up this directory is described in Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81.

Here is an example of using *`SSLCAPath`*:

```
SSLCAPath %D/cadirectory
```

3.58.10. SSLCAClientCert

This optional parameter specifies the location of the SSL client certificate that this LDAP connection uses to verify itself with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

Here is an example of using *SSLCAClientCert*:

```
SSLCAClientCert %D/certificates/cert-clt.pem
```

3.58.11. SSLCAClientKey

This optional parameter specifies the location of the SSL private key that this connection uses to communicate with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

It is common for the SSL client private key to be in the same file as the client certificate. In that case, both *SSLCAClientCert* and *SSLCAClientKey* refer to the same file.

If *SSLCAClientKey* contains a private key in encrypted format, you need to specify the decryption password in *SSLCAClientKeyPassword*.

Here is an example of using *SSLCAClientKey*:

```
SSLCAClientKey %D/certificates/cert-clt.pem
```

3.58.12. SSLCAClientKeyPassword

If the SSLCAClientKey contains an encrypted private key, then you must specify the decryption password with this parameter. If a key is required, you will generally have been given the password by whoever provided the private key and certificate.

```
SSLCAClientKeyPassword whatever
```

3.59. <AuthBy IMAP>

This clause authenticates from an IMAP server. It requires the Mail::IMAPClient Perl module version 2.2.5 or better. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3. There is an example configuration file in `goodies/imap.cfg`. <AuthBy IMAP> was mostly contributed by Karl Gaissmaier.

<AuthBy IMAP> can support SSL or non-SSL connections to the IMAP server. Use of SSL connections requires IO::Socket::SSL from CPAN and OpenSSL. For more information about CPAN, see Section 2.1.2. CPAN on page 3.

<AuthBy IMAP> only supports PAP authentication in incoming RADIUS requests. CHAP and MS-CHAP are not supported, since the plaintext password is not available within Radiator.

<AuthBy IMAP> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.59.1. Host

This parameter specifies the host name of the IMAP server.

```
Host your.imap.server.com
```

3.59.2. Port

This optional parameter specifies the port number to contact on the IMAP server. Defaults to 143, the standard imap port.

```
Port 9000
```

3.59.3. LocalAddr

Local host bind address.

3.59.4. Timeout

This optional parameter specifies a timeout in seconds. If the connection to the IMAP server is not complete within this time, the authentication will fail with REJECT. Defaults to 10 seconds.

```
Timeout 2
```

3.59.5. Debug

If this optional parameter is set, Mail::IMAPClient prints details of its transactions to stdout.

3.59.6. SSLVerify

This optional parameter specifies what sort of SSL server verification that AuthBy IMAP will demand from the IMAP server. The options are ‘none’, ‘optional’ or ‘require’. Defaults to IO::Socket::SSL default.

```
SSLVerify require
```

3.59.7. SSLCAFile

Use this option to locate the file containing the certificates of the trusted certificate authorities. Thus, you can verify that the server certificate has been signed by a reputable certificate authority. Special characters are permitted.

Here is an example of using *`SSLCAFile`*:

```
SSLCAFile %D/certificates/demoCA/cacert.pem
```

3.59.8. SSLCAPath

`SSLCAPath` parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS client certificates. Radiator looks for root certificates first in *`SSLCAFile`*, then in *`SSLCAPath`*, so there usually is no need to set both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files.

Special characters are supported.The certificates and CRLs must be in PEM format, one per file. The file name has a special format. Setting up this directory is described in Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81.

Here is an example of using *`SSLCAPath`*:

```
SSLCAPath %D/cadirectory
```

3.59.9. SSLCAClientCert

This optional parameter specifies the location of the SSL client certificate that this LDAP connection uses to verify itself with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

Here is an example of using *SSLCAClientCert*:

```
SSLCAClientCert %D/certificates/cert-clt.pem
```

3.59.10. SSLCAClientKey

This optional parameter specifies the location of the SSL private key that this connection uses to communicate with the server. If SSL client verification is not required, then this option does not need to be specified. Special characters are permitted.

It is common for the SSL client private key to be in the same file as the client certificate. In that case, both *SSLCAClientCert* and *SSLCAClientKey* refer to the same file.

If *SSLCAClientKey* contains a private key in encrypted format, you need to specify the decryption password in *SSLCAClientKeyPassword*.

Here is an example of using *SSLCAClientKey*:

```
SSLCAClientKey %D/certificates/cert-clt.pem
```

3.59.11. SSLCAClientKeyPassword

If the SSLCAClientKey contains an encrypted private key, then you must specify the decryption password with this parameter. If a key is required, you will generally have been given the password by whoever provided the private key and certificate.

```
SSLCAClientKeyPassword whatever
```

3.60. <AuthBy LSA>

This module provides authentication against user passwords on a stand alone server, Windows Active Directory or Domain Controller, by using the Windows LSA (Local Security Authority). Since it accesses LSA directly, it can authenticate passwords with PAP, CHAP, MSCHAP, MSCHAPV2, LEAP, EAP-MSCHAP-V2, EAP-TTLS and PEAP.

<AuthBy LSA> is only available on Windows 7/8/8.1/10 and Server 2008/2012/2016/2019, home editions are not supported. It requires the `Win32::Lsa` Perl module from Radiator Software.

Radiator Windows MSI package comes with `Win32::LSA` pre-installed. If you are not using Radiator MSI package, install the `Win32::Lsa` Perl module for Strawberry Perl from the Radiator distribution's `ppm \strawberryperl\` directory:

```
ppm install Win32-Lsa.ppd
```

To use *<AuthBy LSA>*, Radiator must be run on Windows as a user that has the ‘Act as part of the operating system's security policy (SE_TCB_PRIVILEGE) enabled. 'Local System' account that Windows services use by default has this privilege enabled.

Tip

Users can only be authenticated with *<AuthBy LSA>* if they have the 'Access this computer from the network' security policy enabled (this is the normal configuration for Windows Domains). *<AuthBy LSA>* honours the Logon Hours, Workstation Restrictions and 'Account is Disabled' flags in user accounts.

Tip

CHAP passwords can only be authenticated if the user has the 'Store password using reversible encryption' option enabled in their Windows Account. CHAP challenge must also be 16 octets long. This is the default for the most CHAP implementations.

Tip

See `goodies/lsa.cfg` and `goodies/lsa_eap_peap.cfg` for examples on how to configure Radiator to authenticate PAP, CHAP, MSCHAP, MSCHAPV2, LEAP, EAP-MSCHAP-V2, EAP-TTLS and PEAP against Windows user passwords.

Tip

If you are running Radiator on Unix or Linux, and wish to authenticate to Windows Active Directory or to a Windows Domain Controller. For more information, see Section 3.74. <AuthBy NTLM> on page 301.

3.60.1. Domain

This optional parameter specifies which Windows domain will be used to authenticate passwords, regardless of whether the user supplies a domain when they log in. It can be the name of any valid domain in your network. The default is to authenticate against local accounts on the machine that Radiator is running on. Special characters are permitted.

```
Domain OPEN
```

3.60.2. DefaultDomain

This optional parameter specifies the Windows Domain to use if the user does not specify a domain in their username. Special characters are supported. Can be an Active directory domain or a Windows NT domain controller domain name. Empty string (the default) means the local machine.

```
DefaultDomain OPEN
```

3.60.3. Workstation

This optional parameter specifies a workstation name that will be used to check against workstation logon restrictions in the users account. If the user has any workstation restrictions specified in their account, this is the workstation name that will be used to check the restriction. Defaults to an empty string, which means that LSA will not check any workstation logon restrictions.

```
Workstation WLAN
```

3.60.4. ProcessName

This optional parameter specifies a process name for LSA internal logging. Defaults to ‘Radiator’.

3.60.5. Origin

This optional parameter specifies a request origin name for LSA internal logging. Defaults to ‘Radiator’.

3.60.6. Source

This optional parameter specifies a source name for LSA internal logging. Defaults to ‘Radiator’.

3.60.7. LSARewriteHook

This optional parameter allows you to define a Perl function to rewrite the username that is passed to LSA. Username passed to LSA API is changed to whatever is returned by this function. The username in request is not changed. This may be needed, for example, with Wi-Fi roaming where roaming username can not be directly used with Windows authentication because of local naming conflicts with roaming requirements.

The following parameters are passed to LSARewriteHook:

- *`$_[0]: $p`*, the current Radius::Radius request object
- *`$_[1]: $user`*, the current username to pass to LSA

Here are some examples:

```
# We use file instead of inline code
LSARewriteHook file:"%D/lsa-rewrite-hook.pl"
```

```
# Use inline code to change our global roaming realm to windows domain
LSARewriteHook sub { my ($user) = $_[1]; \
    $user =~ s/example\.com\z/org.local/; \
    return $user; }
```

3.60.8. Group

This optional parameter allows you to specify that each user must be the member of at least one of the named Windows Global or Local groups. More than one required group can be specified, one per Group line. Requires Win32::NetAdmin (which may require separate installation with Strawberry Perl). If no Group parameters are specified, then Group checks will not be performed.

```
# Each user must be in Administrators and/or Domain Users
Group Administrators
Group Domain Users
```

3.60.9. DomainController

This optional parameter is used only if one or more Group check parameters are set. It specifies the name of the Windows Domain Controller that will be used to check each users Group membership. If no Group parameters are specified, DomainController will not be used. Defaults to empty string, meaning the default controller of the host where this instance of Radiator is running.

3.61. <AuthBy SOAP>

This module handles authentication and accounting by sending it to a remote RADIUS server over TCP using the SOAP protocol. Each RADIUS request is transformed into a SOAP request, which is sent by HTTP or HTTPS to a remote SOAP server. The Remote SOAP server can be any implementation, but example SOAP server code is provided with Radiator.

AuthBy SOAP can be useful in order to tunnel RADIUS requests through ports 80 or 443 in a firewall, where UDP port 1645 is not permitted thought the firewall. It can also be used to improve reliability in some environments by using TCP rather than UDP. And RADIUS security can be improved by using HTTPS as the transport protocol, which will encrypt all the contents of the RADIUS request. Requires the SOAP::Lite Perl module and its prerequisites.

AuthBy SOAP uses a simple SOAP interface. An example Web Service Description Language file (`goodies/soaprequest.wsdl`) is provided for people wishing to implement their own SOAP RADIUS client.

Included with Radiator is a sample SOAP server CGI script and a SOAP handler module. The file `goodies/soapradius.cgi` receives SOAP radius requests as sent by AuthBy SOAP and invokes Radius::SOAPRequest::radius(), which is provided by `Radius/SOAPRequest.pm`. SOAPRequest transforms incoming SOAP requests into standard RADIUS requests and sends them to a RADIUS server. The RADIUS reply is sent back to AuthBy SOAP as the SOAP reply. In order to install the SOAP server, you will need to edit `SOAPRequest.pm` and specify the location of your RADIUS dictionary, and the address and port of the destination RADIUS server. The address defaults to localhost: 1647. You must also install `soapradius.cgi` in a suitable location in your web server's CGI directory. You may also need to edit `soapradius.cgi` if your Radiator per modules are not installed in the usual place.

Tip

See `goodies/soap.cfg` for an example configuration file.

3.61.1. Endpoint

With this parameter, you can specify any number of SOAP proxy points. AuthBy SOAP will try to contact each one in turn until the SOAP call succeeds in getting a reply. Defaults to `http://localhost/cgi-bin/soapradius.cgi`.

```
Endpoint https://your.server.com/cgi-bin/soapradius.cgi
```

3.61.2. URI

This parameter specifies the SOAP URI that AuthBy SOAP will try to run. This is not a URL. It is used by the server to deduce the right SOAP module to load. You should not need to change this. Defaults to http://www.open.com.au/Radius/SOAPRequest.

3.61.3. SOAPTrace

This enables some or all of the SOAP::Lite internal tracing. Allowable values are

- transport
- dispatch
- result
- parameters
- headers

- objects
- method
- fault
- freeform
- trace
- debug
- all

or any combination. Defaults to no tracing. Tracing is printed to STDOUT.

```
SOAPTrace all
```

3.61.4. Timeout

With this optional parameter, you can control how long to wait for the SOAP reply from the SOAP server. Time is in seconds. Defaults to 3 seconds.

3.62. <AuthBy OTP>

This module is extensible and customisable to support a range of One-Time-Password (OTP) schemes, including automatic password generation and sending of passwords through a back-channel such as SMS. AuthBy OTP is suitable for authenticating 802.1X Wired and Wireless access with custom one-time password and token card authentication systems.

The default behaviour of AuthBy OTP demonstrates how it can be used and tested, but it is not suitable for use in a production environment: it tells the user the correct password in the challenge. In almost all cases, you will need to develop at least your own ChallengeHook, and possible a VerifyHook to work with your local system. See `goodies/otp.cfg` for a sample configuration file.

In the most common use of AuthBy OTP, it will be configured to generate a random password (according to a configurable password pattern) and then send it to the user by SMS or some other channel. AuthBy OTP will then challenge the user to enter the correct password (after they have received it through the SMS system or whatever). In order to achieve this, you must configure at least the *`ChallengeHook`* to call some external program that will deliver the password to the user.

AuthBy OTP works with EAP-OTP (One-Time-Password), EAP-GTC (Generic-Token-Card) as well as standard RADIUS PAP. Caution: some clients may not handle OTP challenges very well. AuthBy OTP supports PAP in the following way: if the user attempts to log in with an empty (zero length) password, the ChallengeHook will be called and the challenge will be sent back to the client. This may result in a message for the user, but often does not, depending on the client on the users computer.

Tip

You can test AuthBy OTP with the following radpwtst commands:

```
# Conventional RADIUS PAP
radpwtst -noacct -interactive -password ''
# EAP-OTP authentication
radpwtst -noacct -eapotp
# EAP-GTC auth (with EAPType set to Generic-Token):
# radpwtst -noacct -eapgtc
```

3.62.1. ChallengeHook

ChallengeHook is a fragment of Perl code that is expected to generate a OTP (if necessary) save the OTP (in $context is sometimes convenient) and send the OTP to the user by a back channel (if necessary). It should return a challenge string that will be presented to the user by the client, informing them of how to get or generate their password.

It is passed the following arguments:

- Reference to the current AuthBy module object
- User name
- Current RADIUS request packet
- User context that will be available later in VerifyHook. It can be used to store information such as the correct password until later in the authentication process.

The default ChallengeHook generates a random password according to PasswordPattern, saves it in the context and returns a challenge message telling the user what the correct password is. The default ChallengeHook must not be used in a production environment.

This example shows how to generate a random password and pass it to an external program which must deliver it to the user through some back channel like SMS. The example just echoes it to stdout. You can see that the generate_password() function can be used to generate a random password that conforms to PasswordPattern. The password is stored in the context so it can be checked later in the VerifyHook.

```
ChallengeHook sub {my ($self, $user, $p, $context) = @_;\
      $context->{otp_password} = $self->generate_password();\
      system('/bin/echo', "in sample ChallengeHook for", \
      $user, "password is", $context->{otp_password});\
      return "Your OTP password has been printed by Radiator on STDOUT";}
```

3.62.2. VerifyHook

VerifyHook is a fragment of Perl code that is expected to validate a OTP and return 1 on success. You will need to specify your own VerifyHook if you require an external program to verify the correct OTP.

VerifyHook is passed the following arguments:

- Reference to the current AuthBy module object
- User name
- Submitted OTP password in plaintext
- Current RADIUS request packet
- Same user context that was passed to ChallengeHook for this user when the challenge was generated. $context->{otp_password} will generally contain the correct plaintext password for this user.

The default VerifyHook compares the submitted password to $context->{otp_password}.

This example VerifyHook only accepts the password 'xyzzy':

```
VerifyHook sub {my ($self,$user,$submitted_pw,$p,$context)=@_;\
return $context->{otp_password} eq 'xyzzy';}
```

3.62.3. PasswordPattern

This optional parameter specifies a character pattern that will be used to generate random passwords by generate_password() and the default ChallengeHook.

- a

  Lowercase alphanumeric (abcdefghijklmnopqrstuvwxyz0123456789)

- A

  Uppercase alphanumeric (ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789)

- c

  lowercase consonant (bcdfghjklmnpqrstvwxyz)

- C

  uppercase consonant (BCDFGHJKLMNPQRSTVWXYZ)

- v

  lowercase vowel (aeiou)

- V

  uppercase vowel (AEIOU)

- 9

  numeric (0123456789)

- anything else is used literally

The default is cvcvcvc99 which produces passwords like:

```
vosuyic04
rezeqek86
jocupon50
```

3.62.4. ContextTimeout

This optional parameter specifies how long (in seconds) the context passed to ChallengeHook for a user will be kept. It defaults to 120 seconds, which is expected to be enough time for most users to receive and enter their correct OTP. You should not need to change this.

3.63. <AuthBy RSAAM>

This module provides authentication via RSA Authentication Manager AM 7.1 and later. AM 7.1 provides more features than the ACE server and RSA Mobile servers it replaces.

AM 7.1 supports traditional SecurID two-factor token cards, as well as static passwords. It also supports OnDemand tokencodes, where a random tokencode is sent to the user via email or SMS. It also supports authentication through a series of user-configurable security questions. All these authentication methods are supported by AuthBy RSAAM.

AuthBy RSAAM can authenticate the following protocols against AM. Note that CHAP, MSCHAPV1, MSCHAPV2 and EAP-MSCHAPV2 cannot be authenticated against AM.

- PAP
- TTLS-PAP

- EAP-GTC
- EAP-OTP
- PEAP-GTC

AuthBy RSAAM works on all platforms supported by Radiator, including Windows, Linux, Solaris, Unix etc. AuthBy RSAAM connects the AM server by SSL and SOAP, and therefore required the following Perl modules from CPAN:

- SOAP::Lite and its prerequisites
- Either Crypt::SSLeay or IO::Socket::SSL
- Net::SSLeay

For more information, see Section 2.1.2. CPAN on page 3.

Tip

Sample configuration files are provided in the goodies directory of your distribution in `rsaam.cfg` and `eap_peap_gtc_rsaam.cfg`.

Tip

RSA AM is not able to specify the preferred authentication policy to use for each user. Therefore, if you need to use different authentication policies for different groups of user, you will need an <AuthBy RSAAM> clause for each policy, and then direct requests to the appropriate clause using one of the many methods supported by Radiator.

Tip

AuthBy RSAAM returns IGNORE if it unable to communicate with its configured AM server.
This means you can chain several AuthBy RSAAM clauses together using AuthByPolicy ContinueWhileIgnore to implement failover from one AM server to another in the event of AM server unavailability.

Tip

In some circumstances, The Radiator connection to RSA AM may fail with an error message in the RSA Weblogic server like:

```
Received fatal alert: bad_record_mac at sun.reflect.NativeConstructorAccessorImpl.
newInstance0
```

This can be fixed by adding these lines to the weblogic server start file:

- `Dhttps.protocols=SSLv3,TLSv1`
- `Dsun.security.ssl.allowLegacyHelloMessages=true`
- `Dsun.security.ssl.allowUnsafeRenegotiation=true`

Configuring Authentication Manager for AuthBy RSAAM

In order to configure Authentication Manager to work with AuthBy RSAAM:

1. Install RSA AM 7.1 on your platform of choice, or Install 8.0 virtual appliance
2. Install Radiator on your platform of choice. It may be the same as the AM 7.1 host, or a different one in case of AM 8.0.
3. Install SOAP::Lite and its prerequisites on the Radiator host.
4. Starting with one of the sample RSAAM configuration files, configure Radiator.
5. Get the user name and password required for AuthBy RSAAM to connect to AM. These commands will print out the user name and password that AM automatically generates during installation.

   Do this on AM7.1 or earlier:

   ```
   cd "C:\Program Files\RSA Security\RSA Authentication Manager\Utils rsautil
         manage-secrets -m <MASTERPWD> -a list
   ```

   Do this on AM 8.0:

   ```
   cd /opt/rsa/am/utils
   ./rsautil manage-secrets --action list
   ```

   This will print out the user name and password required for Radiator to connect to AM 7.1 or 8.0. Enter the user name and password as SessionUsername and Session- Password in your Radiator configuration file.
6. Select which authentication method you will use to authenticate all your users. Set Policy in your Radiator configuration file.
7. Set Host in your Radiator configuration file to the FQDN (fully qualified domain name) and port number of your AM host. For example

   ```
   Host boodgie.open.com.au:7002
   ```
8. Add and configure a test user to AM. If required allocate a token to the user.
9. Start Radiator and test with a command like:

   ```
   radpwtst -noacct -user username -password password -interactive -timeout 60
   ```

3.63.1. Host

This parameter specifies the address and port number of the RSA AM server. It is used as %1 in the Endpoint parameter. The default is "localhost:7002". You will have to change this to the hostname/address and port number of your RSA AM server, since by default AM does not listen on localhost. 7002 is the usual port number for RSA AM.

3.63.2. Endpoint

This optional parameter specifies how to create the endpoint of the SOAP connection to the RSA AM server. Special characters are permitted. %0 is replaced by the value of the Protocol parameter (for more information, see Section 3.63.3. Protocol on page 269) and %1 is replaced by the value of the Host parameter (for more information, see Section 3.63.1. Host on page 268). The default is **`%0://%1/ims-ws/ services/ CommandServer`**.

You should not normally need to change this from the default.

3.63.3. Protocol

This optional parameter specifies the protocol that will be used to contact the RSA AM server. It is used as %0 in the Endpoint parameter. The default is "https". You should not normally need to change this.

3.63.4. URI

This optional parameter specifies the SOAP URI that will be accessed in the RSA AM server. The default is "http://webservice.rsa.com/". You should not normally need to change this. Note that this is not the address of a web resource and it is not accessed by Radiator during authentication.

3.63.5. Policy

This optional parameter specifies the authentication policy that is to be used. Defaults to RSA_Password.

Options are:

- SecurID_Native

  Traditional SecurID two-factor token cards. User enters their PIN followed by the tokencode currently showing on their token card.

- OnDemand

  User enters their PIN. AM sends a temporary tokencode to the user by email or SMS, according to however AM is configured. User then enters the tokencode they receive.

- RSA_Password

  Static password stored in the RSA internal database.

- LDAP_Password

  Static password stored in an LDAP database.

- Security_Questions

  User is asked a series of security questions, and enters answers that they have previously configured using the RSA Self-Service Console.

- SecurID_Proxy

3.63.6. SessionUsername

User name used to authenticate the SSL connection to AM. Created automatically by AM during installation. For more information, see Section 3.63.1. Host on page 268.

3.63.7. SessionPassword

Password used to authenticate the SSL connection to AM. Created automatically by AM during installation. For more information, see Section 3.63.1. Host on page 268

3.63.8. ChallengeHasPrompt

Add RADIUS Prompt attribute to Access-Challenge messages. Prompt value is based on responses received from RSA AM. Default is not to add Prompt in Access-Challenges. Prompt attribute is a hint to the client software to echo or not echo sensitive user input such as PINs or security question answers.

3.63.9. SessionRealm

Tip

This is now an obsolete parameter.

Previously, the realm name used to authenticate the SSL connection to AM.

3.63.10. Timeout

This optional parameter specifies the timeout in seconds that will be used during authentication requests sent by Radiator to the RSA AM server. The default is 20 seconds.

3.63.11. SOAPTrace

This optional parameter enables low level protocol tracing in the SOAP::Lite module. Setting it to ”debug” will cause details of each incoming and outgoing SOAP request to be printed on STDOUT.

3.63.12. Message

This optional parameter enables customisation of various user messages generated by this module. The key for each message is the RSA AM message, and the value is the string you want the user to see.

3.63.13. SSLVerify

May be used to control how the Server's certificate will be verified. May be one of "none" or "require".

3.63.14. SSLCAFile

Use this option to locate the file containing the certificates of the trusted certificate authorities. Thus, you can verify that the server certificate has been signed by a reputable certificate authority. Special characters are permitted.

Here is an example of using *`SSLCAFile`*:

```
SSLCAFile %D/certificates/demoCA/cacert.pem
```

3.63.15. SSLCAPath

`SSLCAPath` parameter specifies the name of a directory containing CA root certificates that may be required to validate TLS client certificates. Radiator looks for root certificates first in *`SSLCAFile`*, then in *`SSLCAPath`*, so there usually is no need to set both. When Certificate Revocation List (CRL) checks are enabled, this directory is also used by TLS library to look for CRL files.

Special characters are supported.The certificates and CRLs must be in PEM format, one per file. The file name has a special format. Setting up this directory is described in Setting up this directory is described in Section 3.11.3. TLS_CAPath on page 81.

Here is an example of using *`SSLCAPath`*:

```
SSLCAPath %D/cadirectory
```

3.63.16. SSLVerifyCNName and SSLVerifyCNScheme

SSLVerifyCNName sets the name which is used when verifying the hostname against the certificate presented by RSA AM HTTPS server. *SSLVerifyCNScheme* controls how the verification is done, for example, if wildcards are allowed. For more information, see IO::Socket::SSL [https://metacpan.org/pod/IO::Socket::SSL].

The following allow wildcard certificate name *.example.com to match.

```
SSLVerifyCNName example.com
SSLVerifyCNScheme http
```

3.63.17. SSL_CertificateFile

Specifies the name of a client certificate file which will be use to authenticate SSL connection to the AM server. The certificate will be sent to the AM server SSL authentication. The certificate file must be in PEM. The certificate file can also contain the client's TLS private key if the SSL_PrivateKeyFile parameter specifies the same file. Not required if AM does not require client certificate authentication.

3.63.18. SSL_PrivateKeyFile

Specifies the name of the file containing the SSL client's private key. It is sometimes in the same file as the client certificate (SSL_CertificateFile). The private key must not be encrypted and must not require a passphrase.

3.64. <AuthBy SQLDIGIPASS>

Tip

<AuthBy SQLDIGIPASS> was previously called *<AuthBy DIGIPASS>*. *<AuthBy SQLDIGIPASS>* is completely compatible with and replaces *<AuthBy DIGIPASS>*. Although *<AuthBy DIGIPASS>* is still a recognised name, it is officially deprecated, and support for it may be removed in the future. New installations are encouraged to use *<AuthBy SQLDIGIPASS>*.

This module provides authentication of Vasco Digipass tokens from an SQL database. For more information, see Vasco website [https://www.vasco.com/]. Digipass tokens are small hand-held devices that generate one-time passwords that change every minute. They can be purchased from Vasco and issued to your users. Such tokens provide much higher levels of security than static passwords. Additionally, with some types of token, users can set up individual PINs, which provides even higher levels of security with two-factor authentication. Some types of Digipass token can operate in a Challenge-Response mode. Vasco Digipass is supported by Radiator on Solaris, Linux, and Windows.

<AuthBy SQLDIGIPASS> supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

<AuthBy SQLDIGIPASS> can be used to authenticate PAP, CHAP, MSCHAP, MSCHAPV2, EAP-MSCHAPV2, EAP-OTP, and EAP-GTC protocols.

<AuthBy SQLDIGIPASS> supports for Response Only (RO) and Challenge/Response (CR) tokens. It supports RADIUS PAP, TTLS-PAP, EAP-GTC, and EAP-OTP authentication methods. When using Challenge/Response tokens with PAP or TTLS-PAP, when the user enters an empty password, *<AuthBy SQLDIGIPASS>* generates the Challenge to enter into the Digipass token. The token then generates a Response which the users enters as their real password.

Radiator and *<AuthBy SQLDIGIPASS>* can be configured in several ways including:

- As a simple stand-alone system. A single SQL table contains information about each Digipass token and the user it is assigned to. You can use the `digipass.pl` program supplied with `Authen::Digipass` to import tokens, assign them to users and otherwise administer tokens and users. The example `digipass.cfg` Radiator configuration file shows a simple example of how to configure Radiator for such a system. Sample SQL database table definition files are provided with Radiator for a range of free and commercial SQL databases.
- As an addition to a Radiator-compatible user-management system or ISP billing system. In this mode, Radiator is configured to authenticate using *<AuthBy SQLDIGIPASS>* from an SQL table, but also uses other information from the user-management system to save usage data, get user- or service-specific RADIUS reply items and so on.
- In conjunction with Radiator Software's RAdmin RADIUS user management system. RAdmin provides an easy-to-install, easy-to-use web-based graphical system for managing RADIUS users for dial-up, wired and wireless authentication. RAdmin version 1.9 includes support for importing, allocating and administering Digipass tokens for authenticating users against Digipass instead of static passwords. RAdmin also works with any free or commercial SQL database. For more information about RAdmin, see RAdmin website [https://radiatorsoftware.com/products/radmin/].

<AuthBy SQLDIGIPASS> requires an additional `Authen::Digipass` module to be installed. The `Authen::Digipass` Perl module provides access to the Vasco Controller software that does the authentication of each token. Contact Radiator Software to obtain `Authen::Digipass` module for Solaris, Linux and Windows. See `goodies/digipass-install.txt` in your distribution for details on how to install and test `Authen::Digipass` for your platform.

<AuthBy SQLDIGIPASS> also requires an SQL database to hold information about each Digipass token that your system knows about. When you purchase a Digipass token from Vasco, you get also a DPX file that contains important data about the token. This DPX file must be imported into the *<AuthBy SQLDIGIPASS>* database before the token can be authenticated by Radiator. You can use any free or commercial SQL database with *<AuthBy SQLDIGIPASS>*.

RAdmin provides all the tools you need for importing, allocating and administering Digipass tokens and users. In this case, use `goodies/radminDigipass.cfg` as an example Radiator configuration.

If you intend to use Digipass tokens with the example SQL database schema supplied in `goodies/*.sql`, use the `digipass.pl` program supplied and installed with `Authen::Digipass` for importing, allocating and administering Digipass tokens and users. In this case, use `goodies/digipass.cfg` as an example Radiator configuration.

Using Vasco Digipass tokens to generate passwords for AuthBy SQLDIGIPASS

Vasco produces a number of different types of token, some which require a PIN to be entered into the token, some which support user selected PINs, and some which do not support PINs. All tokens display a ‘tokencode’, usually in response to pressing a button on the face of the token. The different types of token are described below.

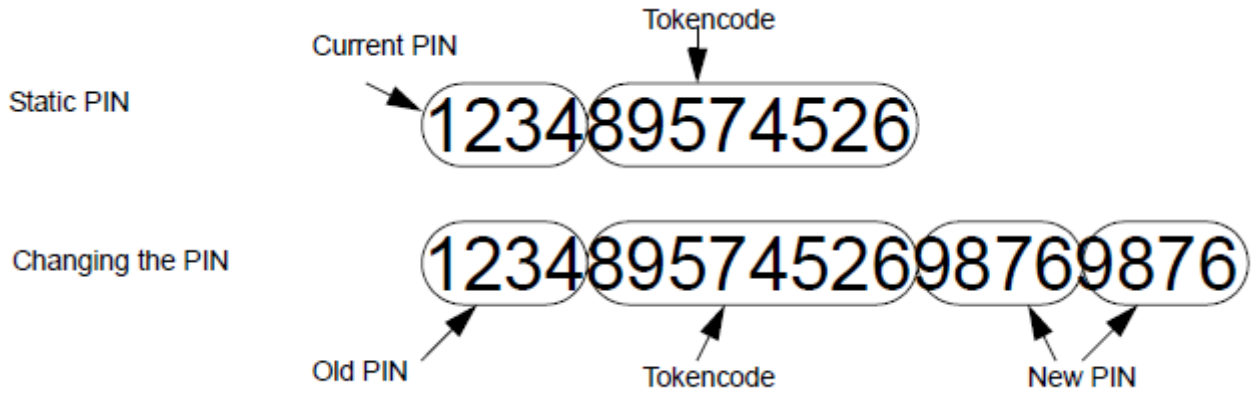
The simplest Digipass tokens include the Go-3, and which have an LCD display and a single button but no keypad. For these tokens, you press the button and use the tokencode displayed on the token as your password. Note that any attempt to use the same tokencode twice results a rejection. The error message on the Radiator log says ‘Code Replay Attempt’.

Go-1 and Go-3 tokens among others support user-selected static PINs, and also support the ability to change your PIN dynamically. Your initial PIN may have been selected by your administrator and given to you with your token. In any case, if you have a PIN, the password is made from the PIN followed by the tokencode displayed on the token. For example, if your PIN is 1234, and the token displays 89574526, then your password

is ‘123489574526’. If you have not been assigned a PIN for your token, just use the tokencode displayed on the token as your password: ‘89574526’.

Go-1 and Go-3 tokens (among others) also support the ability to change your PIN. To change your PIN, you form the login password from your current PIN, followed by the tokencode, followed by the new PIN, followed by the new PIN again. For example, if your current PIN is 1234, and you wish to change the PIN to 9876, and the token is displaying 89574526, then you use the password 12348957452698769876. After this password has been accepted, your new PIN is active, and you must enter the new PIN at the beginning of each subsequent password.

Figure 2. Digipass Go-1 PIN passwords



After resetting the PIN (using either the `digipass.pl` application that comes with `Authen::Digipass` or the ‘Reset static password (PIN) for this token’ button in RAdmin), the token has no PIN associated with it. In that case, provide a new PIN with the next authentication attempt.

To do this enter the tokencode from the token followed by the new PIN twice. For example, if the current tokencode is 656565 and you want your new PIN to be 1234, enter 65656512341234 at your next authentication. For subsequent authentications, prefix the token code with your new PIN.

Some Digipass tokens have a keypad, which requires the token's PIN to be entered, and which also support Challenge-Response (CR). In Challenge-Response, when you first attempt to log in, Radiator sends a ‘Challenge’ (a sequence of digits) that you must enter into the token in order to generate the correct Response, which is then used as your password.

The first step is to attempt to log in with an empty password (i.e. a password with nothing in it). If the user's token supports CR, Radiator *<AuthBy SQLDIGIPASS>* sends a challenge of (typically) 4 digits:

```
Digipass Challenge: 8077
```

You use this challenge shortly. Now start the token by pressing the arrow button. The token asks for your PIN. Enter the 4 digit PIN (e.g. 1 - 2 - 3 - 4). The token will then display ‘APPL1’. Press ‘3’ to request Challenge-Response. The token displays ‘- - - -’. Enter the Digipass Challenge sent to you by Radiator (8077 in this example). The token displays a tokencode of 7 digits. Use the tokencode as your password.

Virtual Digipass

<AuthBy DIGIPASS> supports Virtual Digipass. Virtual Digipass tokens to do require an actual physical token. When the user attempts to log in, *<AuthBy SQLDIGIPASS>* generates the correct password and sends it to the user by a separate secure method such as SMS, voice etc. The user then enters the tokencode they receive from the separate channel, and *<AuthBy SQLDIGIPASS>* authenticates it just like a normal Digipass tokencode. Not all Vasco tokens support Virtual Digipass. This is determined by a flag in the token's DPX data file. In order to use Virtual Digipass, you need to purchase one or more Virtual Digipass token data files from Vasco. Real

physical Digipass tokens can have DPX token files that also permit use of Virtual Digipass as a backup for the case where the token is lost or stolen.

Virtual Digipass allows Vasco token support even if the user does not have a physical token or has lost it. The SupportVirtualDigipass parameter makes *<AuthBy SQLDIGIPASS>* support Virtual Digipass tokens: If the incoming password is empty, and the token supports Virtual Digipass, *<AuthBy SQLDIGIPASS>* generates the user's correct tokencode and passes it to the VirtualTokencodeHook for delivery to the user by some secure out-of-band method such as SMS.

3.64.1. AuthSelect

This optional parameter specifies the SQL query that will be used to fetch Digipass data from the database. Special characters are permitted, and %0 is replaced with the quoted user name. Defaults to select DP_DATA, DIGIPASS from TBL_VASCODP where USER_ID=%0, which is compatible with the sample schemas provided in goodies/*.sql. The query is expected to return 2 fields in this order:

1. Digipass data block. This is 248 characters of encrypted data about the Digipass. It is encoded in printable characters.

   Example: `10AFIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGAAEAAAAgICAAAACx4 bwOmRYm8XYHErnjBAJIAWecbdhsRHIAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAA +RmOzmh1lFwAAAYAAQAAAERQMzAwAAAAAAAAAAAAACNOQpB57zTB/ XrAgg+7W/y`

2. The Digipass serial number of 22 characters, including the token number and application name, and often some spaces, Example: '0097123456APPL 1 '. The spaces and application name are significant.

3.64.2. UpdateQuery

This optional parameter specifies the SQL query that will be used to store Digipass token data back to the database after authentication. The process of authentication (whether or not it is successful) results in the Digipass data block for that token being changed, and it must be written back to the database after each authentication attempt for correct operation. Special characters are permitted, and %0 is replaced with the new data block, and %1 is replaced with the Digipass serial number retrieved by AuthSelect.

Defaults to update TBL_VASCODP set DP_DATA='%0' where DIGIPASS='% 1', which is compatible with the sample schemas provided in `goodies/*.sql`.

3.64.3. ITimeWindow

This optional parameter specifies the size of the window of opportunity that a token can login with (this is counted in multiples of one-time password "rollovers" in the token. Value can be 2 to 1000. Default is 100 (that means +- 30 minutes for default tokens)

3.64.4. IThreshold

This optional parameter specifies the number of times that a person can unsuccessfully try to login before being locked out. 0 means disabled. Defaults to 0.

3.64.5. SyncWindow

This optional parameter specifies the size of the larger window that is created for use the first time after a token has been reset. This means that if a token gets out of sync (which is not an often occurrence), the user cannot log in so the administrator resets the token, then a larger sync window is produced after the reset so that the token can be recognised and calibrated by the software to allow subsequent use. This parameter is expressed in hours. Value can be 1 to 60. Default is 6 (hours).

3.64.6. CheckChallenge

This optional parameter specifies whether or not to check if the challenge has been corrupted before validation. Value can be 0 to 4:

- 0: No password checking
- 1: Check the parameter then verify (default)
- 2: Always use the DPData to validate responses
- 3: Avoid Challenge-Response Replay Attack by allowing only one challengeresponse authentication per timestep
- 4: Avoid Challenge-Response Replay Attack by rejecting the second response if responses from two consecutive authentication requests are equal and in the same time-step

3.64.7. ChkInactDays

This optional parameter specifies number of days of token inactivity. Past this number of days, the token will have to be reset. Values from 0 to 1024. Default is 0, which means the feature is disabled.

3.64.8. DeriveVector

This optional advanced parameter can be used to make data encryption unique for a host. Defaults to 0x00000000.

3.64.9. EventWindow

This optional advanced parameter specifies the Event Window size by number of iterations. Represents the acceptable event counter difference between Digipass token a and the host. It only applies to event-based operating modes. From 10 to 1000. Defaults to 100.

3.64.10. HSMSlotId

This optional advanced parameter specifies the HSM slot ID which will be used to store the Storage and Transport keys. 0 to 60. Defaults to 0.

3.64.11. StorageKeyId

This optional advanced parameter specifies the key which will be used to decrypt the Digipass data retrieved from the database. 0x00000000 to 0xffffffff. Defaults to 0x00000000.

3.64.12. TransportKeyId

This optional advanced parameter specifies the key which will be used to encrypt the Digipass data written to the database. 0x00000000 to 0xffffffff. Defaults to 0x00000000.

3.64.13. StorageDeriveKey1, StorageDeriveKey2, StorageDeriveKey3, StorageDeriveKey4

These optional advanced parameters specify the derivation keys used to make data encryption unique for a host.

3.64.14. ChallengeMessage

This parameter allows you to customise or internationalise the Reply-Message sent when the user is challenged to enter a Digipass tokencode. %0 is replaced with the digipass challenge string. Defaults to 'Digipass Challenge: %0'

```
# French challenge message:
ChallengeMessage Votre challenge du Digipass: %0
```

3.64.15. SupportVirtualDigipass

This optional parameter causes AuthBy SQLDIGIPASS to support Vasco Virtual Digipass tokens.

3.64.16. VirtualTokencodeHook

If the SupportVirtualDigipass flag is enabled, this parameter specifies Perl code that is called whenever a Virtual Digipass tokencode is to be sent to a user. The hook is expected to transmit the tokencode to the user over some prompt, secure out-of-band method, such as SMS. The VirtualTokencodeHook is called like:

```
VirtualTokencodeHook($self, $username, $tokencode, $p)
```

VirtualTokencodeHook must return an error message if it fails to start delivery of the tokencode the user, otherwise it must return undef.

3.64.17. ChallengeTimeout

This optional parameter sets the maximum period of time that a challenge from a Challenge-Response (CR) token will be valid for. Time is in seconds and defaults to 300 seconds (5 minutes).

3.65. <AuthBy LDAPDIGIPASS>

This module provides authentication of Vasco Digipass tokens from an LDAP database. For more information, see Vasco website [https://www.vasco.com/]. For more information about details about Digipass tokens, how to obtain and operate them, see Section 3.64. <AuthBy SQLDIGIPASS> on page 271.

<AuthBy LDAPDIGIPASS> requires an additional Authen-Digipass module to be installed. The Authen-Digipass Perl module provides access to the Vasco Controller software that does the authentication of each token. Radiator includes pre-compiled binaries of the Authen-Digipass module for Solaris, Linux, and Windows. The Authen-Digipass module also includes the `digipass.pl` command line application for administering Digipass tokens in SQL and LDAP databases. See `goodies/digipassinstall.txt` in your distribution for details on how to install and test Authen-Digipass for your platform.

<AuthBy LDAPDIGIPASS> can be used to authenticate the following protocols:

- PAP
- CHAP
- MSCHAP
- MSCHAPV2
- EAP-MSCHAPV2
- EAP-OTP
- EAP-GTC

<AuthBy LDAPDIGIPASS> can be configured to work in different LDAP environments and schemas. The example file `goodies/radiator-ldap.schema` has a sample LDAP schema to hold Digipass token data. This schema is suitable for OpenLDAP and other compatible LDAP servers. See the notes in the top of that file for details on how to install the schema in your LDAP server, so that you can use it to store Digipass token data. You can use `digipass.pl` program included in the Authen-Digipass module to import token data into this example schema, and to assign them to users, reset tokens, get detailed token information, and so on. By

default, *<AuthBy LDAPDIGIPASS>* works with this sample schema on a local LDAP database, but you can use the *<AuthBy LDAPDIGIPASS>* parameters to configure it to work with other schemas and databases. There is also a sample Radiator configuration file in `goodies/digipass_ldap.cfg`.

This clause supports all the common LDAP configuration parameters. For more information about the LDAP configuration parameters, see Section 3.9. LDAP configuration on page 53.

<AuthBy LDAPDIGIPASS> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.65.1. SearchFilter

This is the LDAP filter to use when searching for the user. It is used in similar way as with all LDAP modules. For more information, see Section 3.9.2. SearchFilter on page 54.

This optional parameter specifies the LDAP search filter that is used to find the Digipass token record for the user attempting to log in. Special characters can be used. **%0** is replaced by *UsernameAttr* and **%1** by the user name, whose token is searched. The default value is **(%0=%1)**.

3.65.2. UsernameAttr

This optional parameter gives the name of the LDAP attribute that contains the user name of the user assigned to that token. It is used as *%0* in the *SearchFilter*. The default value is **oscDigipassTokenAssignedTo**.

3.65.3. TokenDataAttr

This parameter specifies the name of the LDAP attribute that contains the Digipass token data, which is used to authenticate Digipass token logins. Defaults to oscDigipassTokenData.

3.65.4. MaxRecords

This optional parameter specifies the maximum number of Digipass tokens returned by the SearchFilter that will be examined. Defaults to 1.

3.65.5. BaseDN

This is the base DN where searches will be made. For each authentication request, Radiator does a SUBTREE search starting at BaseDN, looking for a UsernameAttr that exactly matches the user name in the radius request (possibly after user name rewriting).

Special formatting characters are permitted. **%0** is replaced by *UsernameAttr* and **%1** by the user name, whose token is searched.

Here is an example of using *BaseDN* with *<AuthBy LDAPDIGIPASS>*:

```
# Start looking here
BaseDN o=University of Michigan, c=US
```

Note

On some LDAP servers, you can get a significant performance increase by narrowing the search to the exact uid you are interested in. This example restricts the search to uid=username,ou=foo,o=bar,c=au:

```
BaseDN      %0=%1,ou=foo,o=bar,c=au
Scope       base
```

3.65.6. Vasco Controller Library parameters

These parameters can be used to control the behaviour of the Vasco Controller Library:

- UpdateQuery
- ITimeWindow
- IThreshold
- SyncWindow
- CheckChallenge
- ChkInactDays
- DeriveVector
- EventWindow
- HSMSlotId
- StorageKeyId
- TransportKeyId
- StorageDeriveKey1
- StorageDeriveKey2
- StorageDeriveKey3
- StorageDeriveKey4
- ChallengeMessage

For more information about these parameters, see Section 3.64. <AuthBy SQLDIGIPASS> on page 271.

3.66. <AuthBy LDAP_APS>

This clause finds user details in a Mac OS-X Directory Server LDAP database, and then authenticates the user password against a Mac OS-X Apple Password Server.

Mac OS-X Server includes a facility called Directory Server which provides information about users (amongst other things). Part of the Directory Server facility is an LDAP server that contains the user details. However, the LDAP server never contains any user passwords, it merely contains information about valid methods for authenticating that user. Users that have been configured to use the ‘Password Server’ authentication method can have passwords authenticated by the Apple Password Server facility.

Therefore, AuthBy LDAP_APS can authenticate any user configured into the Apple Directory Server LDAP server, and configured to use the Apple Password Server authentication method.

AuthBy LDAP_APS is a subclass of AuthBy LDAP2. IT queries the Mac OS-X LDAP server for information about a specific user in the same way as AuthBy LDAP2. It uses the user's authAuthority attribute from the LDAP database to determine how to authenticate the password. If the user is configured to be able to use the Apple Password Server (i.e. the authAuthority contains ApplePasswordServer, a user id and a Password Server address) then AuthBy LDAP_APS will authenticate the user's password by contacting (via TCP/IP) the specified Apple Password Server.

At Mac OS-X Server 10.4, Apple Password Server does not support all possible password authentication methods. In particular, it supports Plaintext (via CRAM-MD5), Digest-MD5 and MSCHAPV2. It does not support CHAP or MSCHAPV1. Therefore you can only use AuthBy LDAP_APS to authenticate PAP, MSCHAPV2, TTLS-PAP, TTLS-MSCHAPV2 or PEAP-MSCHAPV2 requests.

AuthBy LDAP_APS is configured in the same was as AuthBy LDAP2, except that you must specify PasswordAttr as authAuthority, since AuthBy LDAP_APS uses that attribute to find and contact the Password Server for that user.

Since standard TCP/IP is used to talk to the LDAP server and the Apple Password Server, it is not necessary to run Radiator and AuthBy LDAP_APS on the Mac OS-X Directory Server host. Radiator could run on a remote Mac, Linux, Windows or other host, different to the Mac OS-X host running the Directory Server and, in the general case, the Apple Password Server could be on a third host.

AuthBy LDAP_APS understands also the same parameters as <AuthBy LDAP2>. For more information, see Section 3.47. <AuthBy LDAP2> on page 226. There is a sample configuration file in `goodies/ldap-aps.cfg` in your Radiator distribution.

3.66.1. PasswordServerAddress

If this optional parameter is set, it forces Radiator to use the specified address as the address of the Apple Password server, instead of deducing it from the users data record. Addresses may be one of the forms:

- 203.63.154.59
- dns/yoke.open.com.au
- ipv4/ 203.63.154.59
- ipv6/2001:720:1500:1::a100

This can be useful with replicated password servers. It is common to set it to localhost:

```
PasswordServerAddress 127.0.0.1
```

3.67. <AuthBy REST>

<AuthBy REST> allows you to handle authentication and accounting requests by using HTTP REST API backends.

The response must decode to a Perl hash data structure which is automatically done for JSON object format responses. If the response is, for example, a JSON array, you can update the decoded response to a Perl hash with MapResponseHook on page 96

See `goodies/rest.cfg` for an example configuration.

This clause supports all the common HTTP client configuration parameters. For more information about the HTTP client configuration parameters, see Section 3.12. HTTP client configuration on page 93.

Radiator's special character formatting is supported for the URL parameter as follows:

- **`%0`** is the current username
- information of the current request is available

Here is an example where the URL must contain username@example.org in an encoded format:

```
<AuthBy REST>
     # Format result for user 'mik/em' is
     # https://api.example.org/mi%2Fkem%40example.org/auth
     URL https://api.example.org/user/%{URIEncodeUTF8:%0}%%40example.org/auth
     FormatURL
```

The current username in **`%0`** is encoded with `URIEncodeUTF8` formatter. **`%%`** is simply formatted to a single **`%`**. For more about Radiator special formatters, see Section 3.3. Special formatters on page 21.

<AuthBy REST> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.67.1. RestAuthRequestDef

This parameter allows you to define parameters for authentication requests sent via HTTP REST API.

You can specify any number of *RestAuthRequestDef* parameters, one for each parameter sent to the server. The general format is:

```
RestAuthRequestDef authparam,attributename[,type[,formatted]]
```

- **authparam** is the REST parameter name
- **attributename** defines the value for authparam. The value may be fetched from the current request, be a value that is subject to special formatting, or a literal value
- **type** consists of the word “literal” or the word “request”. If type is empty or “request”, the value is fetched from the current request. If type is "literal", it is added to the HTTP request as it is.
- **formatted** indicates that the value specified with **attributename** is to be subject to special character processing before being used.

Tip

The **type** and **formatted** fields are optional. If they are not specified, type defaults to "request" and formatted defaults to not enabled.

Example

The following example adds 3 parameters in the REST API request Radiator sends. Parameter "request_type" is set to literal value "authentication". Parameter "server_name" is set to the hostname of server that Radiator runs on. Value for parameter "username" is copied from the "User-Name" attribute of the currently processed request.

```
RestAuthRequestDef request_type, authentication, literal
RestAuthRequestDef server_name, %h, literal, formatted
RestAuthRequestDef username, User-Name
```

3.67.2. RestAcctRequestDef

This parameter allows you to define parameters for accounting requests sent via HTTP REST API.

You can specify any number of *RestAcctRequestDef* parameters, one for each parameter sent to the server. The general format is:

```
RestAcctRequestDef acctparam,attributename[,type[,formatted]]
```

- **acctparam** is the REST parameter name
- **attributename** defines the value for acctparam. The value may be fetched from the current request, be a value that is subject to special formatting, or a literal value
- **type** consists of the word “literal” or the word “request”. If type is empty or “request”, the value is fetched from the current request. If type is "literal", it is added to the HTTP request as it is.
- **formatted** indicates that the value specified with **attributename** is to be subject to special character processing before being used.

Tip

The **type** and **formatted** fields are optional. If they are not specified, type defaults to "request" and formatted defaults to not enabled.

Example

The following example adds 3 parameters in the REST API request Radiator sends. Parameter "request_type" is set to literal value "accounting". Parameter "server_name" is set to the hostname of server that Radiator runs on. Value for parameter "username" is copied from the "User-Name" attribute of the currently processed request. Values from "acct_sess_id" and "acc_status_id" are respectively copied from "Acct-Session-Id" and "Acct-Status-Type" attributes.

```
RestAcctRequestDef request_type, accounting, literal
RestAcctRequestDef server_name, %h, literal, formatted
RestAcctRequestDef username, User-Name
RestAcctRequestDef acct_sess_id, Acct-Session-Id
RestAcctRequestDef acct_status_type, Acct-Status-Type
```

3.67.3. NoReplyReject

This is an optional flag parameter. When this parameter is set, it forces *<AuthBy REST>* to return with result REJECT to trigger an Access-Reject when a REST request times out. This parameter is not set by default.

When *NoReplyReject* is enabled, the reject reason is set to 'REST request timeout'.

3.67.4. ServerChecksPassword

Normally, Radiator fetches the user's credentials, such as password hash, from the server using the *PasswordAttr* or *EncryptedPasswordAttr* parameter and checks the password internally. This optional parameter causes the server to check the password instead. This is useful with servers that implement proprietary encryption algorithms in their passwords, or do not provide access to password attribute.

When *ServerChecksPassword* is specified, Radiator sends the plaintext password with "password" REST API parameter to the server and the password checking is performed by the server only. This is done in addition to any parameters added by *RestAuthRequestDef* on page 280.

Here is an example of using *ServerChecksPassword*:

```
# Send plaintext password to server to check
ServerChecksPassword
```

CAUTION

ServerChecksPassword is compatible with PAP, EAP-TTLS/PAP, and other authentication methods that provide a plain text password. *ServerChecksPassword* does not work with CHAP, MSCHAP, and most EAP methods since these do not provide a password Radiator can send to the API.

3.67.5. RestAuthReplyDef

This parameter allows you to specify how to use REST API reply parameters as check, reply or other items during authentication.

You can specify any number of *RestAuthReplytDef* parameters, one for each parameter sent to the server. The general format is:

```
RestAuthReplyDef replyparam,attributename[,type[,formatted]]
```

- **replyparam** is the REST parameter name
- **attributename** is the name of the attribute that is used as the check, reply or other item. The special attributename 'GENERIC' indicates that the replyparam value is a list of comma separated attribute=value pairs.
- **type** indicates whether replyparam is a check, reply or other item. Possible values are "check" or "reply" for check and reply items. If type is "request" the value is saved in the current request, from where it can be later collected with a special formatting macro like: %{attributename}.
- **formatted** indicates that the value specified with **attributename** is to be subject to special character processing before being used.

Tip

The **type** and **formatted** fields are optional. If they are not specified, type defaults to empty and formatted defaults to not enabled.

Example

The following example uses 3 parameters from REST API requests Radiator receives. Parameters "nas_id" and "client_mac" must match request attributes "NAS-Identifier" and "Calling-Station-Id", respectively. Parameter "sess_timeout" sets the value for reply attribute "Session-Timeout".

```
# How to handle reply for REST authentication request
RestAuthReplyDef nas_id,NAS-Identifier,check
RestAuthReplyDef client_mac,Calling-Station-Id,check
RestAuthReplyDef sess_timeout,Session-Timeout,reply
```

Tip

Password check is defined with configuration parameters *PasswordAttr*, *EncryptedPasswordAttr* and *ServerChecksPassword* on page 281.

3.67.6. EncryptedPasswordAttr

Name of the HTTP reply parameter that contains an encrypted password for the user. If you specify *EncryptedPasswordAttr*, it will be used instead of *PasswordAttr*, and *PasswordAttr* is ignored. You must specify either *PasswordAttr* or *EncryptedPasswordAttr* or *ServerChecksPassword*.

```
# HTTP server sends password hash with pw-hash parameter
EncryptedPassword pw-hash
```

3.67.7. PasswordAttr

Name of the HTTP reply parameter that has the correct plaintext password for the user. If you specify *EncryptedPasswordAttr*, it will be used instead of *PasswordAttr*, and *PasswordAttr* is ignored. You must specify either *PasswordAttr* or *EncryptedPasswordAttr* or *ServerChecksPassword*.

```
# HTTP server sends plaintext password with pw parameter
PasswordAttr pw
```

3.67.8. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *`HandleAcctStatusTypes`* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *`HandleAcctStatusTypes`*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.68. <AuthBy URL>

This clause authenticates using HTTP from any URL. It can use any given CGI or ASP, that validates user name and password. It requires the Digest::MD5 and HTTP::Request and LWP::UserAgent Perl modules in libwww-perl-5.63 or later. They are part of CPAN. For more information, see Section 2.1.2. CPAN on page 3. <AuthBy URL> supports both GET and POST Method for http query strings.

The user name and password being authenticated are passed as URL tags to the program (i.e. it does not use the web server's HTTP authentication). The CGI or ASP can then validate the user name and password and return a string that indicates whether the authentication succeeded or not. Passwords may be sent in the clear, or Unix crypt or MD5 encrypted. Example CGI scripts are available in the goodies directory of your Radiator distribution. See `goodies/README`.

This module was contributed by Mauro Crovato <mauro@crovato.com.ar>. See the example configuration file in `goodies/url.cfg` in your Radiator distribution.

AuthBy URL understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

Tip

The libwww module that AuthBy URL uses can also support HTTPS requests. In order to enable this support, you must build and install either the Crypt::SSLeay or IO::Socket::SSL Perl modules before building the libwww module (see the README.SSL in the libwww distribution for more details). After that you can specify an HTTPS URL with something like:

```
AuthUrl https://www.mysite.com/validate.cgi
```

3.68.1. AuthUrl

This optional parameter specifies the complete URL that will be used to authenticate the user name and password. It is usually set to the URL of a CGI or ASP program on a web server that you control. HTTPS is supported. For more information, see Section 3.68. <AuthBy URL> on page 283.

```
AuthUrl www.mysite.com/validate.cgi
or ....
```

```
AuthUrl https://www.mysite.com/validate.cgi
```

3.68.2. AcctUrl

This optional parameter specifies the complete URL that will be used to save accounting data from Accounting-Request packets. All the attributes in the request will be sent as HTTP tags, using either GET or POST, depending on the setting of UrlMethod.

```
AcctUrl http://www.mysite.com/cgi-bin/save-accounting.cgi
```

3.68.3. UrlMethod

This optional parameter specifies what type of submit method is going to be used to pass user and pass to the URL. Possible values are GET or POST. It is not sensitive to case. The default is GET.

```
UrlMethod POST
```

3.68.4. Debug

This optional flag parameter specifies if any incoming authentication that result in Auth-Accept, will be logged with the Radiator logging system. The default is to not log.

```
Debug 1
```

3.68.5. Timeout

This optional parameter specifies the timeout (in seconds) for the http connection to the web server. The default 5.

```
Timeout 3
```

3.68.6. UserParam

This optional parameter specifies the name of the URL tag variable used to pass the Username being authenticated, to the URL The default is **`user`**.

```
UserParam username
```

3.68.7. PasswordParam

This optional parameter specifies the name of the URL tag variable used to pass the Password being authenticated, to the URL The default is **`password`**.

```
PasswordParam key
```

3.68.8. AuthOKKeyword

This optional parameter specifies the name of the string that has to be found in the response from the web server, to select an Auth-Accept response message The default is **`AuthOK`**.

```
AuthOKKeyword "auth accept"
```

3.68.9. AuthChallengeKeyword

This optional parameter specifies the name of the string that has to be found in the response from the web server, to select an Auth-Challenge response message The default is ‘AuthChallenge’.

3.68.10. BadUserKeyword

This optional parameter specifies the name of the string that has to be found in the response from the web server, to select an Auth-Reject Bad User response message The default is **`BadUser`**.

```
BadUserKeyword "auth reject bad user"
```

3.68.11. BadPasswordKeyword

This optional parameter specifies the name of the string that has to be found in the response from the web server, to select an Auth-Reject Bad Password response message. The default is **`BadPassword`**.

```
BadPasswordKeyword "auth reject bad pass"
```

3.68.12. PasswordEncryption

This optional parameter specifies the type of encryption that is going to be used, to send the PAP password to the URL. The options available are **`Clear`**, **`Crypt`** and **`MD5`** (case insensitive). The default is **`Clear`**.

```
PasswordEncryption Md5
```

3.68.13. ChapChallengeParam

For CHAP authentication, the name of the web parameter to use to send the CHAP challenge. Not used for PAP or other types of authentication. Defaults to chap_challenge.

3.68.14. ChapResponseParam

For CHAP authentication, the name of the web parameter to use to send the CHAP response. Not used for PAP or other types of authentication. Defaults to chap_response.

3.68.15. MSChapChallengeParam

For MSCHAP authentication, the name of the web parameter to use to send the MSCHAP challenge. Not used for PAP or other types of authentication. Defaults to mschap_challenge.

3.68.16. MSChapResponseParam

For MSCHAP authentication, the name of the web parameter to use to send the MSCHAP response. Not used for PAP or other types of authentication. Defaults to mschap_response.

3.68.17. MSChapV2ChallengeParam

For MSCHAPV2 authentication, the name of the web parameter to use to send the MSCHAPV2 challenge. Not used for PAP or other types of authentication. Defaults to mschapv2_challenge.

3.68.18. MSChapV2ResponseParam

For MSCHAPV2 authentication, the name of the web parameter to use to send the MSCHAPV2 response. Not used for PAP or other types of authentication. Defaults to mschapv2_response.

3.68.19. CopyRequestItem

Adds a tagged item to the HTTP request. Format is CopyRequestItem xxx yyy. The text of yyy (which may be contain special characters) will be added to the HTTP request with the tag xxx. In the special case where yyy is not defined, the value of attribute named xxx will be copied from the incoming RADIUS request and added to the HTTP request as the tagged item yyy. All values are HEX encoded before adding to the HTTP request. Multiple CopyRequestItem parameters are permitted, one per line.

```
CopyRequestItem NAS-Port
CopyRequestItem Calling-Station-Id %{OuterRequest:Calling-Station-
Id}
```

3.68.20. CopyReplyItem

Copies an attribute=value pair in a successful HTTP response to the RADIUS reply. Format is CopyReplyItem xxx yyy. If a successful HTTP reply contains a string like "xxx=hexencodedvalue" the value will be copied to the RADIUS reply as attribute yyy=value. Multiple CopyReplyItem parameters are permitted, one per line.

```
CopyReplyItem MS-CHAP2-Success
CopyReplyItem Reply-Message reply_tag
```

3.69. <AuthBy KRB5>

This clause authenticates using the Kerberos 5 authentication system, which is available on most types of operating system. It authenticates from a previously defined Kerberos KDC (Key Distribution Centre). There is a sample configuration file in `goodies/krb5.cfg` in your distribution. AuthBy KRB5 can authenticate PAP and TTLS-PAP. Accounting are ACCEPTed but discarded.

Requires the Authen::Krb5 module version 1.3 or later. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3.

3.69.1. KrbRealm

This optional parameter is the name of the Kerberos realm that all Kerberos users are assumed to be in. Defaults to the default Kerberos realm defined by your Kerberos administrator.

Kerberos principal names are constructed by appending @KrbRealm to the RADIUS user name (after any RADIUS realm has been stripped off. So if a user tries to authenticate as user@realm.com, and *`KrbRealm`* is set to mykrb.com, then the Kerberos principal name that will be authenticated will be 'user@mykerb.com'.

```
# All users are in this realm.
KrbRealm OPEN.COM.AU
```

3.69.2. KrbServerRealm

This optional parameter is the name of the Kerberos realm that the Kerberos server is assumed to be in. Defaults to the KrbRealm value.

3.69.3. KrbKeyTab

This optional parameter provides the path to a Kerberos keytab file. When this option is present, a service ticket will be obtained as part of each Kerberos authentication attempt to guard against Key Distribution Center spoofing. By default, the keytab is examined to locate the key for the service radius/server@realm where server is the fully qualified domain name of the machine running Radiator and realm is the Kerberos realm used during

authentication. The name of the service may be overridden with the KrbService parameter, the fully qualified domain name with the KrbServer parameter and the realm with the KrbRealm parameter.

```
# Enable KDC spoof detection using service ticket
KrbKeyTab /etc/krb5-radius.keytab
```

3.69.4. KrbService

This optional parameter overrides the default value of "radius" for the service name used when locating a key to obtain a service ticket as part of Kerberos Key Distribution Center spoof detection. This parameter has no effect unless the KrbKeyTab parameter is defined. For more information, see Section 3.69.3. KrbKeyTab on page 286. This parameter should be set to the service name of the service key obtained from your Kerberos administrator.

```
# Service name for radius
KrbService radiusproxyauthentication
```

3.69.5. KrbServer

This optional parameter overrides the default value of the fully qualified domain name of the server running radiator when locating a key to obtain a service ticket as part of Kerberos Key Distribution Center spoof detection. This parameter has no effect unless the KrbKeyTab parameter is defined. For more information, see Section 3.69.3. KrbKeyTab on page 286. This parameter should be set to the hostname included in the service key obtained from your Kerberos administrator.

```
# Hostname of the server
KrbServer radius.example.com
```

3.70. <AuthBy MULTICAST>

This clause sends copies of some or all RADIUS requests to all the remote RADIUS servers indicated by the embedded Host clauses. The syntax for this clause is very similar to <AuthBy RADIUS>, but where <AuthBy RADIUS> only sends to the second and subsequent *`Host`* if no reply is heard from previous *`Host`*s, <AuthBy MULTICAST> always sends to all of them. If any *`Host`*s do not reply, the usual retransmissions will occur (the number and timeout is configurable on a per-*`Host`* basis). For more information, see Section 3.42. <AuthBy RADIUS> on page 202.

Tip

It is not normally useful to send multiple copies of authentication requests to multiple hosts. However, it is sometimes desirable to send multiple copies of Accounting-Request. In this case, the IgnoreAuthentication flag will be useful.

This module was contributed by Andrew Ivins and Swiftel.

<AuthBy MULTICAST> understands also the same parameters as <AuthBy RADIUS>. For more information, see Section 3.42. <AuthBy RADIUS> on page 202.

3.70.1. LoopDetection

If this optional parameter is set, Radiator will not forward a request to a Host if the request to be forwarded was originally received from the same address. Defaults to no loop detection.

```
# Enable loop detection
LoopDetection yes
```

3.71. <AuthBy RADSEC>

<AuthBy RADSEC> proxies RADIUS requests to a *<ServerRADSEC>* clause on remote Radiator using the RadSec (RFC 6614) secure reliable RADIUS proxying protocol. It can be used instead of *<AuthBy RADIUS>* when proxying across insecure or unreliable networks such as the internet. For more information about the RadSec protocol, see Section 16. RadSec (RFC 6614) on page 516.

<AuthBy RADSEC> attempts to establish a RadSec connection to the server Hosts when Radiator start up. If the connection subsequently fails or is disconnected, it will attempt to reestablish the connection at *ReconnectTimeout* intervals.

<AuthBy RADSEC> can be configured for multiple target hosts by specifying multiple Host clauses inside the *<AuthBy RADSEC>* clause. Normally when a packet is to forwarded, *<AuthBy RADSEC>* attempts first to send it to the first Host. If no reply is received from that *Host* within *NoreplyTimeout* seconds, it attempts to send the request to the next Host and so on. If no reply is heard from any *Host*, the *NoReplyHook* is called.

<AuthBy RADSEC> supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

Failure algorithm

<AuthBy RADSEC> implements a configurable algorithm to detect failed RadSec hosts, and to temporarily disregard failed hosts. The algorithm uses the *MaxFailedRequests*, *MaxFailedGraceTime*, and *FailureBackoffTime* parameters to customise the operation of the algorithm. For more information, see Section 3.42.15. MaxFailedRequests on page 209, Section 3.42.16. MaxFailedGraceTime on page 209, and Section 3.42.14. FailureBackoffTime on page 209. It also uses *KeepaliveTimeout* and *UseStatusServerForFailureDetect* in order to use only Status-Server requests for failure detection, instead of any request. For more information, see Section 3.42.9. KeepaliveTimeout on page 208 and Section 3.42.13. UseStatusServerForFailureDetect on page 209.

<AuthBy RADSEC> initially assumes that each Host is not failed. After a request is sent to a RadSec server, if no reply is received after the NoreplyTimeout, that request is deemed to have failed for that Host. *<AuthBy RADSEC>* keeps track of how many consecutive requests failed for each Host since the last time a reply was heard from that Host. If more than *MaxFailedRequests* consecutive requests are deemed to have failed within *MaxFailedGraceTime* seconds of that last reply heard from that *Host*, that *Host* is deemed to have failed.

When the *Host* is deemed to be failed, *<AuthBy RADSEC>* does not attempt to send any requests to it until *FailureBackoffTime* seconds have elapsed. In the meantime, *<AuthBy RADSEC>* attempts to connect or reconnect to the host according to *ReconnectTimeout*. It also skips sending requests to that *Host*, and instead attempts to send to the next *Host* in its list of *Host*s (if any).

The default values for these parameters are:

```
MaxFailedRequests 1
MaxFailedGraceTime 0
FailureBackoffTime 0
```

These values mean that by default *<AuthBy RADSEC>* declares the Host failed after a single packet transmission failure, but that it always tries to transmit the next request to the Host. This means that *<AuthBy RADSEC>* always tries to send every request to the first *Host*, and if nothing is heard from that *Host* within *NoreplyTimeout*, it attempts to send to the next *Host*.

Tip

Judicious use of these parameters allows you to implement a RadSec Host fallback policy, where if one RadSec `Host` fails to respond to requests, then it will automatically temporarily fall back to the next RadSec `Host` and so on.

Certificate validation

When a RadSec Server presents a server certificate to an `<AuthBy RADSEC>` Client, the Client performs a number of checks to validate the server certificate. The server certificate is checked for valid start and end dates, and it also checks the chain of validity back to the issuing Certificate Authority, using the root certificates specified in `TLS_CAFile` or `TLS_CAPath`. If `TLS_PolicyOID` parameter is defined, the OIDs must be present in the certificate path. Also a server certificate is accepted only if at least one of the following conditions are true:

- The `Host` name used to connect to the server matches a subjectAltName with type IPADD (IP Address) or DNS (DNS name) in the certificate. Exact or wild card matches are permitted, so a subjectAltName type DNS of '*.xyz.com' matches for any Host in xyz.com.
- If there are no subjectAltNames of type DNS in the certificate, if one of the Subject CN (Common Names) in the certificate matches the Host name used to connect to the RadSec server. Exact or wild card matches are permitted.
- The Subject in the certificate matches the pattern specified by the `TLS_ExpectedPeerName` parameter in this `<AuthBy RADSEC>` clause.

And

- If `TLS_SubjectAltNameURI` parameter is defined in the `<AuthBy RADSEC>` clause, the certificate must contain a subjectAltName of type URI that matches the `TLS_SubjectAltNameURI` regular expression.
- If `TLS_CertificateFingerprint` parameter is defined in the `<AuthBy RADSEC>` clause, the certificate's fingerprint must match at least one of the `TLS_CertificateFingerprint` options.

Note that the default `TLS_ExpectedPeerName` pattern is undefined, which means that by default `<AuthBy RADSEC>` requires that the `Host` name used to connect to the RadSec server matches the subjectAltName or CN in the Server Certificate.

These certificate validation rules are consistent with RFC 2595.

Important

Normally, an `<AuthBy RADSEC>` clause completes as soon as the request has been forwarded to the remote RadSec server. It does not wait for a reply before moving on to other AuthBy clauses, or handling new requests. `<AuthBy RADSEC>` always returns IGNORE for `AuthByPolicy`.

Gossip support in AuthBy RADSEC

The RadSec Hosts can now distribute information about next hop proxy reachability with Gossip. Instead of the current IP address, the configured host name is used as the key when determining if the current report must be processed.

3.71.1. Host

This parameter specifies the host name or address of a RadSec server (the instance of Radiator with a ServerRADSEC clause) that this *<AuthBy RADSEC>* is to connect to. The address can be an IPv4 or IPv6 name or address. Multiple Host lines are supported, which is equivalent to specifying multiple *<Host>* clauses. Special formatting characters are supported.

Here is an example of using *Host*:

```
# Example Host lines for IPv4 and IPv6 addresses
Host 203.63.154.1
Host oscar.open.com.au
Host ipv6:your.ipv6.host.com
Host 2001:db8:1500:1::a100
# IPv6 loopback:
Host ::1
```

3.71.2. Secret

This parameter specifies the shared secret that will be used to encrypt passwords and provide basic RADIUS protocol authentication mechanisms for requests and replies passed over the RadSec connection. It must be the same as the Secret configured into the *<ServerRADSEC>* this clause connects to. The Secret is used to protect passwords even when TLS is not configured for use. If TLS is used, it is not necessary to change it from the default, since the security of TLS does not depend on the shared secret. For compliance with RFC 6614, the default value is **radsec**. There usually is no need to change this.

3.71.3. Port

This optional parameter specifies the symbolic service name or port number of the port to connect to on Host. The default value is **2083**, the official IANA port number for RadSec. Special formatting characters are supported.

3.71.4. LocalAddress and LocalPort

These parameters control the address and optionally the port number used for the client source port, although this is usually not necessary. *LocalPort* is a string, it can be a port number or name. It binds the local port if *LocalAddress* is defined. If *LocalPort* is not specified or if it is set to **0**, a port number is allocated in the usual way.

When SCTP multihoming is supported, multiple comma separated addresses can be configured. All addresses defined with *LocalAddress* must be either IPv4 or IPv6 addresses.

```
LocalAddress 203.63.154.29
LocalPort 12345
```

3.71.5. NoreplyTimeout

If no reply is received to a proxied request within this number of seconds, the request is sent to the next Host in the list (if any). If there are no further Hosts, the *NoReplyHook* is called for this request. The default value is 5 seconds.

3.71.6. KeepaliveTimeout

This optional integer specifies the maximum time in seconds that a RadSec connection can be idle before a Status-Server request is sent to keep the current TCP connection alive. This helps to keep TCP connections open in the face of "smart" firewalls that might try to close idle connections down. The default value is **0** seconds

and keepalives are not used. When *UseStatusServerForFailureDetect* is enabled, *KeepaliveTimeout* together with *MaxFailedRequests* defines the minimum time it takes to notice the next hop has failed.

3.71.7. KeepaliveNoreplyTimeout

This is an optional integer that specifies the waiting time, in seconds, for a Status-Server request. If no reply is received within the time *KeepaliveNoreplyTimeout* defines, the Status-Server request is marked as lost.

If *KeepaliveNoreplyTimeout* is not defined, the waiting time value depends on the AuthBy you are using:

- *<AuthBy RADCSEC>*: *NoreplyTimeout* value is used instead.
- *<AuthBy RADIUS>*: *RetryTimeout* value is used instead.

It is recommended to have a smaller value for *KeepaliveNoreplyTimeout* and a larger value for *NoreplyTimeout* or *RetryTimeout*. The Status-Server responder is always the next hop host and a reply is received quickly. With a short *KeepaliveNoreplyTimeout*, a possible failure situation is discovered quickly and the request is rerouted to another server. The final destination of an Access-Request or an Accounting-Request message may be located many hops away and for this reason a long *NoreplyTimeout* may be needed.

Here is an example of using both *KeepaliveNoreplyTimeout* and *NoreplyTimeout* in *<AuthBy RADSEC>*:

```
<AuthBy RADSEC>
      NoreplyTimeout 10
      KeepaliveNoreplyTimeout 3
</AuthBy>
```

To use this example with *<AuthBy RADIUS>*, you must use *RetryTimeout* instead of *NoreplyTimeout*.

3.71.8. KeepaliveRequestType

This string defines the type of RADIUS request that is sent as a keep-alive request. Any RADIUS request type is allowed. The default value is Status-Server.

Here is an example of using *KeepaliveRequestType* with *AddToKeepaliveRequest*:

```
# Send Access-Request as keepalive probe
KeepaliveRequestType Access-Request
AddToKeepaliveRequest User-Name=mikem,User-Password=fred
```

3.71.9. AddToKeepaliveRequest

This string adds attributes to the keep-alive request before sending it to Host. The value is a comma-separated list of attribute value pairs on one line, exactly as for any reply item. Special formatting characters are allowed, for more information, see Section 3.3. Special formatters on page 21.

Here is an example of using *AddToKeepaliveRequest* with *KeepaliveRequestType*:

```
# Send Access-Request as keepalive probe
KeepaliveRequestType Access-Request
AddToKeepaliveRequest User-Name=mikem,User-Password=fred
```

3.71.10. UseStatusServerForFailureDetect

If this optional flag is enabled, use only Status-Server requests (if any) to determine that a target server is failed when there is no reply. If not enabled (the default) use no-reply to any type of request. This parameter uses *NoreplyTimeout*, *MaxFailedRequests*, *MaxFailedGraceTime*, *FailureBackoffTime* during failure detection.

If you enable this, ensure also *KeepaliveTimeout* is set to a sensible interval to balance between detecting failures early and loading the target server.

3.71.11. StripFromRequest

Strips the named attributes from the request before forwarding it to any Host. The value is a comma-separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.71.12. AddToRequest

Adds attributes to the request before forwarding to any Host. Value is a list of comma separated attribute value pairs all on one line, exactly as for any reply item. *StripFromRequest* removes attributes from the request before AddToRequest adds any to the request. You can use any of the special % formats in the attribute values. There is no default.

```
# Append a Filter-ID and host name
AddToRequest Calling-Station-Id=1,Login-IP-Host=%h
```

3.71.13. AddToRequestIfNotExist

Adds attributes to the request before forwarding to any Host. Unlike *AddToRequest*, an attribute will only be added if it does not already exist in the request. Value is a list of comma separated attribute value pairs. You can use any of the special % formats in the attribute values. There is no default.

```
# Possibly add our default Operator-Name
AddToRequestIfNotExist Operator-Name=1example.com
```

3.71.14. IgnoreReject

This optional parameter causes Radiator to ignore (not send back to the original NAS) any Access-Reject messages received from the remote RadSec server. This is sometimes useful for authenticating from multiple RADIUS servers. However, note that if all the remote RADIUS servers reject the request, then the NAS receives no reply at all.

```
# If we get a reject from the remote, do not send it to the NAS
IgnoreReject
```

3.71.15. IgnoreAccountingResponse

This optional flag causes *<AuthBy RADSEC>* to ignore replies to accounting requests, instead of forwarding them back to the originating host. This can be used in conjunction with the *AccountingHandled* flag in a Handler or Realm to ensure that every proxied accounting request is replied to immediately, and the eventual reply from the remote RADSEC server is dropped. For more information about AccountingHandled flag, see Section 3.31.6. AccountingHandled on page 152.

3.71.16. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *HandleAcctStatusTypes* is specified and an Accounting request has an Acct-Status-Type not mentioned in *HandleAcctStatusTypes*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all Acct-Status-Type values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.71.17. AcctFailedLogFileName

The name of a files used to log failed Accounting-Request messages in the standard radius accounting log format. If no reply is ever received from any of the remote hosts, the accounting message is logged to the named file. For more information about log file format, see Section 9.5. Accounting log file on page 491. If no *AcctFailedLogFileName* is defined, failed accounting messages are not logged. The default is no logging. The file name can include special formatting characters as described in Section 3.3. Special formatters on page 21, which means that, for example, using the %c specifier, you can maintain separate accounting log files for each Client. The *AcctFailedLogFileName* file is always opened, written and closed for each failure, so you can safely rotate it at any time.

The user name that is recorded in the log file is the rewritten user name when *RewriteUsername* is enabled.

Tip

You can change the logging format with *AcctLogFileFormat*.

Tip

You may want to make the filename depend on the date, so you get one missed accounting file per day.

```
# Log all accounting to a single log file in LogDir
AcctFailedLogFileName %L/misseddetails
```

3.71.18. AcctLogFileFormat

This optional parameter is used to alter the format of the failed accounting log file from the standard radius format when *AcctLogFileFormatHook* is not defined. *AcctLogFileFormat* is a string containing special formatting characters. It specifies the format for each line to be printed to the accounting log file. A new line is automatically appended. It is most useful if you use the **%{attribute}** style of formatting characters (to print the value of the attributes in the current packet.

```
AcctLogFileFormat %{Timestamp} %{Acct-Session-Id}\
%{User-Name}
```

3.71.19. AcctLogFileFormatHook

This specifies an optional Perl hook that is used to alter the format of the failed accounting log file from the standard radius format when defined. The hook must return the formatted accounting record. A new line is automatically appended. By default no hook is defined and *AcctLogFileFormat* or the default format is used. The hook parameter is the reference to the current request.

3.71.20. ReplyHook

This optional parameter allows you to define a Perl function that is called after a reply is received from a remote RadSec server and before it is relayed back to the original client. The following arguments are passed in the following order:

- Reference to the reply received from the remote RadSec server is passed as the first argument
- Reference to the reply packet being constructed for return to the NAS is passed as the second argument
- Reference to the original request from the NAS is passed as the third argument
- Reference to the request that was sent to the remote RadSec server is passed as the fourth argument
- Reference to the Radius::AuthRADSEC structure is passed as the fifth argument

The response type can be enforced when needed. For example, when the remote RadSec server has rejected the request, the ReplyHook can do any local processing required for rejects and then change the response type to accept.

```
# Change RadiusResult in the 3rd argument, the original request
ReplyHook sub { ${$_[2]}->{RadiusResult} = $main::ACCEPT; }
```

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code will be reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks (with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore do not use trailing comments in your hook.

`ReplyHook` Can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things.

```
# Fake a new attribute into the reply going back to the client
ReplyHook sub { ${$_[1]}->add_attr('test-attr', \
      'test-value');}
```

3.71.21. NoReplyHook

This optional parameter allows you to define a Perl function that is called if no reply is received from any RadSec server. A reference to the original request received from the NAS is passed as the first argument. A reference to the request that was forwarded to the remote RADIUS server is passed as the second argument. A reference to the reply packet being constructed for return to the NAS is passed as the third argument (note that the normal behaviour in case of no reply, is for no reply to be sent to the NAS).

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks (with trailing backslashes (\)) are parsed by Radiator into one long line. Therefore do not use trailing comments in your hook.

`NoReplyHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. In particular, you can forward the request to another AuthBy RADSEC clause, allowing you to implement automatic failover of RadSec hosts.

```
# Call another AuthBy RADSEC if this one fails to respond
NoReplyHook sub { Radius::AuthGeneric::find('RADSEC2')\
      ->handle_request(${$_[0]}, ${$_[2]});}
```

3.71.22. NoReplyReject

This is an optional flag parameter. When this parameter is set, it forces *<AuthBy RADSEC>* to return with result REJECT to trigger an Access-Reject when a proxied request times out. This parameter is not set by default.

When *NoReplyReject* is enabled, the reject reason is set to 'Upstream timeout'.

Tip

NoReplyReject allows rejecting timed out requests without hooks such as *NoReplyHook*. For more information, see Section 3.71.21. NoReplyHook on page 294

3.71.23. Asynchronous

The default behaviour for *<AuthBy RADSEC>* is to return IGNORE as soon as the request has been forwarded to the remote RadSec server. It does not wait for a reply before moving on to other AuthBy classes or handling new requests. You can change this behaviour with the *Asynchronous* flag.

When you enable the *Asynchronous* flag, *Handler* continues to evaluate its AuthBy policy after a reply or timeout from the remote proxy. Other requests are processed while the reply is pending.

For more information about how proxied requests are waited for by *<AuthBy RADSEC>*, see Section 3.71. <AuthBy RADSEC> on page 288.

Here is an example of using *Asynchronous*:

```
# Auth to server1 and continue to AuthBy FILE if server 1 accepts.
# Process other requests while the reply from server 1 is pending.
<Handler>
      AuthByPolicy ContinueWhileAccept
      <AuthBy RADSEC>
            # Evaluate the policy when we get a reply or a timeout
            Asynchronous
            Host server1
            # Other parameters
      </AuthBy>
      <AuthBy FILE>
            Filename %D/users
      </AuthBy>
</Handler>
```

3.71.24. ForwardHook

This is a Perl hook that is called once for each request before the request is forwarded to a remote RADIUS or RadSec server. It allows you to modify the forwarded request without changing the current one. *ForwardHook* receives the following arguments:

- Current request
- Forwarded request

Here is an example of using *ForwardHook*:

```
ForwardHook sub { my $p = $_[0]; my $fp = $_[1]; \
                  $fp->add_attr('OSC-AVPAIR', 'Added by ForwardHook'); }
```

3.71.25. NoForwardAuthentication

This parameter stops *<AuthBy RADSEC>* forwarding Authentication-Requests. They are ACCEPTED, but no further action is taken with them. This is different in meaning to *IgnoreAuthentication*, which IGNOREs them.

```
# Just ACCEPT Authentication-Requests, do not forward them
NoForwardAuthentication
```

3.71.26. NoForwardAccounting

This parameter stops *<AuthBy RADSEC>* forwarding Accounting-Requests. They are ACCEPTED, but no further action is taken with them. This is different in meaning to *IgnoreAccounting*, which IGNOREs them.

```
# Just ACCEPT Accounting-Requests, do not forward them
NoForwardAccounting
```

3.71.27. AllowInRequest

This optional parameter specifies a list of attribute names that are permitted in forwarded requests. Attributes whose names do not appear in this list are stripped from the request before forwarding.

```
# Strip everything except username and password
AllowInRequest User-Name,User-Password
```

3.71.28. MaxBufferSize

This optional advanced parameter specifies the maximum number of octets that are output and input buffered by Radiator's Stream modules. This advanced parameter usually does not need adjusting.

3.71.29. DisconnectTraceLevel

This optional parameter specifies log trace level for peer initiated disconnects. The default value is error level **0**. When connections are known to be short-lived, a non-default value may be useful. This parameter is available for all Stream based modules, such as *<ServerDIAMETER>* and *<AuthBy RADSEC>*.

```
# Debug logging is enough for peer disconnects
DisconnectTraceLevel 4
```

3.71.30. ReconnectTimeout

This optional parameter specifies the number of seconds to wait before attempting to reconnect a failed, dropped or disconnected connection. It also specifies the timeout for the initial connect.

3.71.31. StreamStateChangeHook

This optional parameter allows you to define a Perl function that will be called when a Stream connection state to a peer changes. RadSec, Diameter, TACACS+, HTTP and SIGTRAN clients and servers use Stream connections in Radiator. This is a low layer hook that runs before, for example, RadSec TLS handshake or Diameter Capabilities Exchange messages are exchanged. The following arguments are passed in the following order:

- Reference to this `Radius::Stream` derived type. Examples of derived types are `Radius::RadsecHost` and `Radius::DiameterConnection`.

- New stream state. One of integer constant values:
  - `$Radius::Stream::STREAM_STATE::CONNECTED`
  - `$Radius::Stream::STREAM_STATE::DISCONNECTED`

`StreamStateChangeHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. IP address and other specific information is not passed to the hook. Their format and type depends on the derived type. For example, SCTP streams may have multiple source and destination addresses. The following example shows how to log information from the hook no matter what the type of the stream object is.

```
StreamStateChangeHook sub { \
    my ($self, $new_state) = @_; \
    my $state = ($new_state == $Radius::Stream::STREAM_STATE::CONNECTED) ? \
                 'connected' : 'disconnected'; \
    main::log($main::LOG_INFO, "StreamStateChangeHook: State change to $state"); \
    return; }
```

3.71.32. FailureBackoffTime

This is an integer that defines the number of seconds for how long *<AuthBy RADSEC>* does not attempt to send any requests after host is deemed to be failed. During that time, *<AuthBy RADSEC>* attempts to connect to another host according to host configuration and to send the requests to the next host in the list of hosts, if there is any.

3.71.33. Protocol

This optional parameter specifies which Stream protocol is used to carry RadSec. Options are **`tcp`** for TCP/IP or **`sctp`** for SCTP (Stream Control Transmission Protocol). The default value is **`tcp`**. Not all hosts are able to support **`sctp`**, consult your vendor. The protocol setting must be the same in each RadSec server and client.

```
Protocol sctp
```

Tip

On modern Linux hosts, SCTP support is in a loadable module, and can be enabled with:

```
modprobe sctp
```

3.71.34. ConnectOnDemand

This optional flag parameter tells Radiator not to connect to the server as soon as possible, but to wait until a message must be sent to that server. After the connection has been established and message has been delivered, the connection remains as long as possible. If the connection is lost for any reason, it is only re-established when and if there is another message to be sent.

3.71.35. Gossip

The *`Gossip`* flag parameter enables this AuthBy to send and listen for notifications (aka. gossip) to/from other Radiator servers when any remote Radius server fails to reply. This is disabled by default.

See Section 3.134. <GossipRedis> on page 439 for more about the Gossip framework and `goodies/farmsize.cfg` for a configuration sample.

```
# Use the configured Gossip implementation for notifications
Gossip
```

3.71.36. NoKeepaliveTimeoutForChildInstances

If this optional flag is enabled, only the first instance of this server farm will send Status-Server requests. Recommended when Gossip is used to distribute reachability information. This is not set by default.

3.71.37. GossipNoReply

If the *GossipNoReply* flag parameter is set, then a notification is sent when a remote Radius server fails to reply. Radiator server also increases a counter for failed requests when a notification is received. This is enabled by default.

3.71.38. GossipHostDown

If the *GossipHostDown* flag parameter is set, then a notification is sent when a remote Radius server is marked down. Radiator server also marks the remote Radius server down when a notification is received. This is enabled by default.

3.71.39. GossipHostUp

If the *GossipHostUp* flag parameter is set, then a notification will be send when a remote Radius server is marked up and alive again. Radiator server also marks the remote Radius server healthy when a notification is received. This is enabled by default.

3.71.40. ProxyAlgorithm

You can specify one or more hosts to connect to. The default proxy algorithm is to attempt to connect to the first listed host. If it does not reply within *NoreplyTimeout* seconds, Radiator logs a fail-over and attempts to connect the next listed host.

You can change this default proxy algorithm by configuring *ProxyAlgorithm*. Currently, it supports the following values:

- **FailOver**
- **HashBalance**
- **LoadBalance**
- **VolumeBalance**
- **RoundRobin**
- **EAPBalance** - not recommended

ProxyAlgorithm is not set by default, which means that **FailOver** method is used. Format specifiers, such as *%{GlobalVar:name}*, are evaluated when the configuration is loaded.

Consider migrating current configurations from **EAPBalance**. **HashBalance** has better interoperability with different RADIUS implementations.

Note

LoadBalance and VolumeBalance use BogoMips parameter to determine which target Host to use. For more information, see Section 3.43.1. BogoMips on page 221.

3.71.41. HashAttributes

The *HashBalance* proxy algorithm uses *HashAttributes* configuration parameter to specify which attributes in the incoming request are used to select the target host. The default value is **%{Request:Calling-Station-Id}:%n**.

Here is an example of using *HashAttributes*:

```
HashAttributes %{Request:NAS-IP-Address}
```

3.72. <Host xxxxxx> within <AuthBy RADSEC>

This clause can be used to specify the name and details of remote RadSec servers inside *<AuthBy RADSEC>* and its subtypes. The *<Host xxxxxx>* clause further allows you to customise details for individual hosts. *<AuthBy RADSEC>* permits one or more Host clauses.

In the *<Host xxxxxx>* clause header, the *xxxxxx* is the Host name or IP address of the remote RadSec host to proxy to. The Host name can contain special formatting characters, which are resolved at startup. Here is an example of using Host clause within *<AuthBy RADSEC>*:

```
<AuthBy RADSEC>
      <Host server1.test.com>
            Secret xyzzy
            UseTLS
            TLS_CAFile ....
            ......
      </Host>
      <Host server2.test.com>
            Secret xyzzy
            Port 1645
            UseTLS 0
      </Host>
</AuthBy>
```

The following parameters can be used within a Host clause. They have the same meaning and default values as the parameter of the same name in the enclosing *<AuthBy RADSEC>*:

- Secret on page 290
- Port on page 290
- NoreplyTimeout on page 290
- MaxFailedRequests on page 209
- MaxFailedGraceTime on page 209
- FailureBackoffTime on page 297
- ReconnectTimeout on page 96
- ConnectOnDemand on page 96
- Protocol on page 297
- MaxBufferSize on page 97
- UseTLS on page 92
- TLS_CAFile on page 81

- TLS_CAPath on page 81
- TLS_CertificateFile on page 82
- TLS_CertificateChainFile on page 82
- TLS_CertificateType on page 82
- TLS_PrivateKeyFile on page 82
- TLS_PrivateKeyPassword on page 83
- TLS_RandomFile on page 84
- TLS_DHFile on page 84
- TLS_ECDH_Curve on page 84
- TLS_CRLCheck on page 85
- TLS_CRLCheckAll on page 85
- TLS_CRLFile on page 85
- TLS_SessionResumption on page 92
- TLS_SessionResumptionLimit on page 92
- TLS_ExpectedPeerName on page 86
- TLS_SubjectAltNameURI on page 86
- TLS_SNIHostname on page 87
- TLS_CertificateFingerprint on page 87
- TLS_PolicyOID on page 86
- TLS_SRVName on page 89

3.73. <AuthBy SASLAUTHD>

This clause authenticates against a saslauthd server running on the same host as Radiator. Saslauthd is a Unix authentication server program, part of the Cyrus SASL suite. It can be configured to authenticate from a variety of sources, including PAM, Kerberos, DCE, shadow password files, IMAP, LDAP, SIA or a special SASL user password file. It is part of the Cyrus SASL suite.

AuthBy SASLAUTHD connects to the saslauthd server over a UNIX domain socket. It sends the username, plaintext password, realm and a service name to saslauthd. Saslauthd then authenticates the user using whatever method it has been configured to use and then sends the response back to AuthBy SASLAUTHD.

Requires that saslauthd be installed, configured and running on the Radiator host.

Tip

You can run saslauthd with the -d flag to get a fairly detailed log of what it is doing printed to stdout. This can be helpful determining why authentication is failing.

CAUTION

AuthBy SASLAUTHD is synchronous: it waits until saslauthd responds to an authentication request before sending a RADIUS response to the NAS. Some authentication methods implemented by saslauthd

are slow. For example PAM will wait several seconds before responding if the password is incorrect (this part of the normal behaviour of PAM; it discourages brute force cracking of passwords).

3.73.1. SocketPath

This optional parameter specifies the name of the UNIX domain socket to use to connect to the saslauthd server. Defaults to `/var/lib/sasl2/mux`.

```
# Connect to a non-standard socket
SocketPath /var/state/saslauthd
```

3.73.2. Service

This optional parameter specifies the service name that will be passed to saslauthd in each authentication request. The service name is used by some types of saslauthd authentication methods, for example if saslauthd is using PAM, then this specifies the PAM service name to use. Defaults to **`login`**.

```
# Use the PAM system-auth method
Service system-auth
```

3.74. <AuthBy NTLM>

This clause authenticates against a Windows Domain Controller, using the `ntlm_auth` program, which is part of the Samba suite. For more information, see Samba website [https://www.samba.org/]. `ntlm_auth` runs on all Unix and Linux platforms, and therefore <AuthBy NTLM> can be used on Unix or Linux to authenticate to a Windows Domain Controller.

<AuthBy NTLM> supports PAP, MSCHAP, MSCHAPV2 and EAP-MSCHAPV2 authentication. CHAP is not supported due to limitations in the Windows support for CHAP authentication.

<AuthBy NTLM> requires that `ntlm_auth` and `winbindd`, both part of Samba, are installed and configured correctly. See `goodies/smb.conf.winbindd` for sample configuration and installation hints.

<AuthBy NTLM> runs the Samba utility ntlm_auth as a child process in order to authenticate requests. It keeps ntlm_auth running between requests and passes it authentication information on stdin, and gets back the authentication results from stdout.

Because AuthBy NTLM requires that ntlm_auth be properly installed and configured with `winbindd`, it is vitally important that you confirm that ntlm_auth is working properly before trying to use AuthBy NTLM. You can test ntlm_auth like this:

```
ntlm_auth --username=yourusername --domain=yourdomain --password=yourpassword
```

if that does not work for a valid user name and password, there is no way that AuthBy NTLM will work. Make sure ntlm_auth works first!

CAUTION

AuthBy NTLM blocks while waiting for the result output of ntlm_auth.

Tip

If you are running Radiator on Windows, and wish to authenticate to Windows Active Directory or to a Windows Domain Controller. For more information, see Section 3.60. <AuthBy LSA> on page 260.

Tip

Depending on the ownerships and permissions of certain samba files, Radiator may need to run with root permission.

3.74.1. Domain

This optional parameter specifies which Windows domain will be used to authenticate passwords, regardless of whether the user supplies a domain when they log in. It can be the name of any valid domain in your network. Special characters are permitted. The default is to use the domain configured into winbindd.

3.74.2. DefaultDomain

This optional parameter specifies the Windows Domain to use if the user does not specify a domain in their user name. Special characters are supported. Can be an Active directory domain or a Windows NT domain controller domain name. Empty string (the default) means the domain configured into winbindd.

```
DefaultDomain OPEN
```

3.74.3. NtlmAuthProg

This optional parameter specifies the path name and arguments for the `ntlm_auth` program. The default value is **`/usr/bin/ntlm_auth --helper-protocol=ntlm-server-1`**. The **`--helper-protocol=ntlm-server-1`** is an important part of the arguments to `ntlm_auth` and it is required for the correct interaction between *<AuthBy NTLM>* and `ntlm_auth`. If it is not included, *<AuthBy NTLM>* does not work correctly.

Here is an example how to require the authenticated user to belong to a certain group:

```
NtlmAuthProg /usr/bin/ntlm_auth --helper-protocol=ntlm-server-1 \
             --require-membership-of=MyGroupName
```

Here is an example how to specify that the NTLM authentication request appear to come from a workstation with a specified name. This can be used to restrict authentication for certain users by setting workstation requirements in their Windows user configuration.

```
NtlmAuthProg /usr/bin/ntlm_auth --helper-protocol=ntlm-server-1 \
             --workstation=MyWorkstationName
```

Note

Use `--allow-mschapv2` flag when *`LMCombatibilityLevel`* registry key in Windows configuration is set to value **5** to disable older authentication methods. In this case, MSCHAP and MSCHAP-V2, and EAP-MSCHAP-V2 authentications fail while PAP authentication works with *<AuthBy NTLM>* on Radiator. The availability of `--allow-mschapv2` flag depends on the ntlm_auth version.

3.74.4. UsernameMatchesWithoutRealm

Forces AuthBy NTLM to strip any realm from the user name before authenticating to the domain controller.

3.74.5. NtlmRewriteHook

This optional parameter allows you to define a Perl function to rewrite the username that is passed to `ntlm_auth`. Username passed to `ntlm_auth` is changed to whatever is returned by this function. The username in request is not changed. This may be needed, for example, with Wi-Fi roaming where roaming username can not be directly used with Windows authentication because of local naming conflicts with roaming requirements.

The following parameters are passed to *`NtlmRewriteHook`*:

- *`$_[0]: $p`*, the current `Radius::Radius` request object
- *`$_[1]: $user`*, the current username to pass to `ntlm_auth`

Here are some examples:

```
# We use file instead of inline code
NtlmRewriteHook file:"%D/ntlm-rewrite-hook.pl"
```

```
# Use inline code to change our global roaming realm to windows domain
NtlmRewriteHook sub { my ($user) = $_[1]; \
    $user =~ s/example\.com\z/org.local/; \
    return $user; }
```

3.74.6. UsernameFormat

Controls how the user name that will be sent to NTLM will be derived from User-Name in the incoming request. Special characters are permitted. %0 is replaced with the user name portion of ‘domain\username’. Defaults to ‘%0’.

3.74.7. DomainFormat

Controls how the domain name that will be sent to NTLM will be derived from User-Name in the incoming request. Special characters are permitted. %0 is replaced with the domain portion of ‘domain\username’. Defaults to ‘%0’.

3.75. <AuthBy DNSROAM>

This clause proxies RADIUS requests to remote RADIUS and/or RadSec servers based on the Realm in the User-Name. The appropriate server to send to and the protocol to use is discovered through DNS lookups configured through the Resolver clause. For more information, see Section 3.118. <Resolver> on page 396. You must include a <Resolver> clause in your configuration if you intend to use <AuthBy DNSROAM>.

AuthBy DNSROAM is intended to make it easy to set up a secure, reliable, low maintenance RADIUS/RadSec federation. A RADIUS federation (sometimes called a RADIUS mesh) is a set of RADIUS servers, operated by a set of independent but cooperating organisations. The goal is to permit users who belong to one organisation to be able to use RADIUS-controlled resources at another organisation. A typical example is for a group of Universities to cooperate to permit a user from one University to connect to the wireless network at any other University in the group using their home user name and password. Radiator also permits RADIUS requests to be sent to another Radiator server through RadSec. RadSec provides secure, encrypted, reliable transport of RADIUS requests, with optional mutual authentication of RadSec client and server. For more information about RadSec protocol, see RadSec white paper [https://files.radiatorsoftware.com/radiator/whitepapers/radsec-whitepaper.pdf].

Using AuthBy DNSROAM and DNS to hold information about the target server for each Realm permits convenient and scalable administration of the routing topology within a RADIUS/RadSec mesh.

DNSROAM cooperates well with existing RADIUS infrastructure, and can interoperate with other RADIUS servers and clients, as well as other RadSec servers and clients. It supports hardwired preconfigured RADIUS and RadSec routes as well as DNS discovered routes. It can provide a default fallback, so that Realms that are neither discovered nor hardwired can be forwarded to some catchall server (or dropped). It supports forwarding to IPv4 and/or IPv6 addresses. RadSec can use TCP or SCTP protocol for transport over IPv4 or IPv6. DNSROAM supports discovering RADIUS proxy servers as well as RadSec proxy servers.

The <AuthBy DNSROAM> clause can contain one or more <Route> subclauses which specify hardwired target servers for certain Realms or a DEFAULT fallback server. <Route> subclauses can specify RADIUS or RadSec target servers.

A sample configuration file showing how to use <Resolver>, <AuthBy DNSROAM> and <Route> clauses together can be found in `goodies/dnsroam.cfg` in your Radiator distribution.

AuthBy DNSROAM uses the following algorithm when it receives a RADIUS request for handling:

1. Extract the Realm from the User-Name in the RADIUS request. (The user name and/or realm can be configured to be rewritten by patterns in the enclosing Handler or Client clause).
2. Look for a preconfigured target server <Route> subclause for that Realm.
3. If no preconfigured target server <Route> subclause is found for that Realm, try to discover a target server name or address using DNS (more below on exactly how this is done).
4. If no target server is preconfigured or discovered, try to find a DEFAULT preconfigured target server <Route> subclause.
5. If there is still no target server found, redespatch to the Handler system for handling if *`RedespatchIfNotarget`* configuration parameter is set.
6. If the request is not redespatched, log and drop the request.
7. If the target server is a RadSec server (Protocol=radsec) establish a RadSec connection to the target server (using a private AuthBy RADSEC clause), and if so configured, set up TLS tunnel and perform mutual authentication based on PKI certificates.
8. If the target server is a RADIUS server (Protocol=radius) forward the request using RADIUS protocol over UDP (using a private AuthBy RADIUS clause).
9. When a reply is received from the target server, send the reply back to wherever the request originally came from (there may be multiple proxying hops until the request reaches the home RADIUS server for that Realm).

AuthBy DNSROAM creates private AuthBy RADIUS and/or AuthBy RADSEC clauses to implement each discovered and hardwired RADIUS and RadSec Route. The default values for the parameters for these private clauses are obtained from the enclosing <Route> and/or <AuthBy DNSROAM> clauses, and can be overridden by <Route> clause parameters and parameters discovered from DNS by the <Resolver> clause. Both AuthBy RADIUS and AuthBy RADSEC require a shared secret. The default for AuthBy DNSROAM is **`radsec`** as required by the RadSec RFC. For more information about Route and Resolver, see Section 3.76. <Route> on page 307 and Section 3.118. <Resolver> on page 396.

AuthBy DNSROAM supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

AuthBy DNSROAM understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.75.1. RewriteTargetRealm

This optional parameter can be used to specify one or more rewriting rules which will be used to rewrite the Realm name used by Resolver to discover the appropriate target server.

```
# Find the target server for users@uninett.no by looking up
# as if it were @no.eduroam.org
RewriteTargetRealm s/uninett.no/no.eduroam.org/
```

You can also set up a default realm that will be used if there is no realm in the user name after rewriting by having the last RewriteTargetRealm like this:

```
RewriteTargetRealm s/^$/default.realm.com/
```

3.75.2. RedespatchIfNoTarget

For a given request, if *`Resolver`* does not find a target and there is no explicit *`Route`*, and no *`Route`* with *`Realm DEFAULT`* and this flag is set, the request will be redespatched to the Handler system for handling. This allows for a flexible fallback in the case where DNSROAM cannot find how to route a request. The redespatched request will have the attribute *`OSC-Environment-Identifier`* set to the AuthBy DNSROAM's *`Identifier`* value or **`DNSROAM`** if the *`Identifier`* is not set.

To create a fallback to multiple target hosts, configure *`AuthBy DNSROAM`* without default realm *`Route`* and with *`RedespatchIfNoTarget`* set. Then create a `<Handler ExistsInRequest=OSC-Environment-Identifier>` to match the redespatched requests. This Handler can have, for example, *`AuthBy RADSEC`* clause with two hosts which are used if DNSROAM cannot route the request.

3.75.3. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *`HandleAcctStatusTypes`* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *`HandleAcctStatusTypes`*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.75.4. <Route> parameters

Any of the following *`<Route>`* parameters may be placed in *`<AuthBy DNSROAM>`* as defaults for the enclosed *`<Route>`* clauses:

- Address on page 308
- Transport on page 308
- Protocol on page 308
- Port on page 308
- Secret on page 308
- UseTLS and TLS_Protocols on page 308

3.75.5. <AuthBy RADIUS> parameters

Any of the following *<AuthBy RADIUS>* parameters may be placed in *<AuthBy DNSROAM>* as defaults for any RADIUS proxy. These defaults can be overridden for individual Routes in the *<Route>* clause, and can be overridden by automatically discovered Routes. For more information, see Section 3.42. <AuthBy RADIUS> on page 202.

- Section 3.42.19. StripFromRequest on page 210
- Section 3.42.20. AddToRequest on page 210
- Section 3.42.21. AddToRequestIfNotExist on page 211
- Section 3.42.27. ReplyHook on page 212
- Section 3.42.28. NoReplyHook on page 213
- Section 3.42.23. NoForwardAuthentication on page 211
- Section 3.42.24. NoForwardAccounting on page 211
- Section 3.42.50. AllowInRequest on page 219
- Section 3.42.4. AuthPort on page 206
- Section 3.42.5. AcctPort on page 207
- Section 3.42.3. Secret on page 206
- Section 3.42.7. Retries on page 207
- Section 3.42.8. RetryTimeout on page 207
- Section 3.42.36. UseOldAscendPasswords on page 216
- Section 3.42.37. ServerHasBrokenPortNumbers on page 216
- Section 3.42.38. ServerHasBrokenAddresses on page 216
- Section 3.42.41. IgnoreReplySignature on page 217
- Section 3.42.35. UseExtendedIds on page 215
- Section 3.42.15. MaxFailedRequests on page 209
- Section 3.42.16. MaxFailedGraceTime on page 209
- Section 3.42.14. FailureBackoffTime on page 209

3.75.6. <AuthBy RADSEC> parameters

Any of the following *<AuthBy RADSEC>* parameters may be placed in *<AuthBy DNSROAM>* as defaults for any RADSEC proxy. These defaults can be overridden for individual Routes in the *<Route>* clause, and can be overridden by automatically discovered Routes. For more information, see Section 3.71. <AuthBy RADSEC> on page 288.

- Section 3.71.3. Port on page 290
- Section 3.71.2. Secret on page 290
- Section 3.71.11. StripFromRequest on page 292
- Section 3.71.12. AddToRequest on page 292
- Section 3.42.21. AddToRequestIfNotExist on page 211
- Section 3.71.20. ReplyHook on page 294

- Section 3.71.21. NoReplyHook on page 294
- Section 3.71.25. NoForwardAuthentication on page 296
- Section 3.71.26. NoForwardAccounting on page 296
- Section 3.71.27. AllowInRequest on page 296
- Section 3.71.5. NoreplyTimeout on page 290
- Section 3.71.14. IgnoreReject on page 292
- Section 3.71.15. IgnoreAccountingResponse on page 292
- Section 3.71.32. FailureBackoffTime on page 297
- Section 3.12.21. MaxBufferSize on page 97
- Section 3.12.17. ReconnectTimeout on page 96
- Section 3.12.18. ConnectOnDemand on page 96
- Section 3.11.42. UseTLS on page 92
- Section 3.11.2. TLS_CAFile on page 81
- Section 3.11.3. TLS_CAPath on page 81
- Section 3.11.4. TLS_CertificateFile on page 82
- Section 3.11.5. TLS_CertificateChainFile on page 82
- Section 3.11.6. TLS_CertificateType on page 82
- Section 3.11.7. TLS_PrivateKeyFile on page 82
- Section 3.11.8. TLS_PrivateKeyPassword on page 83
- Section 3.11.14. TLS_RandomFile on page 84
- Section 3.11.15. TLS_DHFile on page 84
- Section 3.11.17. TLS_CRLCheck on page 85
- Section 3.11.20. TLS_CRLFile on page 85
- Section 3.11.44. TLS_SessionResumption on page 92
- Section 3.11.45. TLS_SessionResumptionLimit on page 92
- Section 3.11.22. TLS_ExpectedPeerName on page 86
- Section 3.11.23. TLS_SubjectAltNameURI on page 86
- Section 3.11.26. TLS_CertificateFingerprint on page 87

3.76. <Route>

Route clauses can be used inside an <AuthBy DNSROAM> clause to explicitly specify target servers and protocols for certain Realms, or a DEFAULT fallback server.

The Route clause understands the following parameters. In general, all the Route parameters default to the parameter of the same name in the enclosing AuthBy DNSROAM.

3.76.1. Realm

Specifies the Realm that this Route will apply to. All requests with a User-Name whose Realm component (after applying any RewriteTargetRealm rules) match this realm will by processed using this Route. If the Realm is

'DEFAULT' then this Route will be used to process requests for which no explicit Route exists, and no route could be discovered through DNS and the <Resolver> clause.

3.76.2. Address

Specifies the name or address of the target server to be used to process requests for this Route. Defaults to 'localhost'. There can be only one Address for a Route. For more information about handling a default route with more than one address, see Section 3.75.2. RedespatchIfNoTarget on page 305.

3.76.3. Transport

Specifies the transport to be used to contact the target server. Can be 'sctp', 'tcp', or 'udp'. Defaults to 'tcp'.

3.76.4. Protocol

Specifies the protocol to be used to contact the target server. Can be 'radsec' or 'radius'. Defaults to 'radsec'.

3.76.5. Port

Specifies the port number to be used to contact the target server. Defaults to 2083, the standard port number for RadSec protocol.

3.76.6. UseTLS and TLS_Protocols

Specifies whether TLS is to be used to encrypt the connection to the target server. Valid only for `Protocol=radsec`. Although it is possible to not use TLS for a RadSec connection, it is recommended that RadSec connections always be configured to use TLS. Defaults to true.

`TLS_Protocols` sets the allowed TLS versions. For more information, see Section 3.11.1. TLS_Protocols on page 80

3.76.7. Secret

Specifies the shared secret to be used with the target server. Defaults to **`radsec`** to comply with RadSec. You may want to change this if the enclosing Route is used for RADIUS forwarding. For more information, see Section 3.42.3. Secret on page 206 and Section 3.71.2. Secret on page 290.

3.76.8. <AuthBy RADIUS> parameters

Any of the following `<AuthBy RADIUS>` parameters may be placed in `<Route>` as defaults for a RADIUS proxy. For more information, see Section 3.42. <AuthBy RADIUS> on page 202.

- Section 3.42.19. StripFromRequest on page 210
- Section 3.42.20. AddToRequest on page 210
- Section 3.42.27. ReplyHook on page 212
- Section 3.42.28. NoReplyHook on page 213
- Section 3.42.23. NoForwardAuthentication on page 211
- Section 3.42.24. NoForwardAccounting on page 211
- Section 3.42.50. AllowInRequest on page 219
- Section 3.42.4. AuthPort on page 206
- Section 3.42.5. AcctPort on page 207
- Section 3.42.3. Secret on page 206

- Section 3.42.7. Retries on page 207
- Section 3.42.8. RetryTimeout on page 207
- Section 3.42.36. UseOldAscendPasswords on page 216
- Section 3.42.37. ServerHasBrokenPortNumbers on page 216
- Section 3.42.38. ServerHasBrokenAddresses on page 216
- Section 3.42.41. IgnoreReplySignature on page 217
- Section 3.42.35. UseExtendedIds on page 215
- Section 3.42.15. MaxFailedRequests on page 209
- Section 3.42.15. MaxFailedRequests on page 209
- Section 3.42.16. MaxFailedGraceTime on page 209
- Section 3.42.14. FailureBackoffTime on page 209

```
<AuthBy DNSROAM>
      # Defaults for all enclosed Routes:
      Port 1645
      Transport udp
      Protocol radius
      Secret mysecret
      <Route>
            Realm realm3.open.com.au
            Address oscar.open.com.au
            # Override parameters for AuthBy RADIUS
            Secret xyzzy
      </Route>
      ...
</AuthBy>
```

3.76.9. <AuthBy RADSEC> parameters

Any of the following *<AuthBy RADSEC>* parameters may be placed in *<Route>* as defaults for a RADSEC proxy. For more information, see Section 3.71. <AuthBy RADSEC> on page 288.

- Section 3.71.3. Port on page 290
- Section 3.71.2. Secret on page 290
- Section 3.71.11. StripFromRequest on page 292
- Section 3.71.12. AddToRequest on page 292
- Section 3.71.20. ReplyHook on page 294
- Section 3.71.21. NoReplyHook on page 294
- Section 3.71.25. NoForwardAuthentication on page 296
- Section 3.71.26. NoForwardAccounting on page 296
- Section 3.71.27. AllowInRequest on page 296
- Section 3.71.5. NoreplyTimeout on page 290
- Section 3.71.14. IgnoreReject on page 292

- Section 3.71.15. IgnoreAccountingResponse on page 292
- Section 3.71.32. FailureBackoffTime on page 297
- Section 3.12.21. MaxBufferSize on page 97
- Section 3.12.17. ReconnectTimeout on page 96t
- Section 3.12.18. ConnectOnDemand on page 96
- Section 3.11.42. UseTLS on page 92
- Section 3.11.2. TLS_CAFile on page 81
- Section 3.11.3. TLS_CAPath on page 81
- Section 3.11.4. TLS_CertificateFile on page 82
- Section 3.11.5. TLS_CertificateChainFile on page 82
- Section 3.11.6. TLS_CertificateType on page 82
- Section 3.11.7. TLS_PrivateKeyFile on page 82
- Section 3.11.8. TLS_PrivateKeyPassword on page 83
- Section 3.11.14. TLS_RandomFile on page 84
- Section 3.11.15. TLS_DHFile on page 84
- Section 3.11.17. TLS_CRLCheck on page 85
- Section 3.11.20. TLS_CRLFile on page 85
- Section 3.11.44. TLS_SessionResumption on page 92
- Section 3.11.45. TLS_SessionResumptionLimit on page 92
- Section 3.11.22. TLS_ExpectedPeerName on page 86
- Section 3.11.23. TLS_SubjectAltNameURI on page 86
- Section 3.11.26. TLS_CertificateFingerprint on page 87

```
<AuthBy DNSROAM>
      # Defaults for all enclosed Routes:
      Port 1645
      Transport tcp
      Protocol radsec
      UseTLS 1
      Secret mysecret
      TLS_CAFile ....
      .....
      <Route>
            Realm realm3.open.com.au
            Address oscar.open.com.au
            # Override parameters for AuthBy RADSEC
            Secret xyzzy
            UseTLS 0
            ......
      </Route>
      ...
</AuthBy>
```

3.77. <AuthBy SAFEWORD>

This clause authenticates users from a local or remote SafeWord PremierAccess (SPA) server. SafeWord PremierAccess and tokens are available in Secure Computing website [http://securecomputing.intelsecurity.com/]. It supports PAP, CHAP, TTLS-PAP, EAP-OTP and EAP-GTC. It supports password changing, fixed (static) passwords and Safe- Word Silver and Gold tokens.

Only the first authenticator configured into SPA for the user will be used by Radiator to authenticate the user. The second and subsequent authenticators configured for a user will be ignored.

Note

In order to support CHAP, the user must have a fixed password profile with ‘case sensitive password’ enabled in the SafeWord server.

Note

A user can changed their fixed password at any time by entering their current and new passwords in a special format. The password change will only succeed if the old password is correct, and the two copies of the new password are identical and conform to the password length and other constraints configured into SPA for the user's fixed password authenticator.

```
oldpassword\cnewpassword,newpassword
```

<AuthBy SAFEWORD> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.77.1. Host

This parameter specifies the name or address of the SafeWord PremierAccess server to connect to. The connection will be made with SSL. Defaults to ‘localhost’.

3.77.2. Port

This parameter specifies the port name or number to connect to on Host. Defaults to 5031, the default SafeWord EASSP2 port.

3.77.3. SSLVersion

This optional parameter specifies SSL/TLS protocol version(s) to use when connection to the server. For further information, see Perl IO::Socket::SSL module documentation for SSL_version. The default value and supported versions depend on the IO::Socket::SSL, Net::SSLeay and OpenSSL version available on your system.

Here is an example of using *`SSLVersion`*:

```
SSLVersion TLSv1_2
```

3.77.4. SSLCipherList

This optional parameter specifies ciphers for SSL/TLS to use when connection to the server. For further information, see Perl IO::Socket::SSL module documentation for SSL_cipher_list. The default value and

supported versions depend on the IO::Socket::SSL, Net::SSLeay and OpenSSL version available on your system.

Here is an example of using *SSLCipherList*:

```
SSLCipherList DEFAULT:!EXPORT:!LOW
```

3.77.5. SSLVerify, SSLCAFile, SSLCAPath, SSLCAClientCert, SSLCAClientKey, SSLCAClientKeyPassword

These parameters are used to control the SSL connection to the SafeWord server. They behave as described in "<AuthBy IMAP>", see Section 3.59. <AuthBy IMAP> on page 258.

3.78. <AuthBy FREERADIUSSQL>

This clause handles authentication and accounting from a FreeRADIUS compatible SQL database. The default SQL queries use the standard FreeRadius tables *radcheck*, *radreply*, *radgroupcheck*, *radgroupreply* and *radacct*.

> **Tip**
>
> The sample configuration file in goodies/freeradiussql.cfg in your Radiator distribution shows how to set up a complete FreeRADIUS emulator. Use of this configuration file can make migration from an existing FreeRADIUS installation easy.

<AuthBy FREERADIUSSQL> understands also the same parameters as *<AuthBy SQL>*. For more information, see Section 3.41. <AuthBy SQL> on page 190.

3.78.1. ConvertCleartextPassword

ConvertCleartextPassword flag parameter causes FreeRADIUS **Cleartext-Password** check item to be used as a Radiator clear text password check item. When this flag is not set, Radiator does not consider **Cleartext-Password** a special password check item. All new configurations should enable this parameter. *ConvertCleartextPassword* currently defaults to not enabled for backwards compatibility.

```
# Use Cleartext-Password as a Radiator User-Password check item
ConvertCleartextPassword
```

3.78.2. AuthCheck

This optional parameter specifies an SQL query that is used to get check items for a user. Special characters are supported, as well as a single bind variable for the user name being searched. Defaults to:

```
SELECT id, UserName, Attribute, Value, op FROM radcheck WHERE
Username=? ORDER BY id
```

3.78.3. AuthReply

This optional parameter specifies an SQL query that is used to get reply items for a user. Special characters are supported, as well as a single bind variable for the user name being searched. Defaults to:

```
SELECT id, UserName, Attribute, Value, op FROM radreply WHERE
Username=? ORDER BY id
```

3.78.4. AuthGroupCheck

This optional parameter specifies an SQL query that is used to get check items for a user's group. Special characters are supported, as well as a single bind variable for the group name being searched. Defaults to:

```
SELECT radgroupcheck.id,radgroupcheck.GroupName,radgroupcheck.Attribute,
radgroupcheck.Value,radgroupcheck.op FROM
radgroupcheck,usergroup WHERE usergroup.Username = ? AND
usergroup.GroupName = radgroupcheck.GroupName ORDER BY radgroupcheck.id
```

3.78.5. AuthGroupReply

This optional parameter specifies an SQL query that is used to get reply items for a user's group. Special characters are supported, as well as a single bind variable for the group name being searched. Defaults to:

```
SELECT radgroupreply.id,radgroupreply.GroupName,radgroupreply.Attribute,
radgroupreply.Value,radgroupreply.op FROM
radgroupreply,usergroup WHERE usergroup.Username = ? AND
usergroup.GroupName = radgroupreply.GroupName ORDER BY radgroupreply.id
```

3.78.6. AcctOnoffQuery, AcctStartQuery, AcctStartQueryAlt, AcctUpdateQuery, AcctUpdateQueryAlt, AcctStopQuery, AcctStopQueryAlt

These optional parameters specify SQL queries to handle various accounting requests by inserting or updating the *radacct* table. The 'Alt' versions are run if the non-Alt version fails (usually because the appropriate accounting record is missing).

3.78.7. AcctFailedLogFileName, AcctLogFileFormat and AcctLogFileFormatHook

These parameters have the same meaning as described in *<AuthBy SQL>*, see Section 3.41. <AuthBy SQL> on page 190.

3.79. <AuthBy RADIUSdeskSQL>

This clause handles authentication and accounting using a RADIUSdesk-compatible SQL database. The *<AuthBy RADIUSdeskSQL>* module inherits parameters from *<AuthBy SQL>*. *<AuthBy RADIUSdeskSQL>* can be considered an extension of *<AuthBy FREERADIUSSQL>*, offering additional functionality and features. The following list highlights the enhancements provided by *<AuthBy RADIUSdeskSQL>* compared to *<AuthBy FREERADIUSSQL>*:

User Profile Support
: If a User-Profile check attribute is specified for a user, additional queries are performed to determine the user's profile.

Account Disablement
: When a permanent user is disabled in RADIUSdesk, the *Rd-Account-Disabled* check attribute is set, and the module rejects the authentication request with the message "Account Disabled".

Gigaword Support
: Support for Acct-Input-Gigawords in accounting records.

Realm Restrictions
: Support for Dynamic Client to Realm or NAS to Realm restrictions.

Auto-Registration
: Option to auto-register unknown users.

Custom Attribute Recording

Option to specify an attribute to be recorded during Access Request and Accounting Start requests.

Group Identification

Option to specify a Group Name and Group ID per Radiator instance. These values will be included with the SQL statement for the radacct table used during Accounting Start. This feature is particularly useful in large deployments with multiple Radiator instances where database sharding is implemented.

For a configuration sample, see a configuration sample file `goodies/fidelio-hotspot.cfg`

Background

Before diving into the configurable parameters, it's essential to understand some key concepts in RADIUSdesk. This background information will provide context and help you better comprehend the parameters and their applications.

Realms

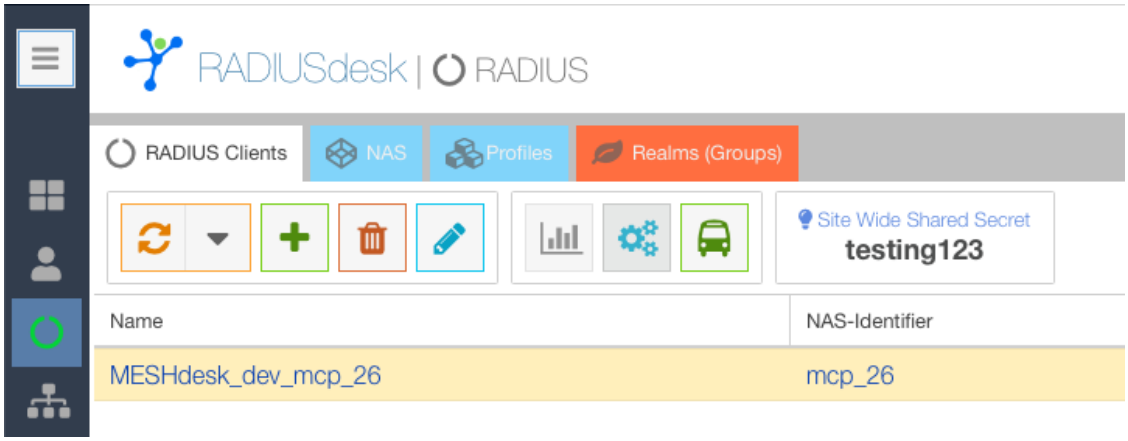
In RADIUS, a realm is typically specified as a suffix to a username, separated by a delimiter (e.g., `jsmith@example.com`, where "`jsmith`" is the username and "`example.com`" is the realm). However, RADIUSdesk offers more flexibility by allowing you to group users by realm without requiring the realm component to be part of the username. This approach provides greater freedom when defining usernames.

Dynamic Clients and NAS

RADIUSdesk offers two methods for declaring RADIUS clients.

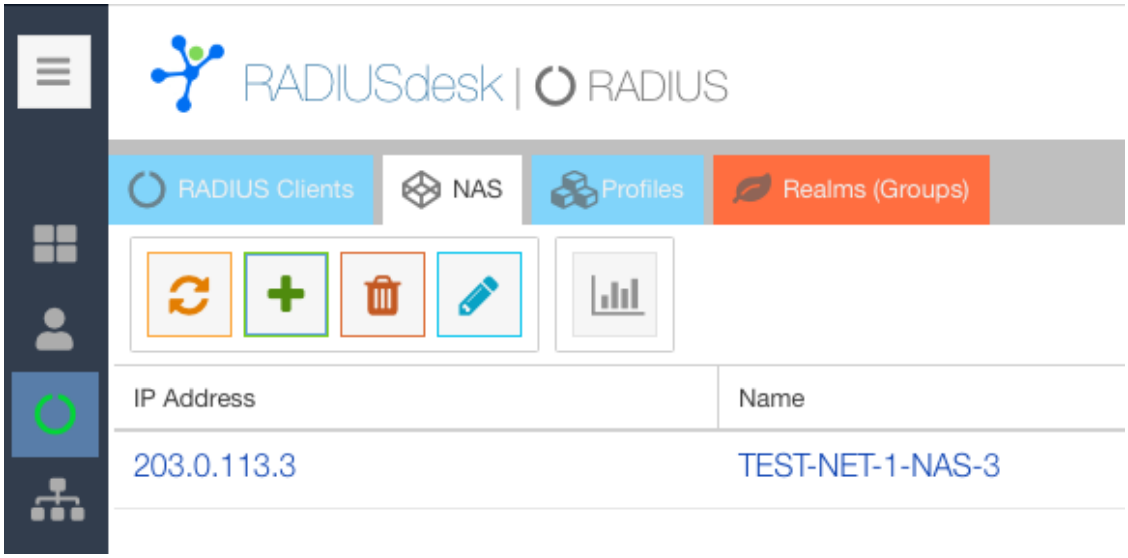
1. *Dynamic Clients*

All Dynamic Clients have a common shared secret and are identified by the *`Nas-Identifier`* attribute of the incoming RADIUS request. This is typically used for clients that contact the RADIUS server from an unknown IP Address.



2. *NAS*

Each NAS is identified by the **IP Address** and each NAS can also have a unique shared secret. This is typically used for clients that contact the RADIUS server from a known IP Address.



Restricting Realms

RADIUSdesk enables realm restrictions for Dynamic Clients and NAS entries. You can limit access to specific realms, ensuring that users from unauthorized realms are denied network access. If a user who belongs to a realm that is not included attempts to access the network, the authentication request will be rejected.

Profiles

RADIUSdesk has a Profile applet used to manage user profiles. Each profile can contain multiple Profile Components. A Profile Component consists of a number of check and reply attributes. These attributes are selected from the dictionaries provided by FreeRADIUS. Sometimes the dictionaries in FreeRADIUS are different compared to the dictionary in Radiator. How to take care of this will be covered later in the document with a practical example.

Password Attribute

In recent FreeRADIUS releases, user passwords are stored in the Cleartext-Password attribute. This is a change from older releases, which used the User-Password attribute. RADIUSdesk follows the same convention as recent FreeRADIUS releases, using Cleartext-Password for storing user passwords.

Register Users

RADIUSdesk provides a comprehensive API that mirrors the functionality of its web GUI. With `<AuthBy RADIUSdeskSQL>`, you can leverage this API to auto-register unknown users.

RADIUSdesk and Radiator Dictionary Compatibility

RADIUSdesk utilizes FreeRADIUS dictionaries when formulating Profile Components. However, differences may exist between FreeRADIUS dictionary entries and those in Radiator.

Next will be a couple of practical points on environments where Juniper BNGs are used as RADIUS clients. For instance, the ERX dictionary used by Juniper equipment (typically located at `/usr/share/freeradius/dictionary.erx`) may have discrepancies.

FreeRADIUS dictionary snippet:

```
VENDOR        ERX 4874
BEGIN-VENDOR  ERX
ATTRIBUTE     ERX-Virtual-Router-Name 1 string
```

Radiator dictionary snippet:

```
#
# Define additional Unisphere ERX Family Attributes
# based on http://www.juniper.net/techpubs/software/junos/junos114/
# radius-dictionary/unisphereDictionary_for_JUNOS__v11-4.dct
#
VENDOR     ERX 4874
VENDORATTR 4874 Unisphere-Virtual-Router 1 string
```

Ensuring Compatibility

Since Radiator queries the SQL database using these values, it's crucial that Radiator can resolve the values from the SQL database. To achieve compatibility, use RADIUSdesk's dictionary override feature to align the FreeRADIUS dictionary with the Radiator dictionary. For more information, see: https://radiusdesk.com/wiki24/technical/fr-dictionary [https://radiusdesk.com/wiki24/technical/fr-dictionary].

Disconnecting Active Sessions

RADIUSdesk's latest version features a Packet of Disconnect (POD) functionality, allowing you to send disconnect requests to BNG/RADIUS clients. This feature is triggered by the following actions:

- Kicking an active session o# using the "Kick User Off" button.
- Disabling a permanent user with an active session.
- Changing the profile of a permanent user with an active session.

The disconnect method used is determined by the Type option of a Dynamic Client or NAS in RADIUSdesk, with the logic defined in `/var/www/rdcore/cake4/rd_cake/src/Controller/Component/KickerComponent.php`

The Juniper type of equipment will typically require RADIUSdesk to send a COA/POD to the BNG. The snippet below shows the code from `KickerComponent.php`:

```
private function kickJuniperSession($ent){
  // -- Sample Disconnect –
  // echo "Acct-Session-ID='2040',User-Name='zaguy@example.com'" | radclient \
  // -c '1' -n '3' -r '3' -t '3' -x '127.0.0.1:3799' 'disconnect' 'testing123'
  $sessionid = $ent->acctsessionid;
  $username  = $ent->username;
  $ip        = $ent->nasipaddress;
  $fwd_ip    = $ip; // You can replace this with a central IP to forward it to
  $secret    = 'testing123';

  shell_exec("echo \"Acct-Session-ID='$sessionid',User-Name='$username',
  NAS-IP-Address='$ip'\" | radclient -c '1' -n '3'
  -r '3' -t '3' -x '$fwd_ip:3799' 'disconnect' '$secret'");
}
```

AuthBy RADIUSdeskSQL configuration parameters

Having covered these key concepts, we can now explore the available parameters for *<AuthBy RADIUSdeskSQL>*.

3.79.1. RealmChkDynamicClients

When set, this optional flag enables *<AuthBy RADIUSdeskSQL>* to apply restrictions defined for RADIUSdesk Dynamic Client items. These restrictions are based on the user's realm and the `Nas-Identifier` in the authentication request. Defaults to not enabled.

```
# Enable this flag parameter
RealmChkDynamicClients
```

3.79.2. RealmChkNas

When set, this optional flag enables *<AuthBy RADIUSdeskSQL>* to apply restrictions defined for RADIUSdesk NAS items. These restrictions are based on the user's realm and the IP address of the authentication request. Defaults to not enabled.

```
# Enable this flag parameter
RealmChkNas
```

3.79.3. AutoRegister

This optional flag enables the auto-registration feature. When set, additional mandatory parameters are required to ensure proper user registration functionality. Defaults to not enabled.

The following example shows all parameters that are required or typically configured when AutoRegister is enabled. Each parameter is described in detail separately.

```
AutoRegister
RegUrl       https://radiusdesk.example.com/cake4/rd_cake/permanent-users/add.json
RegSslCaFile %D/certs/radiusdesk-ca.pem
RegToken     c03cae5d-392c-4cb5-b422-4c628d451ced
RegCloudId   24
RegProfile   AUTOREGISTER
RegRealm     samplerealm
AutoRegisterHook file:"%D/rdAddPermanentUser.pl"
```

3.79.4. RegUrl

This parameter is required when AutoRegister on page 317 is enabled. It specifies the API endpoint for registering users on RADIUSdesk, typically in the format: `http[s]://<IP address|FQDN>/cake4/rd _ cake/permanent-users/add.json`

When using a Fully Qualified Domain Name (FQDN), it's recommended to add the entry to the Radiator server's hosts file to ensure functionality in case of DNS failure.

```
RegUrl https://radiusdesk.example.com/cake4/rd_cake/permanent-users/add.json
```

3.79.5. RegSslCaFile

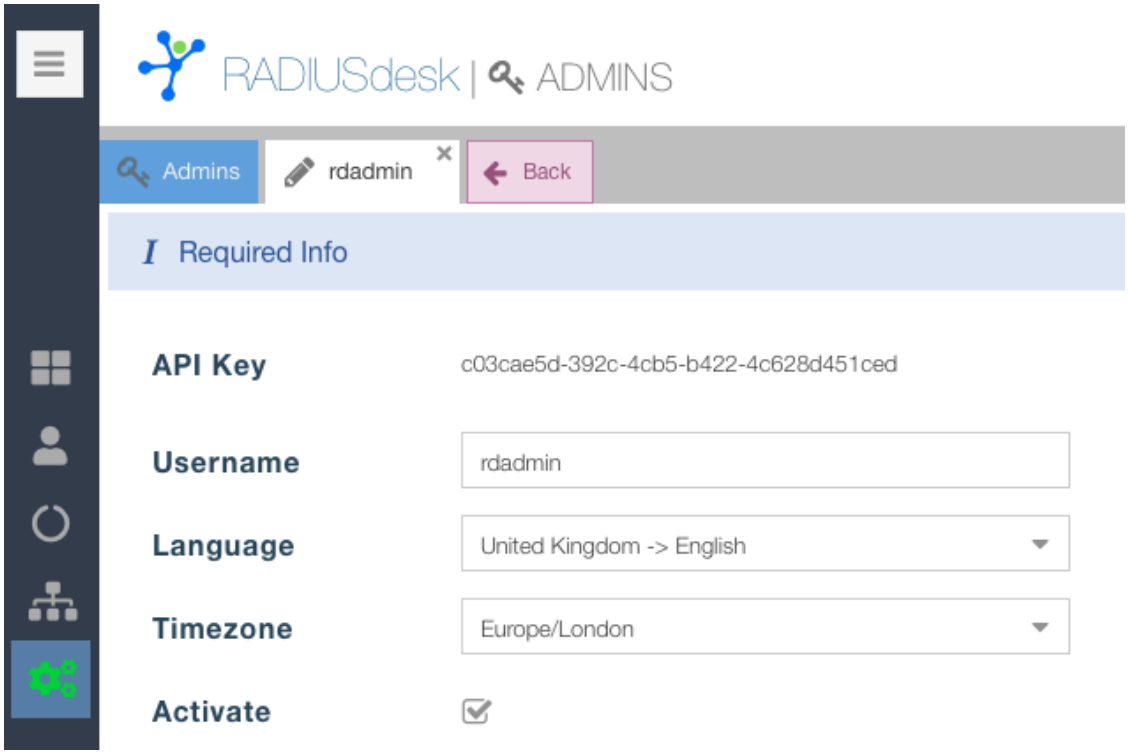
This optional parameter is used with AutoRegister on page 317. It locates the file containing the certificates of the trusted certificate authorities. If not defined when using `https://` in RegUrl on page 317, *<AuthBy RADIUSdeskSQL>* will disable certificate verification. If set, certificate verification will be enforced. It's recommended to store the Certificate Authority file in the DbDir on page 34 subdirectory named `certs`, for example `/etc/radiator/certs/`.

```
RegSslCaFile %D/certs/radiusdesk-ca.pem
```

3.79.6. RegToken

This parameter is required when AutoRegister on page 317 is enabled. It specifies the API key that identifies a RADIUSdesk admin. This is used to authenticate API calls. The recommended way is to create a dedicated admin in RADIUSdesk for user registration and use the API key displayed when editing this admin's profile. Note that the API key will change if the admin's password is updated.

```
RegToken c03cae5d-392c-4cb5-b422-4c628d451ced
```



3.79.7. RegCloudId

This parameter is required when AutoRegister on page 317 is enabled. It specifies the ID of the Cloud under which the user will be registered. In RADIUSdesk, Clouds are entities that group items, and each Cloud has a unique ID. You can find the Cloud ID in the Clouds applet under the Cloud ID column.

```
RegCloudId 24
```

3.79.8. RegProfile

This parameter is required when AutoRegister on page 317 is enabled. It specifies the profile that will be assigned to the newly registered user. The profile name must match an existing profile defined in RADIUSdesk.

```
RegProfile AUTOREGISTER
```

3.79.9. RegRealm

This parameter is required when AutoRegister on page 317 is enabled. It specifies the realm that will be assigned to the newly registered user. The realm name must match an existing realm defined in RADIUSdesk.

```
RegRealm samplerealm
```

3.79.10. AutoRegisterHook

This parameter is required when AutoRegister on page 317 is enabled. It the hook to register the user. A sample hook named `rdAddPermanentUser.pl` is in goodies directory `radiusdesk`.

`AutoRegisterHook` is called with the following arguments:

1. A handle to the current `Radius::AuthRADIUSdeskSQL` object
2. The username to be registered
3. Reference to the current request

Here's an example of configuring *`AutoRegisterHook`*. This assumes we have our AutoRegisterHook file called `rdAddPermanentUser.pl` in the *`DbDir`* on page 34 directory `/etc/radiator`

```
AutoRegisterHook file:"%D/rdAddPermanentUser.pl"
```

3.79.11. RecordExtraAttr1

This optional parameter specifies a RADIUS attribute that, if present in the incoming request, will:

- Update the `extra_name` field of a permanent user upon successful authentication.
- Populate the `operator_name` field in the `radacct` table on accounting start requests.

You can also add custom attributes to the RADIUS request packet before it reaches *`<AuthBy RADIUSdeskSQL>`*, allowing for flexibility beyond the attributes included in the original request from the RADIUS client.

```
RecordExtraAttr1 OSC-User-Identifier
```

3.79.12. AcctGroupname

This optional parameter records a string value in the `groupname` field of the `radacct` table on accounting start requests. It's useful in large deployments with multiple Radiator instances sharing a database, allowing you to tag or group instances with specific values, such as geographical locations.

```
AcctGroupname AU-01
```

3.79.13. AcctGroupId

This optional parameter records an integer value in the `group_id` field of the `radacct` table on accounting start requests. It's useful in large deployments with multiple Radiator instances sharing a database, enabling sharding to segment database tables.

To use this feature, apply the RADIUSdesk SQL patch `9.001_add_group_id_for_sharding.sql`. See RADIUSdesk installation instructions for the details of applying RADIUSdesk SQL patches.

```
AcctGroupId 7
```

3.79.14. ConvertCleartextPassword, AuthCheck, AuthReply and other <AuthBy FREERADIUSSQL> parameters

`ConvertCleartextPassword` flag parameter is required in *`<AuthBy RADIUSdeskSQL>`* because RADIUSdesk stores user passwords as values in the `Cleartext-Password` check attribute in SQL. This parameter defaults to enabled and should only be disabled in special cases.

```
# Special configuration: disable this flag parameter
```

```
ConvertCleartextPassword 0
```

AuthCheck, *AuthReply* and other parameters also available in *AuthBy FREERADIUS* should only be set in special cases. At the time of writing, the default values match the RADIUSdesk defaults.

3.80. <AuthBy HTGROUP>

Checks group membership according to an Apache htgroup file. Apache is a popular HTTP server, and supports flat group files for authenticating web access. With <AuthBy HTGROUP> Radiator can authenticate users against the same files that Apache uses, in order to provide a single source of authentication information for both Apache and Radiator.

In order to use <AuthBy HTGROUP>, the user record must specify an AuthBy check item naming the Identifier of the AuthBy HTGROUP to be used. See the example in `goodies/htgroup.cfg` for details. During authentication, the user's GroupList check items will be used to check for group membership against the Apache group file specified by the GroupFilename parameter.

3.80.1. GroupFilename

Specifies the name of the Apache group file to consult. Special characters are supported (so you may have a different group files for different categories of users).

3.81. <AuthBy HOTSPOT>

<AuthBy HOTSPOT> provides advanced support for hotspot functionality, including differentiated services and prepaid/postpaid quotas. It combines *<ServiceDatabase>*, *<SessionDatabase>*, an authenticating *<AuthBy xxxxxx>* to a working solution which can be used with captive portals (for example MikroTik) and network access controllers supporting RADIUS. Besides hotspot use-case, *<AuthBy HOTSPOT>* can also be used to implement quota control for fixed-line access or cellular APN. It can also utilise *<AuthBy DYNAUTH>* (RADIUS Dynamic Authorization) for sending CoA/DM, for example, after a successful authentication or when a quota has been depleted.

See `goodies/hotspot.cfg` and `goodies/README.hotspot` for a sample configuration file. For a derived AuthBy, see Section 3.83. <AuthBy HOTSPOTFIDELIO> on page 330.

<AuthBy HOTSPOT> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.81.1. ServiceId

This optional string parameter specifies how to derive the Identifier of the Service to provision a subscription. Special variable %0 contains identifier of this AuthBy. There is no default value.

3.81.2. SubscriptionId

This optional string parameter specifies how to derive the Identifier of the Subscription to handle a request. Special variable %0 contains identifier of this AuthBy, %1 the username and %2 hotspot id . Default value is %1.

```
# Lookup a subscription based on username (%1) and Calling-Station-Id MAC address
SubscriptionId %1-%{Calling-Station-Id}
```

3.81.3. SessionId

This optional string parameter specifies how to derive the Identifier of the Session to handle a request. Special variable %0 contains identifier of this AuthBy, %1 the username , %2 hotspot id and %3 Class attribute value . There is no default value.

```
# Lookup a session based on username, Class and Calling-Station-Id MAC address
SessionId %1-%3-%{Calling-Station-Id}
```

3.81.4. ServiceAttribute

This optional string parameter specifies name of RADIUS attribute that is used to detect different services. There is no default value.

```
# Our hotspot is configured this send this attribute in RADIUS requests
ServiceAttribute OSC-AVPAIR
```

3.81.5. ServiceAttributePrefix

This optional string parameter specifies the prefix in ServiceAttribute value. You need to specify this if there is possibility to have multiple RADIUS attributes that are defined to be ServiceAttribute. There is no default value.

```
# Access-Request from our captive portal has multiple instances
# of ServiceAttribute, use the ones with this prefix
ServiceAttributePrefix Mikrotik-Service=
```

3.81.6. SubscriptionAttribute

This optional string parameter specifies name of RADIUS attribute that is used to detect different subscriptions. There is no default value.

3.81.7. ServiceAttributePrefix

This optional string parameter specifies the prefix in SubscritionAttribute value. You need to specify this if there is possibility to have multiple RADIUS attributes that are defined to be SubscriptionAttribute. There is no default value.

3.81.8. SessionAttribute

This optional string parameter specifies name of RADIUS attribute that is used to detect different sessions. If this is set to an empty value, *`SessionId`* parameter is used to detect sessions. Defaults to *`Acct-Session-Id`*.

```
# Set this to an empty value
SessionAttribute
```

3.81.9. SessionAttributePrefix

This optional string parameter specifies the prefix in SessionAttribute value. You need to specify this if there is possibility to have multiple RADIUS attributes that are defined to be SessionAttribute. There is no default value.

3.81.10. DefaultService

This optional string parameter specifies a name of default service when the request does not contain ServiceAttribute. Defaults to *`default`*.

```
# Name of default service defined in our ServiceDatabase
DefaultService free
```

3.81.11. DefaultServiceDefinition

This optional string parameter defines the default service to use when no service definition was found from our ServiceDatabase. Defaults to *`name:default price:0`*.

```
# Make this empty for no default configured service
DefaultServiceDefinition
```

3.81.12. ConfirmSubscription

This optional flag parameter specifies if RADIUS *`Access-Reject`* message with *`Reply-Message`* attribute is used to ask for a confirmation of the upgrade or renewal charge. Disabled by default.

```
# Use Access-Reject with Reply-Message for confirmation
ConfirmSubscription
```

3.81.13. ConfirmationMessage

This optional string parameter specifies the message that will ask the guest to confirm the upgrade or renewal charges when *`ConfirmSubscription`* flag parameter is set. Defaults to: *`You are going to upgrade or renew your plan, please login again to confirm the charge.`*

```
# Change the default to more specific
ConfirmationMessage Please log in again to approve Wi-Fi charge at checkout.
```

3.81.14. PreProvisionSubscription

Set this optional flag parameter to create a subscription externally before the authentication. Defaults to not set.

3.81.15. PreProvisionSession

Set this optional flag parameter to create a session when the authentication is accepted. Defaults to not set.

```
# If RADIUS accounting will be used for quota monitoring,
# create a new session upon a successful authentication
PreProvisionSession
```

3.81.16. AuthBy

This string parameter specifies which AuthBy will be used to authenticate users. Special formatting characters are permitted. Defaults to not set but must be configured with a value.

```
# Authenticate Hotspot users with AuthBy that has this identifier
AuthBy AuthBy-FILE
```

3.81.17. ServiceDatabase

This string parameter specifies which ServiceDatabase will be used to query service and subscription information. Special formatting characters are permitted. Defaults to not set but should be configured with a value.

```
# Use ServiceDatabase with this identifier for services and subscriptions
ServiceDatabase ServiceDatabase-INTERNAL
```

3.81.18. SessionDatabase

This string parameter specifies which SessionDatabase will be used to query session information. Special formatting characters are permitted. Defaults to not set but should be configured with a value.

```
# Use SessionDatabase with this identifier for sessions
SessionDatabase SessionDatabase-INTERNAL
```

3.81.19. PoolIPv4Attribute

This optional string parameter specifies name of RADIUS attribute to return when Service/Subscription has IPv4 pool. Defaults to *`PoolHint`*.

```
# Use non-default IPv4 pool attribute name to return service pool name
PoolIPv4Attribute Framed-Pool
```

3.81.20. PoolIPv6Attribute

This optional string parameter specifies name of RADIUS attribute to return when Service/Subscription has IPv6 pool. Defaults to *`PoolHint6`*.

```
# Use non-default IPv6 pool attribute name to return service pool name
PoolIPv6Attribute Framed-IPv6-Pool
```

3.81.21. UsageMonitoring

When this optional flag parameter is set, deplete Subscription quotas only based on RADIUS accounting. Disabled by default.

```
# Use accounting and return to NAS remaining time quota with Session-Timeout
# and data quota with Mikrotik attributes.
UsageMonitoring
ReplyWithDataQuota
DataLimitAttribute Mikrotik-Total-Limit
DataLimitGigawordsAttribute Mikrotik-Total-Limit-Gigawords
```

3.81.22. ReplyWithBandwidthControl

When this optional flag parameter is set, add Service/Subscription bandwidth limit RADIUS attributes to the reply. Enabled by default.

```
# Enable BW control for pre-paid, post-paid and throttled (quota exceeded) cases
ReplyWithBandwidthControl
UploadRateAttribute WISPr-Bandwidth-Max-Up
DownloadRateAttribute WISPr-Bandwidth-Max-Down
```

3.81.23. UploadRateAttribute

This optional string parameter specifies name of RADIUS attribute that is used to return Service/Subscription Upload bandwidth limit. Default value is *`OSC-Upload-Rate`*. See Section 3.81.22. ReplyWithBandwidthControl on page 323 for an example.

3.81.24. DownloadRateAttribute

This optional string parameter specifies name of RADIUS attribute that is used to return Service/Subscription Download bandwidth limit. Default value is *`OSC-Download-Rate`*. See Section 3.81.22. ReplyWithBandwidthControl on page 323 for an example.

3.81.25. ReplyWithDataQuota

When this optional flag parameter is set, Add Service/Subscription data limit RADIUS attributes to the reply. Disabled by default. See Section 3.81.21. UsageMonitoring on page 323 for an example.

3.81.26. DataLimitAttribute

This optional string parameter specifies name of RADIUS attribute that is used to return Service/Subscription data limit. Default value is *`OSC-Data-Limit`*. See Section 3.81.21. UsageMonitoring on page 323 for an example.

3.81.27. DataLimitGigawordsAttribute

This optional string parameter specifies name of RADIUS attribute that is used to return Service/Subscription data limit gigawords. Default value is *`OSC-Data-Limit-Gigawords`*. See Section 3.81.21. UsageMonitoring on page 323 for an example.

3.81.28. ChargeHook

This optional parameter allows you to define a Perl function to modify a subscription charge.

`ChargeHook` must return new username, a charging reference and new price. Charging reference can be undefined. If username is undefined, charging is not done and request is rejected.

`ChargeHook` has the following arguments:

- The current request
- Username
- Id, such as room number, plane seat number or loyalty card number or other value typically sent as password with a hotspot service
- Price
- Subscription

Here is an example of *`ChargeHook`*:

```
# Charge mikem with double rate
ChargeHook sub {my $name = $_[1]; my $price = $_[3]; \
                return ($name, undef, $price*2);}
```

3.81.29. ReplyAdjustHook

This optional parameter allows you to define a Perl function to adjust RADIUS *`Access-Accept`* reply.

`ReplyAdjustHook` return value is ignored. However, it is passed references to current request and reply for modification.

`ReplyAdjustHook` has the following arguments:

- The current request
- The current reply

- Time left
- Octets left
- Gigawords left
- Reference to the Service object
- Reference to the Subscription object
- Reference to the Session object
- Reference to the current AuthBy

Here is an example of *ReplyAdjustHook*:

```
# Reply debugging for radpwtst testing
ReplyAdjustHook sub {my $rp = $_[1]; my $time_left = $_[2]; \
                     $rp->add_attr('OSC-AVPAIR', "time-debug=$time_left"); }
```

3.81.30. SessionQuotaSupplyFraction

This optional integer parameter specifies percentage (1-100) of remaining Subscription quota to allocate for a session. Defaults to 20.

```
# Allocate only 10%
SessionQuotaSupplyFraction 10
```

3.81.31. SessionQuotaResupplyThreshold

This optional integer parameter specifies percentage threshold (1-100) for session quota usage to resupply session quotas from a subscription. Defaults to 80.

```
# Experiment with 90% threshold
SessionQuotaResupplyThreshold 90
```

3.81.32. SessionQuotaResupplyTimeMin

This optional string parameter specifies the minimum amount of time quota to allocate from a subscription for a session. Value is seconds or a string with one letter suffix of y,d,h,m for year, day, hour or minute, respectively. Defaults to 10 minutes.

```
# Change resupply time to 20 minutes. 1200 would work also as the value.
SessionQuotaResupplyTimeMin 20m
```

3.81.33. SessionQuotaResupplyDataMin

This optional string parameter specifies the minimum amount of data quota to allocate from a subscription for a session. Value is octets or a string with one letter suffix of G,M,K for 10^9, 10^6 or 10^3 octets, respectively. Defaults to 1M octets.

```
# Change resupply data to 20 000 000 octets. Can use 20M or 20000000 as the value.
SessionQuotaResupplyDataMin 20M
```

3.82. <AuthBy FIDELIO>

<AuthBy FIDELIO> authenticates users from Micros-Fidelio Opera. Opera is a popular hotel Property Management System from Micros Fidelio [http://www.micros-fidelio.eu/].

AuthBy FIDELIO acquires the hotel guest list from Opera, and by default, authenticates each guest using their Room Number as the User-Name and their Guest Number as their password (this is configurable). By default, RADIUS Accounting Stops generate Opera PS simple postings. This can be changed with the PostingRecordID parameter.

Note

If the acquired guest list is empty, this may be caused by Opera settings where the full guest list synchronisation is disabled. This option needs to be enabled for AuthBy FIDELIO to work.

By default, accounting postings are sent to Opera following receipt of an Accounting-Stop. The default PS record will contain:

- P# Automatically generated posting sequence number, starting at 1
- TA the cost computed by ComputeCostHook or CentsPerSecond
- DU The call duration in seconds
- PT fixed value of ‘C’ (Direct Charge)
- SO fixed value of ‘Internet connection’
- DD Called-Station-Id from the incoming request, with any non-digit characters stripped out

Additional fields can be added to the Opera posting record with the PostingExtraFields parameter.

AuthBy FIDELIO requires a connection to the Opera computer system. Both RS232 serial and TCP/IP (Ethernet) connections are supported. RS232 support requires Device::SerialPort. It is part of CPAN. For more information, see Section 2.1.2. CPAN on page 3.

3.82.1. Protocol

Specifies the protocol to be used to connect to Opera. ‘serial’ or ‘tcp’ are supported. If ‘serial’ is specified, AuthBy FIDELIO will open the serial port specified by Port and configure it to use the RS232 parameters specified by Baudrate, Databits, Parity and Stopbits and Handshake. If ‘tcp’ is specified, AuthBy FIDELIO will connect to Opera by TCP/IP, using the parameters Host and Port. Defaults to ‘tcp’.

3.82.2. Port

When Protocol is serial, specifies the device name of the serial port to use (defaults to ‘/ dev/ttyS0’). When Protocol is tcp, specifies the service name or number of the TCP/IP port to connect to (defaults to 5010, the standard Opera TCP/IP port).

3.82.3. Host

When Protocol is tcp, specifies the DNS name or address of the Opera host. Defaults to ‘localhost’.

3.82.4. Baudrate

When Protocol is serial, specifies the baud rate for the serial port. Defaults to 9600.

3.82.5. Databits

When Protocol is serial, specifies the number of data bits for the serial port. Defaults to 8.

3.82.6. Parity

When Protocol is serial, specifies the data parity for the serial port. May be ‘odd’, ‘even’ or ‘n’. Defaults to ‘n’.

3.82.7. Stopbits

When Protocol is serial, specifies the number of stop bits for the serial port. Defaults to 1.

3.82.8. Handshake

When Protocol is serial, specifies the handshaking standard for the serial port. May be ‘none’, ‘rts’ or ‘xoff’. Defaults to ‘xoff’.

3.82.9. ReadCharTimeout

When Protocol is serial, specifies the character read timeout in milliseconds. Defaults to 2000ms. This should not need to be changed.

3.82.10. TransmitTimeout

Specifies the time in seconds to wait before retransmitting an unacknowledged message to Opera. Default to 2 seconds. This should not need to be changed.

3.82.11. ReconnectTimeout

Specifies the time in seconds to wait before attempting to reconnect to Opera after the connection is lost. Defaults to 5 seconds.

3.82.12. InterfaceFamily

Specifies the interface family that AuthBy FIDELIO will send to Opera, identifying the type of interface Opera is to use to communicate with Radiator. Defaults to ‘WW’, the standard Opera code for Internet.

3.82.13. FieldSeparator

Specifies the field separator character to be sued by Opera. Defaults to ‘|’. This should not need to be changed.

3.82.14. BindAddress

When Protocol is tcp, specifies the bind address to be used for the TCP/IP port. Defaults to the global BindAddress (if specified) or 0.0.0.0. Useful to specify the source address interface to use for multi-homed hosts. For more information, see Section 3.7.9. BindAddress on page 33.

3.82.15. MaxBufferSize

Specifies the maximum read buffer size. Defaults to 100000. Should not need to be changed.

3.82.16. UseChecksums

Specifies whether to use checksums in the communications with Opera. The correct setting for this depends on the version of Opera, but a setting of disabled will work for most modern installations. Defaults to enabled (1) for serial Protocol and disabled (0) for tcp Protocol.

3.82.17. LinkRecords

Specifies the list of required fields for each message type in messages passed between Radiator and Opera. May be specified one per line for each type of message. This should only need to be changed if additional data is required for each gust in the GI (Guest In), GO (Guest Out) and GC (Guest Change) messages. Defaults to:

```
GI    FLG#RNGNSF
GO    FLG#RNS
GC    FLG#RNGN
PS    FLP#RNPTTATIDUDADDSOCT
PA    FLASRNP#DATICT
```

3.82.18. GuestNameField

Specifies the name of the Opera guest field that will be used match the User-Name in authentication requests. Defaults to 'RN' (Room Number).

3.82.19. GuestPasswordField

This parameter specifies the name of the Opera guest field that is used to match the password in authentication requests. The default value is 'G#' (Guest Number). Usually there is no need to change this parameter. Use *`UserPasswordHook`* instead to extract password from user information.

3.82.20. ComputeCostHook

Specifies an optional Perl hook that can be used to calculate the cost when a RADIUS Accounting Stop is received. The hook is passed the call duration in seconds, and is expected to return the cost in cents. If ComputeCostHook is not specified, then CentsPerSecond will be used to calculate the cost.

3.82.21. UserPasswordHook

Specifies an optional Perl hook that can be used to determine the correct password for the user. It is expected to return the correct user password in plaintext.

The hook is passed the following arguments:

- $_[0] A reference to the current AuthBy module object
- $_[1] A reference to a hash containing the guest's Opera data (as received in a GI message from Opera)
- $_[2] A reference to the current RADIUS request

The default is:

```
UserPasswordHook sub {return $_[1]->{'G#'}}
```

which specifies the guests Guest Number (G#) is to be used as their password.

3.82.22. CentsPerSecond

When UserPasswordHook is not specified, this number will be used to compute the cost to be used in the accounting PS resulting from a RADIUS Accounting Stop. The charge will be computed from the call duration (in seconds) times CentsPerSecond.

3.82.23. PostingRecordID

Defines the type of transaction code to be used for sending Posting records to Opera. Defaults to PS but can be changed to PR.

CAUTION

Use of PR would also require a suitable LinkRecords entry for the desired PR data. For more information, see the example in Section 3.82.24. PostingExtraFields on page 329.

Contact your Micros representative for more details on field types and codes that can be sent to Opera.

3.82.24. PostingExtraFields

List of fields that are to be added to the standard fields sent to Opera in a Posting transaction. These are sent in addition to the standard ones of P#, TA, DU, PT, SO, CT and DD. Format is in the form: <fieldid>,<data>. Where <fieldid> is the 2 letter FieldID and <data> is the data to be sent in that field (special characters are permitted).

The special characters in <data> may include:

- %0 the user (room) name
- %1 the posting sequence number
- %2 the cost computed by CentsPerSecond or ComputeCostHook
- %3 the duration of the connection in 'hhmmss' format
- %4 the reservation number (G#) corresponding to the room number

In the following example, the type of posting is changed to PR, and the PR is defined to use the P#, TA, DU, PT, SO, CT and DD field (the defaults) and the additional field T1 (tax code). When a posting is sent, the T1 tax code is set to 3, and the additional G# (reservation number) is set to the user's reservation number from the Opera database.

```
PostingRecordID PR
LinkRecords PR,FLP#RNPTTATIDUDADDSOCTT1
PostingExtraFields T1,3
PostingExtraFields G#,%4
```

3.82.25. MessageHook

MessageHook is called after a message from Fidelio has been unpacked into a hash and before the record is passed to handle_message(). It can be used to change or transform any fields in the record before it is passed to handle_message() and processed by Auth- FIDELIO. The first argument is a pointer to the AuthBy FIDELIO clause, and the second is a pointer to a hash containing the unpacked record fields.

3.82.26. CheckoutGraceTime

Specifies a number of seconds after check-out to still allow a user to log in. Defaults to 0.

3.82.27. HandleAcctStatusTypes

This optional parameter specifies a list of `Acct-Status-Type` attribute values that will be processed in Accounting requests. The value is a comma-separated list of valid `Acct-Status-Type` attribute values including, **`Start`**, **`Stop`**, **`Alive`**, **`Modem-Start`**, **`Modem-Stop`**, **`Cancel`**, **`Accounting-On`** and **`Accounting-Off`**. See your dictionary for a full list.

If *`HandleAcctStatusTypes`* is specified and an Accounting request has an `Acct-Status-Type` not mentioned in *`HandleAcctStatusTypes`*, then the request will be ACCEPTed but not otherwise processed by the enclosing clause. The default is to handle all `Acct-Status-Type` values.

```
# Only process Start and Stop requests, ACCEPT and acknowledge everything else
HandleAcctStatusTypes Start,Stop
```

3.83. <AuthBy HOTSPOTFIDELIO>

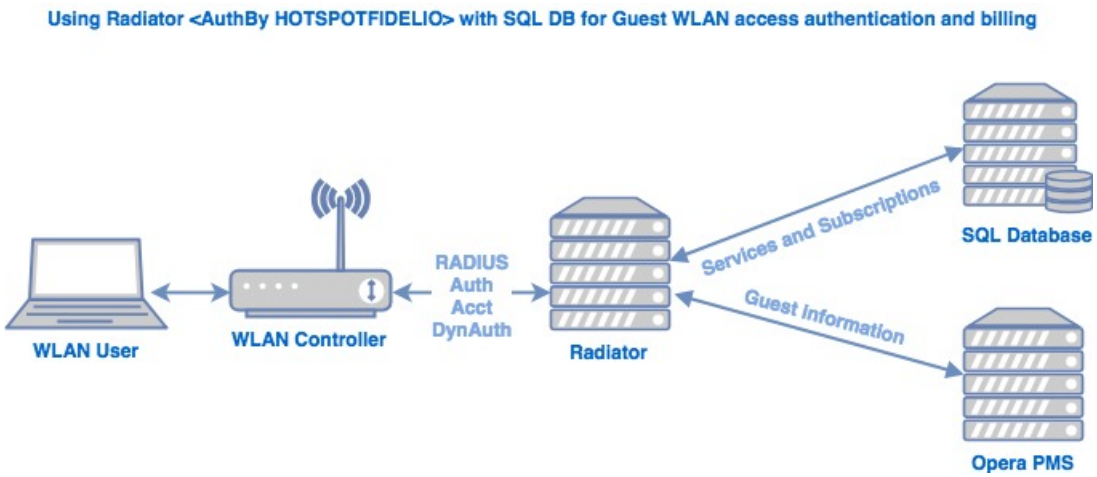
<AuthBy HOTSPOTFIDELIO> combines *<AuthBy HOTSPOT>* and *<AuthBy FIDELIO>* to an easy way to integrate WiFi hotspots and captive portals with Opera hotel Property Management System from Micros Fidelio [http://www.micros-fidelio.eu/].

It uses a service database to keep track of prepaid access time blocks for Opera guests and tracks ongoing sessions with a session database. It authenticates new sessions with the guest's room number and guest number (as with *<AuthBy FIDELIO>*), creates new prepaid blocks (and posts the charge to Opera). Time-limited prepaid plans are supported.

<AuthBy HOTSPOTFIDELIO> understands also the same parameters as *<AutBy HOTSPOT>* and *<AuthBy FIDELIO>*. For more information, see Section 3.81. <AuthBy HOTSPOT> on page 320 and Section 3.82. <AuthBy FIDELIO> on page 325. *<AuthBy HOTSPOTFIDELIO>* supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

For a configuration sample, see files `goodies/README.hotspot-fidelio` and `goodies/hotspot-fidelio.cfg`. The sample configuration provides also an example of dynamically changing user sessions when they exceed their parameters, such as prepaid quota limits.

Figure 3. AuthBy HOTSPOTFIDELIO with dynamic authorization



3.83.1. BlockDuration

Specifies a number of seconds a prepaid block of time will last for, from the time it is first purchased. Defaults to 86400 (1 day).

3.83.2. BlockPrice

Specifies a number of cents a prepaid block of time costs. Defaults to 900 cents ($9.00). This cost will be posted to Opera whenever a new block is purchased.

3.83.3. PostSendQuery

This optional string defines the exact form of the insert query that is used to insert posting records.

3.83.4. PostSendQueryParam

This optional string array enables the use of SQL bind variables with *PostSendQuery*. Set *PostSendQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *PostSendQueryParam*s. Here is an example of using the SQL bind variables:

```
PostSendQuery INSERT INTO posts (roomNumber, guestNumber, macAddress, postNumber,
      posted, cost) VALUES (?, ?, ?, ?, ?, ?)
PostSendQueryParam %0
PostSendQueryParam %1
PostSendQueryParam %2
PostSendQueryParam %3
PostSendQueryParam %4
PostSendQueryParam %5
```

The same query without the SQL bind variables looks this:

```
PostSendQuery INSERT INTO posts (roomNumber, guestNumber, macAddress, postNumber,
      posted, cost) VALUES (%0, %1, %2, %3, %4, %5)
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.83.5. PostAnswerQuery

This optional string defines the exact form of the insert query that is used to insert Posting Answer (PA) data.

3.83.6. PostAnswerQueryParam

This optional string array enables the use of SQL bind variables with *PostAnswerQuery*. Set *PostAnswerQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *PostAnswerQueryParam*s. Here is an example of using the SQL bind variables:

```
PostAnswerQuery INSERT INTO postacks (roomNumber, postNumber, transactionNumber, received)
      values (?, ?, ?, ?)
PostAnswerQueryParam %0
PostAnswerQueryParam %1
PostAnswerQueryParam %2
PostAnswerQueryParam %3
```

The same query without the SQL bind variables looks this:

```
PostAnswerQuery INSERT INTO postacks (roomNumber, postNumber, transactionNumber, received)
      values (%0, %1, %2, %3)
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84. <AuthBy FIDELIOHOTSPOT>

This extension of *<AuthBy FIDELIO>* provides an easy way to integrate WiFi hotspots and captive portals with Opera hotel Property Management System from Micros Fidelio [http://www.micros-fidelio.eu/].

It uses an SQL database to keep track of prepaid access time blocks for Opera guests. It authenticates new sessions with the guest's room number and guest number (as with *<AuthBy FIDELIO>*), creates new prepaid blocks (and posts the charge to Opera), and keeps track of the remaining time left in each prepaid block. Time-limited prepaid plans are also supported.

<AuthBy FIDELIOHOTSPOT> understands also the same parameters as *<AuthBy SQL>* and *<AuthBy FIDELIO>*. For more information, see Section 3.41. <AuthBy SQL> on page 190 and Section 3.82. <AuthBy FIDELIO> on page 325. *<AuthBy FIDELIOHOTSPOT>* supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

For a configuration sample, see a configuration sample file `goodies/fidelio-hotspot.cfg`

Migrating to AuthBy HOTSPOTFIDELIO

This module is superseded by *<AuthBy HOTSPOTFIDELIO>* starting with Radiator 4.22. See Section 3.83. <AuthBy HOTSPOTFIDELIO> on page 330 for more information and sample configuration.

To migrate your current configuration, see configuration samples for *<AuthBy HOTSPOTFIDELIO>* and the notes below:

- ConfirmationQuery and ConfirmationQueryParam configuration parameters are deprecated
- Change ConfirmUpgradeOrRenew configuration parameter to ConfirmSubscription
- GetServiceQuery configuration parameter has been deprecated. Configure to use ServiceDatabase instead
- GetCurrentServiceQuery configuration parameter has been deprecated. Configure to use ServiceDatabase instead
- AddSessionQuery configuration parameter has been deprecated. Configure to use ServiceDatabase and SessionDatabase instead
- GetSessionQuery configuration parameter has been deprecated. Configure to use ServiceDatabase and SessionDatabase instead
- UpdateSessionQuery configuration parameter has been deprecated. Configure to use ServiceDatabase and SessionDatabase instead

3.84.1. BlockDuration

Specifies a number of seconds a prepaid block of time will last for, from the time it is first purchased. Defaults to 86400 (1 day).

3.84.2. BlockPrice

Specifies a number of cents a prepaid block of time costs. Defaults to 900 cents ($9.00). This cost will be posted to Opera whenever a new block is purchased.

3.84.3. ServiceAttribute

This string specifies the RADIUS attribute that is used to select the desires prepaid service or plan.

You can create a menu into Mikrotik login page, see an example:

```
<tr><td>Service:</td><td>
<select name="radius0-9048">
<option value="Mikrotik-Service=free">best effort (free)</option>
<option value="Mikrotik-Service=premium">premium ($5)</option>
</select></td></tr>
```

`name="radius0-9048"` is *OSC-AVPAIR*, this is listed in Radiator dictionary. Use the following parameter setting in Radiator configuration file:

```
ServiceAttribute OSC-AVPAIR
```

3.84.4. ServiceAttributePrefix

This optional string specifies the prefix in *ServiceAttribute*. For more information, see Section 3.84.3. ServiceAttribute on page 332. Use this parameter if there is a possibility to have several *ServiceAttribute* instances and they must be separated to be able to choose the correct one. Here is an example about using *ServiceAttributePrefix*:

```
ServiceAttributePrefix Mikrotik-Service=
```

3.84.5. ConfirmUpgradeOrRenew

This flag defines if the system asks a confirmation for an upgrade or renewal charge. This is disabled by default, and the upgrade or renewal of the current payment plan is processed automatically.

3.84.6. ConfirmationMessage

This string specifies the message that is shown to the guest when asking a confirmation for upgrade or renewal recharge. To show the message, include the following example into Mikrotik login page:

```
$(if error)<br /><div style="color: #FF8080; font-size: 14px">$(error)</div><br>$(endif)
```

Then add the *ConfirmationMessage* to your Radiator configuration:

```
ConfirmationMessage "You are going to upgrade or renew your plan, please login again
      to confirm the charge"
```

3.84.7. ConfirmationQuery

This string is an optional parameter. It defines the exact form of the query that is used to add the upgrade or renewal confirmation record to the database.

3.84.8. ConfirmationQueryParam

This optional string array enables the use of SQL bind variables with *ConfirmationQuery*. Set *ConfirmationQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *ConfirmationQueryParam*s. Here is an example of using the SQL bind variables:

```
ConfirmationQuery UPDATE sessions SET confirmation_requested=1 WHERE roomNumber=?
      AND guestNumber=? AND macAddress=?
ConfirmationQueryParam %0
ConfirmationQueryParam %1
ConfirmationQueryParam %2
```

The same query without the SQL bind variables looks this:

```
ConfirmationQuery UPDATE sessions SET confirmation_requested=1 WHERE roomNumber=%0
      AND guestNumber=%1 AND macAddress=%2
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.9. PostSendQuery

This optional string defines the exact form of the insert query that is used to insert posting records.

3.84.10. PostSendQueryParam

This optional string array enables the use of SQL bind variables with *PostSendQuery*. Set *PostSendQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *PostSendQueryParam*s. Here is an example of using the SQL bind variables:

```
PostSendQuery INSERT INTO posts (roomNumber, guestNumber, macAddress, postNumber,
      posted, cost) VALUES (?, ?, ?, ?, ?, ?)
PostSendQueryParam %0
PostSendQueryParam %1
PostSendQueryParam %2
PostSendQueryParam %3
PostSendQueryParam %4
PostSendQueryParam %5
```

The same query without the SQL bind variables looks this:

```
PostSendQuery INSERT INTO posts (roomNumber, guestNumber, macAddress, postNumber,
      posted, cost) VALUES (%0, %1, %2, %3, %4, %5)
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.11. PostAnswerQuery

This optional string defines the exact form of the insert query that is used to insert Posting Answer (PA) data.

3.84.12. PostAnswerQueryParam

This optional string array enables the use of SQL bind variables with *PostAnswerQuery*. Set *PostAnswerQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *PostAnswerQueryParam*s. Here is an example of using the SQL bind variables:

```
PostAnswerQuery INSERT INTO postacks (roomNumber, postNumber, transactionNumber, received)
      values (?, ?, ?, ?)
PostAnswerQueryParam %0
PostAnswerQueryParam %1
PostAnswerQueryParam %2
PostAnswerQueryParam %3
```

The same query without the SQL bind variables looks this:

```
PostAnswerQuery INSERT INTO postacks (roomNumber, postNumber, transactionNumber, received)
      values (%0, %1, %2, %3)
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.13. GetServiceQuery

This optional string defines the exact form of the query that is used to fetch information, such as price, duration, or reply attributes, about the service. **%0** denotes the value of a RADIUS attribute denoted by *ServiceAttribute*.

3.84.14. GetServiceQueryParam

This optional string array enables the use of SQL bind variables with *GetServiceQuery*. Set *GetServiceQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *GetServiceQueryParam*s. Here is an example of using the SQL bind variables:

```
GetServiceQuery SELECT price, duration, replyattributes FROM services WHERE serviceclass=?
GetServiceQueryParam %0
```

The same query without the SQL bind variables looks this:

```
GetServiceQuery SELECT price, duration, replyattributes FROM services WHERE serviceclass=%0
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.15. GetCurrentServiceQuery

This optional string defines the exact form of the query that is used to fetch information about the prepaid service and the user's current session.

3.84.16. GetCurrentServiceQueryParam

This optional string array enables the use of SQL bind variables with *GetCurrentServiceQuery*. Set *GetCurrentServiceQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *GetCurrentServiceQueryParam*s. Here is an example of using the SQL bind variables:

```
GetCurrentServiceQuery SELECT expiry,replyattributes,price,sessions.serviceclass,confirmation_re
      FROM sessions LEFT JOIN services ON \ sessions.serviceclass=services.serviceclass
      WHERE roomNumber=? AND guestNumber=? AND macAddress=?
GetCurrentServiceQueryParam %0
GetCurrentServiceQueryParam %1
GetCurrentServiceQueryParam %2
```

The same query without the SQL bind variables looks this:

```
GetCurrentServiceQuery SELECT expiry,replyattributes,price,sessions.serviceclass,confirmation_re
      FROM sessions LEFT JOIN services ON \ sessions.serviceclass=services.serviceclass
      WHERE roomNumber=%0 AND guestNumber=%1 AND macAddress=%2
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.17. AddSessionQuery

This optional string defines the exact form of the query that is used to add information about the current prepaid or postpaid session.

3.84.18. AddSessionQueryParam

This optional string array enables the use of SQL bind variables with *AddSessionQuery*. Set *AddSessionQuery* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *AddSessionQueryParam*s. Here is an example of using the SQL bind variables:

```
AddSessionQuery INSERT INTO sessions (roomNumber, guestNumber, macAddress, serviceclass,
```

```
      expiry) VALUES (?, ?, ?, ?, ?)
AddSessionQueryParam %0
AddSessionQueryParam %1
AddSessionQueryParam %2
AddSessionQueryParam %3
AddSessionQueryParam %4
```

The same query without the SQL bind variables looks this:

```
AddSessionQuery INSERT INTO sessions (roomNumber, guestNumber, macAddress, serviceclass,
      expiry) VALUES (%0, %1, %2, %3, %4)
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.19. GetSessionQuery

This optional string defines the exact form of the query that is used to fetch information about the current postpaid session.

3.84.20. GetSessionQueryParam

This optional string array enables the use of SQL bind variables with *`GetSessionQuery`*. Set *`GetSessionQuery`* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *`GetSessionQueryParam`*s. Here is an example of using the SQL bind variables:

```
GetSessionQuery SELECT expiry FROM sessions WHERE roomNumber=? AND guestNumber=?
      AND macAddress=?
GetSessionQueryParam %0
GetSessionQueryParam %1
GetSessionQueryParam %2
```

The same query without the SQL bind variables looks this:

```
GetSessionQuery SELECT expiry FROM sessions WHERE roomNumber=%0 AND guestNumber=%1
      AND macAddress=%2
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.84.21. UpdateSessionQuery

This optional string defines the exact form of the query that is used to update information about the current prepaid or postpaid session.

3.84.22. UpdateSessionQueryParam

This optional string array enables the use of SQL bind variables with *`UpdateSessionQuery`*. Set *`UpdateSessionQuery`* using the SQL bind variables or without them. If you use the SQL bind variables, you must specify all query parameters with respective *`UpdateSessionQueryParam`*s. Here is an example of using the SQL bind variables:

```
UpdateSessionQuery UPDATE sessions SET serviceclass=?, expiry=?, confirmation_requested=0
      WHERE roomNumber=? AND guestNumber=? AND macAddress=?
UpdateSessionQueryParam %3
UpdateSessionQueryParam %4
UpdateSessionQueryParam %0
```

```
UpdateSessionQueryParam %1
UpdateSessionQueryParam %2
```

The same query without the SQL bind variables looks this:

```
UpdateSessionQuery UPDATE sessions SET serviceclass=%3, expiry=%4,confirmation_requested=0
      WHERE roomNumber=%0 AND guestNumber=%1 AND macAddress=%2
```

For more information about SQL bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.85. <AuthBy PRESENCESQL>

<AuthBy PRESENCESQL> authenticates users from an SQL database and records presence (current user location) to an SQL database. Implements a special form of RADIUS Access-Request that allows user presence data to be retrieved by suitably authorised devices.

This module can be used with telephone and VOIP systems to automatically route telephone calls according to the user's current location.

A sample configuration file and documentation can be found in `goodies/presencesql.cfg` in your distribution.

3.86. <AuthBy HANDLER>

This clause allows requests to be redirected to a Handler based on the Handler's Identifier. This allows rapid selection of a Handler from amongst many Handlers using a hash-based match, instead of relying on the normal linear search that is used to find <Handler> clauses. This can be useful in special cases where there are many Handlers, and where a PreHandlerHook is able to identify the correct Handler to use.

When <AuthBy HANDLER> runs, it uses the HandlerId to create the name of the desired Handler to use. The named Handler is found based on its Identifier, and the request is redirected to that Handler.

3.86.1. HandlerId

This optional parameter specifies how to derive the Identifier of the Handler to use to handle requests. When a request is received by AuthBy HANDLER, this string will be used to derive a Handler Identifier. If a Handler with that Identifier is found, the request will be redespatched to that Handler. Special characters are supported. Defaults to ”handler%{ Request:Called-Station-Id}”.

3.87. <AuthBy WIMAX>

This clause handles requests from a WiMAX system. It handles, authentication, accounting and DHCP server key distribution. It acts as a Home AAA (HAAA) as per WiMAX End-to-End Network Systems Architecture Stage 2-3 Release 1.1.0 and NWG_R1.1.0-Stage-3.pdf. Answers requests from NAS, HA and DHCP servers.

AuthBy WIMAX requires an SQL database to hold user details (including password), and to cache various keys. WiMAX users must be added to the SQL database before they can be authenticated.

Several types of reply attribute are handled specially by AuthBy WIMAX. If the attribute is present in a reply (perhaps from a user's reply attributes or profile), they will be converted from text format to the binary format required by WiMAX devices.

- WiMAX-Packet-Flow-Descriptor

  A WiMAX packet flow descriptor. Example:

  ```
  Packet-Data-Flow-ID=01,Service-Data-Flow-ID=1,Direction=Bi-
  Directional,Transport-Type=IPv4-CS,Activation-Trigger="Activate",
  ```

```
Uplink-QoS-ID=1,Downlink-QoS-ID=2
```

- WiMAX-QoS-Descriptor

  A WiMAX QOS descriptor. Example:

  ```
  QoS-ID=1,Media-Flow-Type=Robust-Browser,Schedule-Type=BESTEFFORT,
  Traffic-Priority=0,Maximum-Sustained-Traffic-Rate=128000
  ```

Sample configuration file, including explanation of supported parameters is available in `goodies/wimax.cfg`. Sample database schema appears in `goodies/wimax.sql` in your Radiator distribution.

3.87.1. KeyLifetime

This optional parameter specifies the lifetime for all mobility keys in seconds. Defaults to 3600 (1 hour).

3.87.2. HAPassword

This optional parameter specifies the PAP password required for access by a WiMAX HA (Home Agent). If not defined, HA does not have to present a password before its requests are satisfied. If HAPassword is defined, the HA must present a PAP password with an exact match, and the HA must be configured to send this password, otherwise its requests will be REJECTed. Not all HAs are able to send a password with requests to the HAAA, so use of this parameter depends on your HA. Defaults to undefined.

3.87.3. ProfileHotlining

This optional parameter indicates whether to provide profile-based hotlining. If set, and the user has a Hotline Profile ID, the SQL database will be consulted for the Hotline profile, and the contents of the hotline profile id will be returned. Defaults to not set.

3.87.4. RulebasedHotlining

This optional parameter indicates whether to provide rule-based hotlining. If set, and the user has a Hotline Profile ID, the SQL database will be consulted for the Hotline profile, and the contents of the hotline NAS-Filter-Rule will be returned. Defaults to not set.

3.87.5. HTTPRedirectionHotlining

This optional parameter indicates whether to provide HTTP Redirection-based hotlining. If set, and the user has a Hotline Profile ID, the SQL database will be consulted for the Hotline profile, and the contents of the hotline HTTP-Redirection-Rule will be returned. Defaults to not set.

3.87.6. IPRedirectionHotlining

This optional parameter indicates whether to provide IP Redirection-based hotlining. If set, and the user has a Hotline Profile ID, the SQL database will be consulted for the Hotline profile, and the contents of the hotline IP-Redirection-Rule will be returned. Defaults to not set.

3.87.7. MSKInMPPEKeys

Forces the MSK to be encoded in MS-MPPE-Send-Key and MS-MPPE-Recv-Key, as well as the usual WiMAX-MSK reply attributes. This is required by some non-compliant clients, such as some Alcatel-Lucent devices.

3.87.8. GetCachedKeyQuery

SQL query to get the cached keys for a given AAA-Session-ID. Defaults to:

```
select sessionid, mip_rk, mip_spi, fa_rk from device_session
where sessionid=?
```

3.87.9. InsertSessionQuery

SQL query to get create a new session for a given AAA-Session-ID. Defaults to:

```
insert into device_session (outer_nai, sessionid, napid, bsid,
nspid, msid, capabilities, timezoneoffset, nai, cui, mip_rk,
mip_spi, fa_rk, key_expires) values (?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?)
```

3.87.10. UpdateSessionQuery

SQL query to get update a session for a given AAA-Session-ID. Defaults to:

```
update device_session set outer_nai=?, nai=?, cui=?, mip_rk=?,
mip_spi=?, fa_rk=?, key_expires=? where sessionid=?
```

3.87.11. GetHotlineProfileQuery

SQL query to get hotlining parameters. Defaults to:

```
select profileid, httpredirectionrule, ipredirectionrule
```

3.87.12. GetQosProfileQuery

SQL query to get QOS parameters. Defaults to:

```
select globalscname, scname, scheduletype, priority, maxsusrate,
minresrate, maxburst, jitter, maxlatency, reducedresources,
flowtype, grantinterval, sdusize, unsolpollinginterval
from qosprofile where id=?
```

3.88. <AuthBy SQLYUBIKEY>

Yubikeys are small USB tokens for one-time password authentication. They are produced by Yubico and are relatively inexpensive. Much of the surrounding technology and sample code is open source. They can support one factor or 2 factor authentication, but not challenge-response. For more information, see Yubico website [https://www.yubico.com/]

A Yubikey produces a unique one-time password each time the little button on it is pressed. The interesting thing is that the Yubikey acts like a USB keyboard, so the onetime- password is automatically typed into wherever the keyboard cursor is placed. This means that the user does not have to type in their token code by hand, as with many other tokens. It also means that it can work easily with many existing wired and wireless authentication methods, supplicants, web pages, Radius clients and other applications. The keys work on Windows, Linux, macOS and other platforms.

Each Yubikey has a secret AES key programmed into it in the factory. This secret key can be used to authenticate the key against any of the public services provided by Yubico. In order to use a Yubikey in a 3rd party authentication service such as Radiator, it is necessary to reprogram each token with a new (known) AES secret key, and to record that same key in the Radiator database. See `goodies/yubikey.txt` for more information about that process.

Yubico provide a number of sample open source programs for managing and authenticating Yubikeys. A typical one is the Yubico Java Server, a YubiKey OTP validation server in Java. This is a Tomcat application which provides an HTTP based web service API for authenticating Yubikeys. It looks up the secret key in a MySQL database and checks whether a Yubikey one-time password is correct.

Radiator now ships with the *<AuthBy SQLYUBIKEY>* module and a sample configuration file in `goodies/yubikey.cfg`. It supports RADIUS PAP, EAP-One-Time-Password and EAP-Generic-Token-Card protocols.

RAdmin from Radiator Software also supports Yubikeys, and provides an easy-to-use web-based tool for administering users and Yubikey tokens, including importing, allocating and deallocating tokens to users. 2 factor authentication is also supported in RAdmin. For more information, see RAdmin website [https://radiatorsoftware.com/products/radmin/].

The design of the *<AuthBy SQLYUBIKEY>* module allows it to be configured to work with a wide range of database schemas, but by default it works with the same schema that comes with the Yubico Java Server mentioned above. This means that you can provide web services API based authentication and Radius authentication for Yubikeys from the one token database.

<AuthBy SQLYUBIKEY> can be optionally configured to support or require 2 factor authentication if the token database also contains a static password for each user. In this case, the user types their static password first, followed by a colon (':'), followed by the output from the one-time password token. The user is only authenticated if both the static password and the one-time password are correct. *<AuthBy SQLYUBIKEY>* supports replay attack detection: any attempt to use the same one-time password twice in a row will fail. For example, if the user's static password was 'fred', the resulting password would be something like:

```
fred:ccccccccefeiujbfhkhfurbitjcvuvnedivhbeighuvf
```

Radiator does not come with any tools for managing the Yubikey token database. Use RAdmin (for more information, see RAdmin [https://radiatorsoftware.com/products/radmin/]) or use one of the tools provided by Yubico. It is relatively easy to integrate this with almost any existing user database.

The example configuration file `goodies/yubikey.cfg` in the Radiator distribution shows the main configuration options. It will work by default with the database schema provided with Yubico Java Server, but can be customised for many other SQL database schemas. It assumes the token ID and secret are in the database in Hex (no spaces) format, and that there is a one-to-one mapping between Yubikeys and users. Other formats and multiple key or multiple user mappings can be supported with custom SQL query parameters.

Tip

Radiator can also authenticate Yubikeys against the Yubikey Validation Server. For more information, see Section 3.89. <AuthBy YUBIKEYVALIDATIONSERVER> on page 342. Another possibility is to use the Yubikey PAM module and AuthBy PAM.

<AuthBy SQLYUBIKEY> supports EAP-OTP and EAP_GTC for use with various EAP compatible devices.

<AuthBy SQLYUBIKEY> requires the `Auth::Yubikey_Decrypter` module version 0.05 or later from CPAN website [https://www.cpan.org/] and the `Crypt::Rijndael` module, also available from CPAN.

<AuthBy SQLYUBIKEY> supports the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.88.1. AuthSelect

Specifies an SQL query that will be used to fetch Yubikey data from the database. Special characters are permitted, and %0 is replaced with the quoted user name, the Token ID in Base64 format in %1, the Token ID in hex format in %2 and the Token ID in modhex format in %3. The result field 0 (secret) is the base64 encoded AES secret for the key. It must be present for the authentication to succeed. All others fields are optional. If field 1 (active) is defined is must be 1 else the authentication is rejected. Field 2 (userId) is not currently used. Field 3 (counter) is the key use counter. If defined, it will be used to detect replay attacks, and must be updated by UpdateQuery. Field 4 (session_use) is the session_use counter. Field 5 is currently ignored. The static password field (field 6) contains the users correct static password in any of the formats supported by Radiator including plaintext, {SHA}, {crypt}, {MD5}, {rcrypt}, {mysql}, {mssql}, {nthash}, {dechpwd}, {NS-MTA-MD5}, {clear} etc. TranslatePasswordHook is also supported.

The default works with the sample Yubikey database created by db_schema.sql from the YubiKey Validation Server. The default is:

```
AuthSelect select secret, active, userId, counter, low,
high,NULL from yubikeys where userId=%0
```

which assumes that there is a one-to-one mapping between Yubikeys and users. It also assumes the Token ID and AES secret are in Hex (no spaces). You could support multiple tokens per user or multiple user per token with a custom AuthSelect like:

```
AuthSelect select secret, active, userId, counter, low,
high,NULL from yubikeys where tokenId=%1 and userId=%0
```

3.88.2. UpdateQuery

Specifies SQL query that will be used to store Yubikey token data back to the database after authentication. Special characters are permitted, and %0 is replaced with the new session counter, %1 with the new session_use counter, %2 with 0 (this column not currently used), %3 with the quoted user name, %4 with the quoted token ID in Base64, %5 with the current time (seconds in the unix epoch), %6 with the token ID in Hex and %7 with the token ID in modhex. The default works with the sample Yubikey database created by db_schema.sql from the YubiKey Validation Server. The default is:

```
UpdateQuery update yubikeys set accessed=current_timestamp(),
counter=%0, low=%1, high=%2 where userId=%3
```

which assumes that there is a one-to-one mapping between Yubikeys and users. You could support multiple tokens per user or multiple user per token with a custom Update- Query like:

```
UpdateQuery update yubikeys set accessed=current_timestamp(),
counter=%0, low=%1, high=%2 where tokenId=%4 and userId=%3
```

3.88.3. Require2Factor

Forces all authentications to require 2 factor authentication. In 2 factor authentication, the user is required to enter (as their password) both their static password (also called their PIN) followed by the one-time-password from the Yubikey.

3.88.4. CheckSecretId

If CheckSecretId is set, then check that the secretId fetched from the database matches the secretId encoded in the submitted Yubikey OTP. This increases the security of the Yubikey OTP and is recommended best practice.

3.89. <AuthBy YUBIKEYVALIDATIONSERVER>

This module authenticates YubiKey tokens (yubico.com) against YubiCloud validation service or locally hosted YubiKey Validation Server. This allows flexibility in deciding which validation service or server to use and where to plug in a YubiHSM. This module does not require any YubiKey specific modules because all required work is done by the validation server and possibly by YubiHSM. PyHSM validation server allows using Radiator with YubiHSM (Hardware Security Module) for storing the YubiKey secrets.

Yubico's Validation Server (YK-VAL) and YubiCloud API versions 1.0 and 2.0 are supported. For YubiCloud you should set *`APIVersion`* to **`2.0`** and *`ClientID`* to the value assigned to you by Yubico. Configuring *`APIKey`* is optional but recommended especially when *`ValidationServerURL`* is set to http instead of https. If you run a self hosted YK-VAL, set *`APIVersion`*, *`ClientID`* and *`APIKey`* to match the server configuration.

Yubico's PyHSM validation server and its one line response format is also supported. Yubico's PyHSM validation server yhsm-val supports Yubico OTP, OATH-HOTP and OATH-TOTP.

See a sample configuration file `goodies/yubikey-validationserver.cfg` for two-factor, single factor and EAP configuration examples.

AuthBy YUBIKEYVALIDATIONSERVER understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.89.1. ValidationServerURL

The URL for Yubikey Validation server. OTP protocol specific part will be appended to the ValidationServerURL. Defaults to **`http://127.0.0.1:8003/yhsm/validate?`** which is compatible with Yubico's PyHSM validation server yhsm-val.

HTTPS requires LWP::Protocol::https Perl module. If it's not installed, Radiator will log an error without connecting to the server.

```
# Use Yubico's service over HTTPS
ValidationServerURL https://api.yubico.com/wsapi/2.0/verify?
```

3.89.2. OTPProtocol

This is a comma separated list of OTP protocols Radiator is allowed to support. The possible values are:

- **`YubicoOTP`**
- **`OATH-HOTP`**
- **`OATH-TOTP`**

OATH protocols **`OATH-HOTP`** and **`OATH-TOTP`** are only supported by PyHSM validation server. Default value is **`YubicoOTP`**.

```
# We use PyHSM validation server and can support HOTP too
OTPProtocol YubicoOTP, OATH-HOTP
```

3.89.3. APIVersion

This is a comma separated list of OTP protocols Radiator is allowed to support. Format specifiers, such as *`%{GlobalVar:name}`*, are evaluated when the configuration is loaded. The possible values are:

- **`yk-ksm`**
- **`1.0`**
- **`2.0`**

For YubiCloud you should set *APIVersion* to **2.0** and *ClientID* to the value assigned by Yubico. *APIVersion* defaults to **yk-ksm** which requires PyHSM validation server yk-val that runs with *--short-otp* parameter.

```
# YubiCloud supports version 2.0 API
APIVersion 2.0
```

3.89.4. ClientID

ClientID is required for signed requests and responses. Yubico allocates this for YubiCloud users and seems to always require it when using YubiCloud validation servers.

There is no default value.

```
# Value allocated to us by Yubico
ClientID 1
```

3.89.5. APIKey

APIKey is required for signing requests and responses. It is allocated for a *ClientID* and specified in Base64 format. When this parameter is non-empty, requests are signed and a valid signature is required in responses. Special formatting characters are allowed.

There is no default value.

```
# Value allocated to us by Yubico
APIKey t2ZMtKeValdA+H0jVpj3LIichn4=
```

3.89.6. OTPCharset

OTPCharset allows limiting and changing the characters allowed in OTPs. Defaults to **0-9cbdefghijklnrtuv** which allows OATH and Yubico OTPs. The value of the parameter is a Perl character set specification. See your Perl reference manual for details about how to construct Perl character set specifications. Note that the some special characters must be escaped with a backslash.

```
# Allow Yubico OTPs only
OTPCharset cbdefghijklnrtuv
```

Note

If you need to support non-standard keyboard layouts, such as Dvorak, you may need to change *OTPCharset*.

3.89.7. Timeout

Connection timeout in seconds. Defaults to 3.

3.89.8. SSLVerify

May be used to control how the Yubikey Validation Server's certificate will be verified. May be one of "none" or "require".

3.89.9. SSLCAPath

When verifying the XML Yubikey Validation Server's certificate, set this to the pathname of the directory containing CA certificates. These certificates must all be in PEM format. The directory in must contain certificates named using the hash value of the certificates' subject names.

3.89.10. SSLCAFile

Use this option to locate the file containing the certificates of the trusted certificate authorities. Thus, you can verify that the server certificate has been signed by a reputable certificate authority. Special characters are permitted.

Here is an example of using *SSLCAFile*:

```
SSLCAFile %D/certificates/demoCA/cacert.pem
```

3.90. <AuthBy SQLHOTP>

This module supports authentication using HOTP (RFC 4226) authentication. HOTP is an open specification for event-based one-time passwords, developed by OATH [https://openauthentication.org/].

HOTP is an event-based authentication protocol, and is designed for use in 2 factor tokens and other similar authentication processes. It uses the well-known SHA-1 hash function, along with a secret key and an incrementing counter. The specification is completely open and free and is the result of community collaboration with OATH.

The *<AuthBy SQLHOTP>* authentication module detects replay and brute-force attacks. It supports optional PIN, also known as static password, for 2 factor authentication when the user prefixes their static password before the HOTP one-time password.

The secret key, current counter and other information are stored in a SQL database. Any database supported by Radiator can be used. A sample configuration file and SQL schema for MySQL are supplied in the `goodies/` directory of your Radiator distribution.

<AuthBy SQLHOTP> supports the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.90.1. AuthSelect

AuthSelect is an SQL query that fetches HOTP token data from the SQL database. *AuthSelect* is expected to return a number of fields that describe the token.

The following fields are mandatory:

- Field 0 is the HEX encoded secret key for the token
- Field 1 is the counter high part
- Field 2 is the counter low part

The following fields are optional:

- If field 3 (active) is defined, it must be 1 or other true value, else the authentication is rejected. Empty and 0 are false.
- Field 4 (pin) is the user's static PIN. It will be checked if the user specifies a static password or if *Require2Factor* is not set to disabled.

- Field 5 (digits) is the number of digits in the user's HOTP code. If NULL, the value of *DefaultDigits* is be used.
- Field 6 (bad_logins) counts the number of consecutive authentication failures. If defined it will be used to detect brute force attacks and must be updated by UpdateQuery.
- Field 7 (last_time_accessed) is the unix timestamp of the last authentication attempt. It is used to detect brute force attacks.

Current username is available as **%0** which is SQL quoted when used in *AuthSelect* and unmodified when used with *AuthSelectParam*.

The default works with the sample database schema provided in `goodies/hotp.sql`. The default is:

```
select secret, counter_high, counter_low, active, pin, digits,
bad_logins, unix_timestamp(accessed) from hotpkeys where username=%0
```

3.90.2. AuthSelectParam

This optional parameter specifies a bind variable to be used with AuthSelect. See Section 3.8.1. SQL bind variables on page 47 for information about how to use bind variables.

3.90.3. UpdateQuery

UpdateQuery is an SQL query that updates the HOTP token data in the SQL database. After a successful or failed authentication it will be passed the new authentication counter high in **%0**, new authentication counter low in **%1**, bad login count in **%2**, the user name in **%3**, The default works with the sample database schema provided in `goodies/hotp.sql`.

%0 and the other formatters are SQL quoted, if needed, when used in *UpdateQuery* and unmodified when used with *UpdateQueryParam*.

The default *UpdateQuery* is:

```
update hotpkeys set accessed=now(), counter_high=%0, counter_low=%1,
bad_logins=%2 where username=%3
```

3.90.4. UpdateQueryParam

This optional parameter specifies a bind variable to be used with UpdateQuery. See Section 3.8.1. SQL bind variables on page 47 for information about how to use bind variables.

3.90.5. Require2Factor

If flag parameter *Require2Factor* is not set to disabled, then the user must provide their static password as a prefix to their one-time password. The correct static password is returned by *AuthSelect*. If the user provides a static password prefix, then the static password is always checked regardless of *Require2Factor* setting.

3.90.6. DefaultDigits

DefaultDigits specifies the number of one-time password digits to use if the user record does not define digits. Defaults to 6. Minimum allowed is 4.

3.90.7. MaxBadLogins

MaxBadLogins specifies how many consecutive bad PINs or bad OTP codes will be tolerated in the last *BadLoginWindow* seconds. If more than *MaxBadLogins* bad authentication attempts occurs and if the last one

is within the last *BadLoginWindow* seconds, the authentication attempt will be rejected. The user must wait at least *BadLoginWindow* seconds before attempting to authenticate again. *MaxBadLogins* defaults to 10.

Attempt counter and window information is maintained in SQL with *UpdateQuery* and *AuthSelect*.

3.90.8. BadLoginWindow

Period of time in seconds that the user will be locked out after *MaxBadLogins* have occurred.

3.90.9. ResyncWindow

ResyncWindow defines the maximum number of missing authentications that will be tolerated for counter resynchronisation. Defaults to 20.

3.91. <AuthBy SQLTOTP>

This module supports authentication using TOTP (RFC 6238) authentication. TOTP is an open specification for time-based one-time passwords, developed by OATH [https://openauthentication.org/]

TOTP is a time-based authentication protocol, and is designed for use in 2 factor tokens and other similar authentication processes. It uses well-known SHA-1, SHA-256 or SHA-512 hash functions, along with a secret key and a timestamp. The specification is completely open and free and is the result of community collaboration with OATH.

The *<AuthBy SQLTOTP>* authentication module detects replay and brute-force attacks. It supports optional PIN, also known as static password, for 2 factor authentication when the user prefixes their static password before the TOTP one-time password.

The secret key, PIN and other information are stored in a SQL database. Any database supported by Radiator can be used. A sample configuration file and SQL schema for MySQL are supplied in the goodies directory of your Radiator distribution.

Tip

Correct operation of time based authentication tokens such as TOTP requires accurate synchronisation of the clocks on the client and Radiator server computers.

<AuthBy SQLTOTP> supports by default PAP, EAP-TOP and EAP-GTC. CHAP, MSCHAP and MSCHAPv2 are supported but need to be enabled with AuthenProto on page 169 configuration parameter. EAP-MSCHAP-V2 is supported as MSCHAPv2 when ConvertedFromEAPMSCHAPV2 on page 70 is enabled. The CHAP methods do not support detection of bad PIN values.

Static passwords can be stored in an encrypted format when PAP, EAP-OTP or EAP-GTC is used. For more information about encrypted formats, see Section 7.1.2. Encrypted-Password on page 466

<AuthBy SQLTOTP> supports the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.91.1. AuthSelect

AuthSelect is an SQL query that fetches TOTP token data from the SQL database. *AuthSelect* is expected to return a number of fields that describe the token.

The following fields are mandatory:

- Field 0 is the HEX encoded secret key for the token

The following fields are optional:

- If field 1 (active) is defined, it must be 1 or other true value, else the authentication is rejected. Empty and 0 are false.
- Field 2 (pin) is the user's static PIN It will be checked if the user specifies a static password or if *`Require2Factor`* is not set to disabled.
- Field 3 (digits) is the number of digits in the user's TOTP code. If NULL, the value of *`DefaultDigits`* is be used.
- Field 4 (bad_logins) counts the number of consecutive authentication failures. If defined it will be used to detect brute force attacks and must be updated by UpdateQuery.
- Field 5 (last_time_accessed) is the unix timestamp of the last authentication attempt. It is used to detect brute force attacks.
- Field 6 is the last TOTP timestep validated, which should be updated automatically by *`UpdateQuery`*.
- Optional field 7 (algorithm) is the SHA algorithm which defaults to SHA-1 if the value is NULL or empty or has an unknown value. Possible values are **`SHA1`**, **`SHA256`** and **`SHA512`**.
- Optional field 8 (timestep) is the user's time step which defaults to the *`TimeStep`* configuration parameter if the value is 0 or NULL.
- Optional field 9 (timestep_ origin) is the Unix epoch time of the first time step which defaults to *`TimeStepOrigin`* configuration parameter if the value is NULL.

Current username is available as **`%0`** which is SQL quoted when used in *`AuthSelect`* and unmodified when used with *`AuthSelectParam`*.

The default works with the sample database schema provided in `goodies/totp.sql`. The default is:

```
select secret, active, pin, digits, bad_logins, unix_timestamp(accessed),
last_timestep from totpkeys where username=%0
```

3.91.2. AuthSelectParam

This optional parameter specifies a bind variable to be used with AuthSelect. See Section 3.8.1. SQL bind variables on page 47 for information about how to use bind variables.

3.91.3. UpdateQuery

`UpdateQuery` is an SQL query that updates the TOTP token data in the SQL database. After a successful or failed authentication it will be passed the bad login count in **`%0`**, the user name in **`%1`** and last TOTP timestep in **`%2`**. The default works with the sample database schema provided in `goodies/totp.sql`. The default is:

```
update totpkeys set accessed=now(), bad_logins=%0, last_timestamp=%2
where username=%1
```

3.91.4. UpdateQueryParam

This optional parameter specifies a bind variable to be used with UpdateQuery. See Section 3.8.1. SQL bind variables on page 47 for information about how to use bind variables.

3.91.5. Require2Factor

If flag parameter *Require2Factor* is not set to disabled, then the user must provide their static password as a prefix to their one-time password. The correct static password is returned by *AuthSelect*. If the user provides a static password prefix, then the static password is always checked regardless of *Require2Factor* setting.

3.91.6. EncryptedPIN

This parameter must be set if PIN, also known as static password, is stored in one of the encrypted formats Radiator supports. Encrypted PIN works only with PAP, EAP-OTP and EAP-GTC because these allow Radiator to check the PIN and TOTP code separately.

For more information about encrypted formats, see Section 7.1.2. Encrypted-Password on page 466.

```
# We use PAP, EAP-OTP or EAP-GTC and can use encrypted PIN
EncryptedPIN
```

3.91.7. DefaultDigits

DefaultDigits specifies the number of one-time password digits to use if the user record does not define digits. Defaults to 6. Minimum allowed is 4.

3.91.8. MaxBadLogins

MaxBadLogins specifies how many consecutive bad PINs or bad OTP codes will be tolerated in the last *BadLoginWindow* seconds. If more than *MaxBadLogins* bad authentication attempts occurs and if the last one is within the last *BadLoginWindow* seconds, the authentication attempt will be rejected. The user must wait at least *BadLoginWindow* seconds before attempting to authenticate again. *MaxBadLogins* defaults to 10.

Attempt counter and window information is maintained in SQL with *UpdateQuery* and *AuthSelect*.

3.91.9. BadLoginWindow

Period of time in seconds that the user will be locked out after *MaxBadLogins* have occurred.

3.91.10. DelayWindow

DelayWindow is the maximum number of timeslots transmission delay that can be permitted between the client and server. Defaults to 1, the value recommended by the TOTP specification.

3.91.11. TimeStep

TimeStep is the size of the time step in seconds to use if the user record does not define time step. Defaults to 30 seconds, the value recommended by the TOTP specification.

3.91.12. TimeStepOrigin

TimeStepOrigin the Unix epoch time of the first time step to use if the user record does not define the origin. Defaults to 0 seconds (Jan 1, 1970), the value recommended by the TOTP specification.

3.92. <AuthBy SQLAUTHBY>

This clause does an SQL lookup for each incoming request to determine the actual Radiator AuthBy Clause to use to handle the request. If the resulting Radiator clause has not previously been used, it is created using parameters retrieved from SQL and the request is given to it for authentication or other handling. If the resulting Radiator clause has been used previously, it is reused and the request is sent to it.

Any database supported by Radiator can be used. A sample configuration file is supplied in the `goodies/` directory of your Radiator distribution.

<AuthBy SQLAUTHBY> supports the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.92.1. AuthBySelect

AuthBySelect is the SQL query used to retrieve AuthBy clause configuration parameters based on the user's realm. The clause is cached for reuse. You can use *AuthBySelectParam* to bind variables to the *AuthBySelect* query. **%0** is replaced with SQL quoted realm when no *AuthBySelectParam*s are configured.

The default value is:

```
AuthBySelect select HOST, PORT, AUTHDN, AUTHPASSWORD, BASEDN, USERNAMEATTR, \
     PASSWORDATTR, HOLDSERVERCONNECTION from RADSQLAUTHBY where TARGETNAME=%0
```

This is suitable for an AuthBy LDAP2 clause.

3.92.2. AuthBySelectParam

This optional parameter specifies a bind variable to be used with AuthBySelect. Unquoted realm is available as **%0**. For more information, see Section 3.8.1. SQL bind variables on page 47.

Here is an example of using *AuthBySelectParam*:

```
AuthBySelect select HOST, PORT, AUTHDN, AUTHPASSWORD, BASEDN, USERNAMEATTR, \
       PASSWORDATTR, HOLDSERVERCONNECTION from RADSQLAUTHBY where TARGETNAME=?
AuthBySelectParam %0
```

3.92.3. Class

This parameter defines the type of Radiator AuthBy clause to create. The class name is the type of AuthBy clause to create. For example

```
Class LDAP2
```

will cause <AuthBy SQLAYTHBY> to create clauses of type <AuthBy LDAP2>. Defaults to ‘LDAP2’.

3.92.4. DefaultParam

DefaultParam sets a default value for any clause parameter, which may be overridden with a column from the SQL query using ParamColumnDef.

```
DefaultParam RewriteUsername s/^([^@]+).*/$1/
DefaultParam Version 3
```

3.92.5. ParamColumnDef

Maps the columns returned by AuthBySelect to parameter names in the target clause. If non-NULL, each will override the corresponding default value given by DefaultParam.

```
ParamColumnDef 0,Host
ParamColumnDef 1,Port
```

```
ParamColumnDef 2,AuthDN
ParamColumnDef 3,AuthPassword
```

3.93. <AuthBy HEIMDALDIGEST>

This clause authenticates users against Heimdal Kerberos, using the kdigest program part of Heimdal Kerberos. For more information, see Heimdal website [http://www.h5l.org/].

Works with RADIUS-PAP, EAP-MD5, EAP-MSCHAPV2 (and therefore TTLS-PAP, TTLS-EAP-MD5, PEAP-EAP-MD5, PEAP-EAP-MSCHAPV2, TTLS-EAPMSCHAPV2).

Other types of authentication cannot be supported by Heimdal for technical reasons.

AuthBy KRB5 for a module that can work with any Kerberos, but is limited to PAP and TTLS-PAP. For more information, see Section 3.69. <AuthBy KRB5> on page 286

See `goodies/heimdaldigest.cfg` for an example configuration file. See `goodies/heimdal.txt` for guidance and help in setting up a simple Heimdal system for testing.

<AuthBy HEIMDALDIGEST> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.93.1. KdigestPath

The path to the executable Heimdal kdigest program. This program will be run externally by AuthBy HEIMDALDIGEST to authenticate each password.

Defaults to /usr/libexec/kdigest.

```
KdigestPath /usr/heimdal/lib/kdigest
```

3.93.2. KdigestSuffix

String that will be added to the end of each user name before authenticating with kdigest. Defaults to empty string. See also default_realm in `krb5.conf`, which will be used if user name does not contain a Kerberos realm.

```
KdigestSuffix @MYCOMPANY.COM
```

3.93.3. KdigestRealm

String specifying the Kerberos realm that will be used to authenticate each user. Used to specify --kerberos-realm= to kdigest. Defaults to undefined.

```
KdigestRealm OPEN.COM.AU
```

3.94. <AuthBy SIP2>

This clause authenticates users with 3M Standard Interchange Protocol 2 as used in 3Ms Automated Circulation Systems (ACS) for book libraries. AuthBy SIP2 supports TCPIP connection to 3M ACS systems, and authenticates against library patron name and password.

Works with EAP-GTC and PAP and therefore with e.g. EAP-TTLS/PAP. See goodies/ sip2.cfg for an example configuration file.

Tip

SIP2 is not related to Session Initiation Protocol (SIP) used for Voice Over IP and other multimedia applications.

<AuthBy SIP2> understands also the same parameters as <AuthBy xxxxxx>. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.94.1. Port

Port specifies the TCP port name or number of the ACS server. Defaults to 6001.

3.94.2. Host

Host specifies the name or address of the ACS server. Defaults to localhost.

3.94.3. Delimiter

The field delimiter ACS server uses. Defaults to "|".

3.94.4. Timeout

Timeout interval in seconds that Radiator will wait for when trying to contact the SIP2 server. Defaults to 3.

3.94.5. Retries

This is an integer that specifies the number of times Radiator tries to send a request and wait for the reply from the SIP2 server before disconnecting and initiating possible failure backoff. The default value is **`3`**.

If the SIP2 server returns its desired reply count and this reply count is smaller than the Retries value, the server's desired value is used.

3.94.6. FailureBackoffTime

This is an optional integer that specifies for how long time a failed SIP2 server is marked as failed. If no reply is received from the requested SIP2 server after all retries are used, the server is marked as failed. It is not connected again until *`FailureBackoffTime`* has been expired. The unit is seconds and the default value is **`0`**, which means that the server is never marked as failed and always expected to be working.

3.94.7. LoginUserID

User ID that Radiator will use to log into the ACS server. If this is defined as an empty string, then the login phase will not be performed. You need to be sure that this matches what the SIP2 server expects from clients. Many servers do not require a login phase. Defaults to **`scclient`**.

```
# LoginUserID is empty to skip the login phase
LoginUserID
```

3.94.8. LoginPassword

Password that Radiator will use to log into the ACS server. Defaults to **`clientpwd`**.

```
LoginPassword clientpwd
```

3.94.9. LocationCode

Location code that Radiator will use to log into the ACS server. Defaults to ‘Radiator’.

3.94.10. TerminalPassword

Terminal Password that Radiator will use to log into the ACS server. Not all installations require this.

```
TerminalPassword terminal password
```

3.94.11. Institution

The library's institution ID that Radiator will use Patron Information and Status Request messages. Defaults to not set. When set, overrides the value learnt from ACS Status message.

```
Institution UWOLS
```

3.94.12. SendChecksum

This optional flag tells Radiator to send checksums in every request sent to ACS. This must agree with the configuration of the ACS and defaults to off.

```
# This ACS requires sending checksums
SendChecksum
```

3.94.13. VerifyChecksum

Tells Radiator to verify checksums sent by ACS are present and correct. This must agree with the configuration of the ACS and defaults to off.

```
# We want to verify the checksums this ACS sends
VerifyChecksum
```

3.94.14. SIP2Hook

Perl hook that is run for each request handled by SIP2. This hook can be used for further authorisation checks. The hook is passed the following arguments:

- Reference to the current AuthBy SIP2 object
- Response received from the ACS
- Reference to the current request

The hook return value must be one of:

```
$main::ACCEPT
$main::REJECT
$main::IGNORE
$main::CHALLENGE
$main::REJECT_IMMEDIATE
```

Note

When the optional parameter NoCheckPassword is enabled, the hook return value determines the authentication result. See `goodies/sip2hook.pl` for an example.

```
SIP2Hook file:"%D/sip2hook.pl"
```

3.95. <AuthBy DUO>

This clause authenticates users against two-factor authentication provided by Duo Security Auth API. *<AuthBy DUO>* requires Perl modules `HTTP::Async` 0.19 or later and `Net::HTTPS::NB`. For more information, see Duo website [https://duo.com/]

To get started you need to sign up for a Duo account and create a new Auth API integration. Duo Security will then generate your API Hostname, integration key and secret key which are used to set up Radiator configuration.

See `goodies/duo.cfg` for an example configuration file and examples of how to combine the password and password factor (see Section 3.95.2. DefaultFactor on page 353) for controlling how two-factor authentication is done. See `goodies/duosim.cgi` for setting up a partial authentication API simulator for testing.

<AuthBy DUO> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.95.1. Hostname, SecretKey and IntegrationKey

API Hostname, secret key, and integration key are assigned by Duo. These are mandatory parameters and there are no default values. SecretKey and IntegrationKey support special formatting characters. The formatting is done once during the module activation.

```
Hostname api-aabbcczz.duosecurity.com
SecretKey aaaabbbbccccddddeeeeffffgggghhhhiiiijjjj
IntegrationKey kkkkllllmmmmnnnnoooo
```

3.95.2. DefaultFactor

If the user does not specify a valid password or factor, this will be the factor requested from Duo. May be one of "push", "sms", "phone", "auto". Defaults to "auto".

```
DefaultFactor push
```

3.95.3. PreAuth

This optional flag specifies whether a pre-authentication is done for the user. Default is to do pre-authentication.

Duo security Auth API `/preauth` endpoint determines if the user is authorised to log in and returns the available authentication factors for the authorised user.

```
# Turn off pre-authentication
PreAuth no
```

3.95.4. Address

Address specifies what details in the incoming request will be used in the 'Address' field sent to Duo. The contents of the field will show up in the Duo logs. It does not actually have to be an address (which in any case is not usually available until after authentication is complete), and the default of Calling-Station-Id might be a sensible option.

```
Address %{Calling-Station-Id}
```

3.95.5. SSLVerify, SSLCAFile and SSLCAPath

The SSL* parameters allow controlling how the Duo server's certificate is verified. By default no SSL* parameters are set and the system defaults are used. SSLVerify, SSLCAFile and SSLCAPath behave as described in <AuthBy IMAP>. For more information, see Section 3.59. <AuthBy IMAP> on page 258.

3.95.6. SSLVerifyCNName and SSLVerifyCNScheme

SSLVerifyCNName sets the name which is used when verifying the hostname against the certificate presented by Duo's server. SSLVerifyCNScheme controls how the verification is done, for example, if wildcards are allowed. For more information, see IO::Socket::SSL [https://metacpan.org/pod/IO::Socket::SSL].

The following allow wildcard certificate name *.duosecurity.com to match.

```
SSLVerifyCNName duosecurity.com
SSLVerifyCNScheme http
```

3.95.7. SSLCertificateVerifyHook

Further certificate checks can be added with a custom hook. For more information, see SSL_verifycn_name [https://metacpan.org/pod/IO::Socket::SSL#SSL_verify_callback]. The hook should return 0 or 1 for fail and pass respectively. The hook is called once for each certificate in the certificate chain.

The following arguments are passed to the hook:

- $_[0] is a true or false value indicating what OpenSSL's view of the cert.
- $_[1] is a C-style memory address of the certificate store.
- $_[2] is a string containing certificate's issuer and owner.
- $_[3] is a string containing any errors encountered or 0 if there were no errors.
- $_[4] is a C-style memory address of current certificate in the chain.

3.95.8. PollTimerInterval

Number of seconds between checking for replies from the Duo Auth API server. Defaults to 1 second.

3.95.9. CheckTimerInterval

Number of seconds between checking that the Duo Auth API server is alive and responding. Defaults to **5** seconds. When set to **0**, checks are disabled and FailureBackoffTime on page 354 is used to control how long API is considered dead.

```
# Use one minute alive check interval
CheckTimerInterval 60
```

3.95.10. FailureBackoffTime

Number of seconds to consider the Duo Auth API server as dead after an API failure. This parameter is only used when *CheckTimerInterval* on page 354 is set to **0** to turn off periodic API checks. When *FailureBackoffTime* is set to **0**, API is always considered alive. Defaults to **60** seconds.

```
# Do not use alive polling. Allow API to remain in failed state for two minutes
```

```
CheckTimerInterval 0
FailureBackoffTime 120
```

3.95.11. Slots

Specifies the maximum number of simultaneous requests outstanding to the DUO Auth API server, and the maximum number of HTTP connections to the server. If more than this number of requests are waiting, then subsequent requests will be queued and sent after a reply a received from an outstanding request. Defaults to **`20`**.

If the total number of requests that are waiting and queued exceeds *`2 * Slots`*, a warning is logged and the API is marked dead to prevent further overload.

3.95.12. Timeout

Specifies the maximum number of seconds to wait for the start of a reply from the Auth API server. The Auth API server can take up to 60 seconds to reply. Default is 100 seconds. You should not need to change this.

3.95.13. MaxRequestTime

Specifies the maximum number of seconds to wait for a complete reply from the Auth API server. The Auth API server can take up to 60 seconds to reply. Default is 120 seconds. You should not need to change this.

3.95.14. Failmode

This optional parameter specifies whether to reject, accept, or ignore authentication when a Duo API problem occurs. Possible problems are: API call fails, API response returns an error, API response can not be parsed or API call times out. Default is to **`ignore`**. Format specifiers, such as *`%{GlobalVar:name}`*, are evaluated when the configuration is loaded. The possible values are:

- **`accept`**
- **`ignore`**
- **`reject`**

```
# Reject, instead of ignore, when timeout or other API problem occurs
Failmode reject
```

3.95.15. ProxyHost

Specifies the name of a HTTP proxy to use to contact the Auth API server. The default is not to use any proxy.

3.95.16. ProxyPort

Specifies the port on the ProxyHost to use to contact the Auth API server.

3.95.17. EndpointPrefix

The prefix for the Auth API. Defaults to `/auth/v2`. You should not need to change this. See `goodies/duo.cfg` for how to set EndpointPrefix when using the Auth API simulator `duosim.cgi`.

3.95.18. Protocol

The protocol to use to connect to the Auth API server. Defaults to **`https`**. Usually there is no need to change this.

3.96. <AuthBy DIAMETER>

<AuthBy DIAMETER> converts and forwards all RADIUS authentication and accounting requests to a DIAMETER server. The DIAMETER replies are converted back to RADIUS responses and returned to the requesting client which might be a remote client or this Radiator instance itself. The RADIUS requests may originate from a RADIUS client or they may be converted from TACACS+ or DIAMETER requests.

Diameter peering can be configured with one of these methods:

- Directly by an *AuthBy DIAMETER* with *Peer*, *Port* and other parameters. This peering is only usable from within the AuthBy; or
- With separate *DiaPeerDef* clauses that can be shared between multiple *AuthBy DIAMETER* clauses and other Diameter modules.

See `goodies/diameter-authby.cfg` for a sample configuration file.

<AuthBy DIAMETER> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165. *<AuthBy DIAMETER>* supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

3.96.1. DiaPeerDef

This parameter defines the Diameter peers to which requests should be sent to. When no *DiaPeerDef* parameters are set for an *AuthBy DIAMETER*, Diameter peer connection is established with the parameters configured directly within the AuthBy. *DiaPeerDef* requires Radiator Service Provider pack. Configuration with *DiaPeerDef* provides more flexibility, for example, when more than one Diameter peering is needed.

Multiple *DiaPeerDef* parameters allow using load balancing algorithms to relay to multiple destinations. For more information about available relay options, see file:/data/radiator-reference-manual/source/Configuration/AuthByDIAMETER/RelayAlgorithm_diaclient.dita#RelayAlgorithm.

When one or more *DiaPeerDef* parameters are set, the following parameters within an *AuthBy DIAMETER* are ignored: *Peer*, *SCTPPeer*, *Port*, *Protocol*, *AuthApplicationIds*, *AcctAppliationIds*, *SupportedVendorIds*, *LocalAddress*, *LocalPort*, *ReconnectTimeout* and *DisconnectTraceLevel*. These parameters are ignored because the AuthBy does not establish a direct peering.

Example

```
# Relay the requests to peers defined by DiaPeerDef with
# Identifier dra-1 and dra-2 using Session-Id based load balancing.
DiaPeerDef DiaPeerDef-Identifier=dra-1
DiaPeerDef DiaPeerDef-Identifier=dra-2

RelayAlgorithm HashBalance
```

3.96.2. RelayAlgorithm

This parameter defines the how Diameters requests should be relayed to the peers. At least one *DiaPeerDef* parameters is required. Some load balancing relay algorithms support multiple destinations.

The following algorithms are currently supported:

- `FailOver`

  Messages are sent to one peer only. If the peer becomes unreachable, the next configured peer is used. When a failed peer becomes available, it's used again.

- `HashBalance`

Messages are distributed between all configured peers. If a peer is unavailable, the messages are distributed between the remaining peers. Diameter `Session-Id` attribute is used as the distribution key.

`FailOver` can use run-time information about peers for selecting the next hop. For example when a peer advertises its supported applications, `Peer-Auth-Application-Id` can be used to select peers based on the applications they currently advertise.

`HashBalance` adds to its targets only those `DiaPeerDef` clauses that can be found during Radiator startup. Using `DiaPeerDef-Identifier` is recommended. Here's an example:

Example

```
# Balance load to multiple Diameter peers based on Diameter Session-Id attribute
RelayAlgorithm HashBalance

# Balance to peers defined with these Identifier values
DiaPeerDef DiaPeerDef-Identifier=aaa-server1
DiaPeerDef DiaPeerDef-Identifier=aaa-server2
DiaPeerDef DiaPeerDef-Identifier=aaa-server3
DiaPeerDef DiaPeerDef-Identifier=aaa-server4
```

3.96.3. Peer

Name or IP address of DIAMETER peer this *`AuthBy DIAMETER`* should connect to. When one or more DiaPeerDef parameters are configured, this parameter is ignored.

Note

Currently only one Peer is supported.

3.96.4. SCTPPeer

This parameter specifies one host name or address of an SCTP peer to connect to. When one or more DiaPeerDef parameters are configured, this parameter is ignored within *`AuthBy DIAMETER`*. An address can be an IPv4 or IPv6 address. Multiple *`SCTPPeer`* parameters are supported. When *`SCTPPeer`* is defined, it is used instead of *`Host`* or *`Peer`* parameters. Special formatting characters are supported. If SCTP multihoming is not supported, connection is attempted to each peer at a time.

When SCTP multihoming is supported, connection is attempted to all peers at once. In this case, all addresses defined with *`SCTPPeer`* must be either IPv4 or IPv6 addresses.

Here is an example of using *`SCTPPeer`*:

```
# Peer has multiple IPv6 addresses
SCTPPeer 2001:db8:1500:1::a100
SCTPPeer 2001:db8:1500:2::a100
```

3.96.5. Port

This optional parameter specifies port name or number of the Diameter peer. Defaults to 3868, the official IANA port number for Diameter. May be a numeric port number or symbolic port/service name. When one or more DiaPeerDef parameters are configured, this parameter is ignored.

3.96.6. DestinationHost

If *DestinationHost* is unset, no Destination-Host attribute is added to Diameter messages. Setting *DestinationHost* is optional and there is no default value. Special formatting characters are supported. Formatting is done when the configuration is loaded and <*AuthBy DIAMETER*> clause is activated.

3.96.7. DestinationRealm

This optional parameter sets the Destination-Realm attribute in the Diameter messages sent to the peer. Destination-Realm is first taken from username's realm part. If there is no realm, then *DestinationRealm* configuration parameter is used. The default is **testdestinationrealm**. Special formatting characters are supported. Formatting is done when the configuration is loaded and <*AuthBy DIAMETER*> clause is activated.

3.96.8. OriginHost

This parameter specifies the name that AuthBy DIAMETER uses to identify itself to a Diameter peer it connects to when no DiaPeerDef parameters are configured.

After the Diameter connection has been established, *OriginHost* sets the value of the Origin-Realm attribute in the Diameter messages sent to the peer. *OriginHost* is not optional and must be specified in the *AuthBy DIAMETER* clause. *OriginHost* defaults to the hostname of the server Radiator is running on. Special formatting characters are supported. Formatting is done when the configuration is loaded and *AuthBy DIAMETER* clause is activated.

3.96.9. OriginRealm

This parameter sets the value of the Origin-Realm attribute in the Diameter messages sent to the peer. OriginRealm is not optional an must be specified in every *AuthBy DIAMETER* clause. *OriginRealm* defaults to **testoriginrealm**. Special formatting characters are supported. Formatting is done when the configuration is loaded and *AuthBy DIAMETER* clause is activated.

3.96.10. PostDiaToRadiusConversionHook

This optional parameter allows you to define a Perl function that will be called during packet processing. PostDiaToRadiusConversionHook is called after an incoming Diameter request has been converted to its equivalent RADIUS request, allowing you to alter or add to attribute conversions etc. It is passed references to the incoming Diameter request and the converted RADIUS request.

3.96.11. PostRadiusToDiaConversionHook

This optional parameter allows you to define a Perl function that will be called during packet processing. PostRadiusToDiaConversionHook is called after an RADIUS reply has been converted to its equivalent Diameter reply, prior to being sent back to the Diameter client. It is passed references to the RADIUS reply and the converted Diameter reply.

3.96.12. EAP_ApplicationId

EAP_ApplicationId defines the Diameter message's Application-ID value and Auth-Application-Id AVP value for the converted RADIUS EAP requests. The default is to convert RADIUS EAP authentication to Diameter EAP application. This parameter allows, for example, converting RADIUS EAP-AKA to Diameter 3GPP SWm. *EAP_ApplicationId* defaults to value Diameter-EAP. For more information, see configuration sample goodies/diameter-authby.cfg

```
# We can convert EAP-AKA to SWm
EAP_ApplicationId 3GPP SWm
```

3.96.13. Protocol

This optional parameter specifies which Stream protocol will be used to carry Diameter. Options are 'tcp' for TCP/IP or 'sctp' for SCTP (Stream Control Transmission Protocol). Defaults to 'tcp'. Not all hosts are able to support 'sctp': consult your vendor. The protocol setting must be the same as that being used by the Diameter server.

```
Protocol sctp
```

3.96.14. AuthApplicationIds

This optional parameter allows you to define the Auth Application Ids announced in CER. Defaults to '0, 1, 5' (i.e. DIAMETER BASE, NASREQ and Diameter-EAP).

```
AuthApplicationIds 0, 1
```

3.96.15. AcctApplicationIds

This optional parameter allows you to define the Acct Application Ids announced in CER. Defaults to ‘3’ (i.e. BASE_ACCOUNTING).

```
AcctApplicationIds 3
```

3.96.16. SupportedVendorIds

This optional parameter allows you to define the Supported Vendor Ids announced in CER. There is no default and no Supported-Vendor-Id is announced by default. Keyword "DictVendors" is an alias group for all vendors in the default dictionary and the dictionary file configured with DiameterDictionaryFile.

```
# Tell the peer we support all the vendors in our
# default and DiameterDictionaryFile dictionaries
SupportedVendorIds DictVendors
```

3.96.17. LocalAddress and LocalPort

These parameters control the address and optionally the port number used for the client source port, although this is usually not necessary. *`LocalPort`* is a string, it can be a port number or name. It binds the local port if *`LocalAddress`* is defined. If *`LocalPort`* is not specified or if it is set to **`0`**, a port number is allocated in the usual way.

When SCTP multihoming is supported, multiple comma separated addresses can be configured. All addresses defined with *`LocalAddress`* must be either IPv4 or IPv6 addresses.

```
LocalAddress 203.63.154.29
LocalPort 12345
```

3.96.18. ReconnectTimeout

This optional parameter specifies the number of seconds to wait before attempting to reconnect a failed, dropped or disconnected connection. It also specifies the timeout for the initial connect.

3.96.19. StreamStateChangeHook

This optional parameter allows you to define a Perl function that will be called when a Stream connection state to a peer changes. RadSec, Diameter, TACACS+, HTTP and SIGTRAN clients and servers use Stream

connections in Radiator. This is a low layer hook that runs before, for example, RadSec TLS handshake or Diameter Capabilities Exchange messages are exchanged. The following arguments are passed in the following order:

- Reference to this `Radius::Stream` derived type. Examples of derived types are `Radius::RadsecHost` and `Radius::DiameterConnection`.
- New stream state. One of integer constant values:
  - `$Radius::Stream::STREAM_STATE::CONNECTED`
  - `$Radius::Stream::STREAM_STATE::DISCONNECTED`

`StreamStateChangeHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. IP address and other specific information is not passed to the hook. Their format and type depends on the derived type. For example, SCTP streams may have multiple source and destination addresses. The following example shows how to log information from the hook no matter what the type of the stream object is.

```
StreamStateChangeHook sub { \
    my ($self, $new_state) = @_; \
    my $state = ($new_state == $Radius::Stream::STREAM_STATE::CONNECTED) ? \
                  'connected' : 'disconnected'; \
    main::log($main::LOG_INFO, "StreamStateChangeHook: State change to $state"); \
    return; }
```

3.96.20. DisconnectTraceLevel

This optional parameter specifies log trace level for peer initiated disconnects. The default value is error level **`0`**. When connections are known to be short-lived, a non-default value may be useful. This parameter is available for all Stream based modules, such as *`<ServerDIAMETER>`* and *`<AuthBy RADSEC>`*.

```
# Debug logging is enough for peer disconnects
DisconnectTraceLevel 4
```

3.97. <AuthBy RATELIMIT>

AuthBy RATELIMIT allows to limit the maximum number of requests per second that will be served. When the rate is exceeded, the requests will be ignored by default.

3.97.1. MaxRate

Number of requests per second that will be accepted. Defaults to 0 which means no limit.

3.97.2. MaxRateResult

Result to use when MaxRate is exceeded. Possible values are ACCEPT, REJECT, IGNORE and CHALLENGE. Defaults to IGNORE.

3.98. <AuthBy GOSSIP>

<AuthBy GOSSIP> module supports authentication and authorisation against a Gossip backend, such as GossipUDP and GossipREDIS. For more information, see Section 3.135. <GossipUDP> on page 441 and Section 3.134. <GossipRedis> on page 439. <AuthBy GOSSIP> is currently experimental and will be documented later.

3.98.1. Gossip

This defines the Gossip module to use for backend queries.

```
<GossipUDP>
    Identifier gossip-udp
    # GossipUDP parameters
</GossipUDP>

<AuthBy GOSSIP>
    # Use GossipUDP for authentication
    Gossip gossip-udp
    # AuthBy GOSSIP parameters
</AuthBy>
```

3.99. <AuthBy DYNAUTH>

<AuthBy DYNAUTH> builds RFC 5716 Disconnect-Request and CoA-Request messages and dispatches them to Handlers. The dispatched dynauth requests can be matched with <Handler DynAuthRequest=1>. This Handler typically uses <AuthBy RADIUSBYATTR> for forwarding the newly built dynauth requests to the NAS based on the dynauth request contents. The dynauth responses will be handled by <AuthBy DYNAUTH>.

<AuthBy DYNAUTH> can optionally register itself with Gossip to receive requests from, for example, remote management to send dynauth messages pertaining to the online users. <AuthBy DYNAUTH> works with <SessionDatabase REDIS> to share session information between Radiator instances and user management. For more information about Gossip and <SessionDatabase REDIS>, see Section 11. Using Gossip framework on page 507 and Section 3.19. <SessionDatabase REDIS> on page 126.

<AuthBy DYNAUTH> is currently experimental and will be documented later.

3.99.1. NasAddrAttribute

This is a list of attributes in request that contains NAS IP address to which dynauth Radius requests are sent. Value of last present attribute is used. Special formatting characters are supported when configured as NasAddrAttribute value, formatted with %0 replaced by NAS IP address.

```
# NAS-IP-Address can be used to reach our NASes for dynauth
NasAddrAttribute NAS-IP-Address
```

3.99.2. SessionCheck

This is a list of session attributes to check. The format is: **`SessionCheck sessionattribute,comparator,value[,formatted]`**. Supported comparators are **`eq`**, **`ne`**, **`ge`**, and **`le`**.

```
# See that the user still has quota
SessionCheck user_quota,ne,0
```

3.99.3. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  `PreHandlerHook` is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

    PreHandlerHook specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

    If *EAP_LEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-MSCHAPv2

    If *EAP_PEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-GTC

    If *EAP_GTC_PAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  PreHandlerHook is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  PreHandlerHook is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  PreHandlerHook is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  PreHandlerHook is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's *PrehandlerHook* is run before ServerTACASPLUS's *PreHandlerHook*. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

PreHandlerHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *PreHandlerHook*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.100. <AuthBy RADIUSBYATTR>

<AuthBy RADIUSBYATTR> sends a RADIUS message to remote RADIUS peers based on the attributes defined in the message. This AuthBy can be used for proxying to RADIUS servers or sending RFC 5176 dynauth requests to RADIUS clients. The dynauth requests are typically created by *<AuthBy DYNAUTH>*. For more information, see Section 3.99. <AuthBy DYNAUTH> on page 361.

<AuthBy RADIUSBYATTR> is a subclass of *<AuthBy RADIUS>* and handles retransmissions automatically.

<AuthBy RADIUSBYATTR> is currently experimental and not fully documented.

3.100.1. HostsInfoAttribute

This attribute contains Host information in a request. The default is the pseudo-attribute *RadiusHosts*. If the attribute name is empty, *HostParamDef* values are used instead. For more information, see Section 3.100.2. HostParamDef on page 363. Format for the attribute value is below. The value must be on one line:

```
host1,secret,authport,acctport,dynauthport,dynauthsecret;
host2,secret,authport,acctport,dynauthport,dynauthsecret
```

Here is an example of using *HostsInfoAttribute*:

```
# Get the next hop host info from HostParmDef parameters
HostsInfoAttribute
```

3.100.2. HostParamDef

This is a list of parameter definitions for Radius Hosts. Format is: **HostParamDef hostkeyword,radiusattributename[,defaultvalue]**

```
# Set some parameters for sending from pseudo attributes in the request
HostParamDef Retries,x-retries,1
HostParamDef RetryTimeout,x-retry-timeout,2
```

3.100.3. <Host xxxxxx> within <AuthBy RADIUSBYATTR>

This clause can be used to specify the name and details of RADIUS peers inside *<AuthBy RADIUSBYATTR>*. The *<Host xxxxxx>* clause further allows you to customise details for individual peers. *<AuthBy RADIUSBYATTR>* permits one or more Host clauses.

In the *<Host xxxxxx>* clause header, the *xxxxxx* is the Host name or IP address of the remote RADIUS peer to proxy to. The *Host* name can contain special formatting characters, which are resolved at startup. Here is an example of using Host clause within *<AuthBy RADIUSBYATTR>*:

```
<AuthBy RADIUSBYATTR>
      <Host server1.test.com>
            Secret xyzzy
            AuthPort 1645
            AcctPort 1646
      </Host>
      <Host server2.test.com>
            Secret xyzzy
            AuthPort 1645
            AcctPort 1646
      </Host>
</AuthBy>
```

The following parameters can be used within a Host clause. They have the same meaning and default values as the parameter of the same name in the enclosing *<AuthBy RADIUSBYATTR>*:

- Secret on page 206
- AuthPort on page 206
- AcctPort on page 207
- Retries on page 207
- RetryTimeout on page 207
- UseOldAscendPasswords on page 216
- UseExtendedIds on page 215
- ServerHasBrokenPortNumbers on page 216
- ServerHasBrokenAddresses on page 216
- IgnoreReplySignature on page 217
- FailureBackoffTime on page 209
- MaxFailedRequests on page 209
- MaxFailedGraceTime on page 209
- LocalAddress on page 211
- OutPort on page 207

3.100.3.1. DynauthPort

This string defines the destination port to which RADIUS dynamic authorisation requests are sent to. This can be overridden for an individual host inside its Host clause. This parameter has no default value.

3.100.3.2. DynAuthSecret

This string defines the shared secret that is used for encrypting RADIUS dynamic authorisation requests that are sent to this host. The default value is the host's *Secret*.

3.101. <AuthBy RADIATORPROXY>

<AuthBy RADIATORPROXY> supports sending specially labelled RADIUS packets to a Radiator load balancer for forwarding to the actual destination NAS. This module can be used together with *<AuthBy DYNAUTH>*, and currently supports only RFC 5176 dynamic authorisation requests that need to originate from Radiator and be sent by the LB towards the NAS. For more information about *<AuthBy DYNAUTH>*, see Section 3.99. <AuthBy DYNAUTH> on page 361. Gossip framework is supported for learning the LB addresses and dynauth ports. For more information, see Section 11. Using Gossip framework on page 507.

<AuthBy RADIATORPROXY> is a subclass of *<AuthBy RADIUSBYATTR>* and automatically handles retransmissions. For more information, see Section 3.100. <AuthBy RADIUSBYATTR> on page 363.

<AuthBy RADIATORPROXY> is currently experimental and not yet fully documented.

3.101.1. DynAuthPort

This defines the default destination port to which send RADIUS dynamic authorisation requests. The default value is **3799**.

Here is an example of using *DynAuthPort*:

```
# This NAS does not use RFC 5176 defined port
DynAuthPort 1700
```

3.102. <AuthBy RATELIMITSOURCE>

Using *<AuthBy RATELIMITSOURCE>* allows you to limit the maximum number of requests that are served for a source.

The request sources can be divided into 2 separate policer groups, thus there are 2 separate parameters for the request handling parameters. Using 2 policer groups allows you to, for example, set limit for a single source or for number of sources.

See `goodies/ratelimitsource.cfg` for an example configuration.

3.102.1. SourceKey1

This string defines the format of first level policer key. The default value is **`%{Request:Calling-Station-Id}:%n`**.

3.102.2. SourceKey2

This string defines the format of second level policer key. The default value is **`%{Client:Identifier}`**.

3.102.3. MaxRate1

This integer defines the allowed maximum rate for first level policer. The default value is **`10`**.

3.102.4. MaxRate2

This integer defines the allowed maximum rate for second level policer. The default value is **`2000`**.

3.102.5. Policer1_Size

This integer defines the number of first level policer counters. The default value is **`1000`**.

3.102.6. Policer2_Size

This integer defines the number of second level policer counters. The default value is **`100`**.

3.102.7. TimeWindow1

This integer defines the time window for first level policer. The default value is **`2`**.

3.102.8. TimeWindow2

This integer defines the time window for second level policer. The default value is **`10`**.

3.102.9. MaxRateResult

This string defines the result that is used when *`MaxRate1`* or *`MaxRate2`* is exceeded. The allowed values are:

- **`ACCEPT`**
- **`REJECT`**
- **`IGNORE`**
- **`CHALLENGE`**

The values are not case-sensitive. The default value is **IGNORE**, which ignores the request.

3.103. <AuthBy FAILUREPOLICY>

<AuthBy FAILUREPOLICY> allows you to act on repeated failures. The current implementation monitors usernames and only considers those failures that indicate bad password as the failure reason.

Two policers are currently implemented. One counts consecutive failures and the other counts cumulative failures. Both policers implement separate failure counters, policy violation time and windowing. Windowing is optional and allows counters to be reset after the configured time window has passed. Counters are not incremented when a policy violation is active. If a violation is not active when a failure occurs, both consecutive counter and cumulative counter are currently incremented.

Windowing is typically used with the cumulative policer. For example, when cumulative failure threshold is set to 200 and windows size to 12 hours, the counter is reset every 12 hours. Window time is based on Unix timestamp and is not relative to Radiator process startup.

Counters are kept in Radiator process memory. If multiple Radiator instances need to share the counters or they need to persist across process restarts and reloads, see Section 3.104. <AuthBy SQLFAILUREPOLICY> on page 368

Important

See `goodies/failurepolicy.cfg` for an example configuration. Currently a hook is needed to monitor failures and maintain counter history.

<AuthBy FAILUREPOLICY> understands also the same parameters as *<AuthBy xxxxxx>*. For more information, see Section 3.32. <AuthBy xxxxxx> on page 165.

3.103.1. DefaultResult

Result for those requests that are deemed to pass the policy defined by this clause. The default is to **IGNORE**. This matches the default *AuthByPolicy* value **ContinueWhileIgnore**. When the policy is not violated, the next AuthBy is evaluated.

```
# This Handler has multiple AuthBys that all need to pass
DefaultResult ACCEPT
```

3.103.2. PolicyResult

Result for those requests that are deemed to violate the policy defined by this clause. The default is to **REJECT**.

```
# Complex AuthByPolicies can require to change the default
PolicyResult IGNORE
```

3.103.3. FailurePolicyContext

FailurePolicyContext is a string parameter that allows grouping policies defined in different AuthBys. The default value is an empty string which puts all monitored policies in the same group.

For example, when failure policy clauses are configured for two Handlers with the same *FailurePolicContext* value, it is enough for a policy violation to occur in one Handler. The subsequent attempts by the same source will be deemed to be in violation by the both Handlers. With different

FailurePolicyContext values, a violation detected in one Handler does not effect policing done within the other Handler.

```
# This AuthBy has its separate policy group
FailurePolicyContext clause1-policy
```

3.103.4. ConsecutiveFailures

This integer value parameter defines how many consecutive failures trigger a policy violation. There is no default and the policy is not enabled.

```
# 5 consecutive failures trigger a policy violation
ConsecutiveFailures 5
```

3.103.5. ConsecutiveLockTime

This integer value parameter defines in seconds how long a policy violation remains active. There is no default and the lock opens immediately.

```
# When ConsecutiveFailures is hit, violation is active for 5 minutes
ConsecutiveLockTime 300
```

3.103.6. ConsecutiveWindow

This integer value parameter defines in seconds the size of windows within which the consecutive failures must occur. There is no default and consecutive failures are not limited by time. While windowing is available with consecutive failure policer, most configure may choose not to active it.

```
# Reset consecutive counter every two minutes
ConsecutiveWindow 120
```

3.103.7. CumulativeFailures

This integer value parameter defines how many cumulative failures trigger a policy violation. There is no default and the policy is not enabled.

```
# 200 cumulative failures trigger a policy violation
CumulativeFailures 200
```

3.103.8. CumulativeLockTime

This integer value parameter defines in seconds how long a policy violation remains active. There is no default and the lock opens immediately.

```
# When CumulativeFailures is hit, violation is active for 4 hours
CumulativeLockTime 14400
```

3.103.9. CumulativeWindow

This integer value parameter defines in seconds the size of windows within which the cumulative failures must occur. There is no default and cumulative failures are not limited by time. Windowing is typically used with cumulative failure policy to prevent eventual policy violations.

```
# Reset cumulative counter every 12 hours
```

```
CumulativeWindow 43200
```

3.104. <AuthBy SQLFAILUREPOLICY>

<AuthBy SQLFAILUREPOLICY> provides a persistent and multi-server implementation of *AuthBy FAILUREPOLICY*. With this clause the counters are stored in an SQL database.

See `goodies/failurepolicy.cfg` for an example configuration.

<AuthBy SQLFAILUREPOLICY> understands also the same parameters as *<AuthBy FAILUREPOLICY>*. For more information, see Section 3.103. <AuthBy FAILUREPOLICY> on page 366. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.104.1. AddQuery and AddQueryParam

This SQL statement is executed whenever a new policy state entry is created for a username. It is expected to record the details of both consecutive and cumulative counters, policy violation and current window time in the SQL database. Special formatting characters are replaced as follows:

- %0 with the username
- %1 with the *FailurePolicyContext* parameter value
- %2 with the current consecutive failures
- %3 with the current consecutive failure window end timestamp
- %4 with the current consecutive lock end timestamp
- %5 with the current cumulative failures
- %6 with the current cumulative failure window end timestamp
- %7 with the current cumulative lock end timestamp

There is no default and *AddQuery* is not run. *AddQuery* is not needed if *UpdateQuery* can do both the add and update. *AddQuery* supports SQL bind variables with *AddQueryParam*. See the sample configuration file `goodies/failurepolicy.cfg` for an example query. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.104.2. UpdateQuery and UpdateQueryParam

This SQL statement is executed whenever a policy state entry for a username needs an update. It is also executed when *AddQuery* is not defined and policy state needs to be added. It is expected to record the details of both consecutive and cumulative counters, policy violation and current window time in the SQL database. Special formatting characters are replaced as follows:

- %0 with the username
- %1 with the *FailurePolicyContext* parameter value
- %2 with the current consecutive failures
- %3 with the current consecutive failure window end timestamp
- %4 with the current consecutive lock end timestamp
- %5 with the current cumulative failures
- %6 with the current cumulative failure window end timestamp
- %7 with the current cumulative lock end timestamp

There is no default and *UpdateQuery* is not run. *UpdateQuery* supports SQL bind variables with *pdateQueryParam*. See the sample configuration file goodies/failurepolicy.cfg for an example query. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.104.3. GetQuery and GetQueryParam

This SQL statement is executed to get a policy state entry for a username. It is expected to return the details of both consecutive and cumulative counters, policy violation and current window time in the SQL database. Special formatting characters are replaced as follows:

- %0 with the username
- %1 with the *FailurePolicyContext* parameter value

The returned columns are used as described below. Column numbering starts at 0:

- Current consecutive failures as column 0
- Current consecutive failure window end timestamp as column 1
- Current consecutive lock end timestamp as column 2
- Current cumulative failures as column 3
- Current cumulative failure window end timestamp as column 4
- Current cumulative lock end timestamp as column 5

There is no default and *GetQuery* is not run. *GetQuery* supports SQL bind variables with *GetQueryParam*. See the sample configuration file goodies/failurepolicy.cfg for an example query. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.105. <AuthLog xxxxxx>

This module marks the beginning of an AuthLog clause, which defines how to log authentication failures and successes. The xxxxxx is the name of a specific AuthLog module. This section lists the parameters all AuthLogs use. AuthLogs can also use other parameters that are specific for that AuthLog. AuthLog clauses can be defined at the top level or within a Realm or Handler clause.

You can have more than one AuthLog clause for a Realm or Handler. This makes the Realm or Handler log to each AuthLog method in turn.

3.105.1. Identifier

This parameter specifies an optional symbolic name that can be used to refer to the logger from somewhere else:

```
<AuthLog FILE>
      Identifier logger1
      ...
</AuthLog>
<Handler>
      AuthLog logger1
      ...
</Handler>
```

3.105.2. LogSuccess

This parameter indicates whether authentication successes are to be logged. They are not logged by default.

Here is an example of using *LogSuccess*:

```
# Change success logging to be on
LogSuccess 1
```

3.105.3. LogFailure

This parameter indicates whether authentication failures are to be logged. They are logged by default.

Here is an example of using *LogFailure*:

```
# Change failure logging to be off
LogFailure 0
```

3.105.4. LogIgnore

This is a flag parameter. When it is set, ignored authentication attempts are logged. An authentication is typically ignored when the user database fails or Radiator cannot return an accept or reject for some other reason. Proxied requests that return an immediate ignore are not logged because a reply with the final result is expected later.

LogIgnore is not set by default.

3.106. <AuthLog FILE>

This clause logs the authentication successes and failures to a flat file. You can define as many *<AuthLog FILE>* clauses as you need to at the top level or within the Realm or Handler clauses. Each clause can specify different logging conditions and a different log file.

Here is an example of using *<AuthLog FILE>*:

```
# This auth logger logs both success and failure to a file. It
# also log authentications that are ignored.
<AuthLog FILE>
      Identifier myauthlogger
      Filename %L/authlog
      LogSuccess 1
      LogFailure 1
      LogIgnore  1
</AuthLog>
<Realm DEFAULT>
      <AuthBy FILE>
            Filename %D/users
      </AuthBy>
      # Log authentication results to a file
      AuthLog myauthlogger
</Realm>
```

<AuthLog FILE> understands also the same parameters as all AuthLogs. For more information, see Section 3.105. <AuthLog xxxxxx> on page 369.

3.106.1. Filename

This optional parameter specifies the name of the file where authentication log messages are to be written. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. The default value is `%L/password.log`. Special character %0 is replaced by the result of the authentication and %1 by the reason string.

If the Filename parameter starts with a vertical bar character ('|'), the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output is appended to the named file:

```
# Pipe to my-log-prog
Filename |/usr/local/bin/my-log-prog
```

3.106.2. SuccessFormat

This optional parameter specifies the format that is to be used to log authentication successes in Filename when LogFormatHook is not defined. You can use any of the special characters. For more information about special characters, see Section 3.3. Special formatters on page 21. %0 is replaced by the message severity level, %1 by the reason string (usually an empty string for success), and %2 by the tracing identifier. The default is **`%l: %U:%P:OK`**. This logs time stamp in long format, current User-Name, decoded password and text OK.

CAUTION

The default SuccessFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, SuccessFormat that does not include the %P (decoded password) special character is preferable.

3.106.3. FailureFormat

This optional parameter specifies the format that is to be used to log authentication failures in Filename when LogFormatHook is not defined. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. Also %0 is replaced by the message severity level, %1 by the reason string and %2 by the tracing identifier. The default value is **`%l:%U:%P:FAIL`**. This logs time stamp in long format, current User-Name, decoded password and text FAIL.

CAUTION

The default FailureFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, FailureFormat that does not include the %P (decoded password) special character is preferable.

3.106.4. IgnoreFormat

This optional parameter specifies the format that is to be used to log ignored authentication requests in Filename when LogFormatHook is not defined. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. Also %0 is replaced by the message severity level, %1 by the reason string and %2 by the tracing identifier. The default value is **`%l:%U: %P:IGNORE`**. This logs time stamp in long format, current User-Name, decoded password and text IGNORE.

CAUTION

The default IgnoreFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, IgnoreFormat that does not include the %P (decoded password) special character is preferable.

3.106.5. LogFormatHook

This specifies an optional Perl hook that runs for each log message when defined. The hook must return the formatted log message. By default no hook is defined and *SuccessFormat* and *FailureFormat* are used for formatting. The hook parameters are the message severity level, the reason string, a reference to the current request a tracing identifier string.

Here is an example of using *LogFormatHook*:

```
# This auth logger logs both success and failure to a file in
# JSON format. The JSON Perl module must be installed.
<AuthLog FILE>
      Identifier myauthlogger-json
      Filename %L/authlog.json
      LogFormatHook sub { Radius::LogFormat::format_authlog_json(@_); }
      LogSuccess 1
      LogFailure 1
</AuthLog>
```

For more examples, see `goodies/logformat.cfg`.

Note

Consider installing `Cpanel::JSON::XS` or `JSON::XS` for higher performance JSON encoding.

3.107. <AuthLog SQL>

The clause indicates to log authentication successes and failures to an SQL database. You can define as many *<AuthLog SQL>* clauses as you wish at the top level or within Realm or Handler clauses. Each clause can specify different logging conditions and a different log database.

<AuthLog SQL> supports the same parameters as all *<AuthBy xxxxxx>*. For more information, see Section 3.105. <AuthLog xxxxxx> on page 369. It supports also all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

3.107.1. SuccessQuery

This optional parameter specifies the SQL query that will be used to log authentication successes if LogSuccess is enabled (LogSuccess is not enabled by default). There is no default. If SuccessQuery is not defined (which is the default), no logging of authentication successes will occur. In the query, special formatting characters are permitted:

- %0 is replaced with the message severity level.
- %1 is replaced with the quoted reason message (which is usually empty for successes).
- %2 is replaced with the SQL quoted User-Name.
- %3 is replaced with the SQL quoted decoded plaintext password (if any).
- %4 is replaced with the SQL quoted original user name from the incoming request (before any RewriteUsername rules were applied)
- %5 is replaced with tracing identifier.

3.107.2. SuccessQueryParam

This optional parameter specifies a bind variable to be used with SuccessQuery. For more information, see Section 3.8.1. SQL bind variables on page 47.

3.107.3. FailureQuery

This optional parameter specifies the SQL query that is used to log authentication failures if LogFailure is enabled (LogFailure is enabled by default). There is no default. If FailureQuery is not defined (which is the default), no logging of authentication failures occur. In the query, special formatting characters are permitted.

- %0 is replaced with the message severity level.
- %1 is replaced with the quoted reason message.
- %2 is replaced with the SQL quoted User-Name.
- %3 is replaced with the SQL quoted decoded plaintext password (if any).
- %4 is replaced with the SQL quoted original user name from the incoming request (before any RewriteUsername rules were applied)
- %5 is replaced with tracing identifier.

3.107.4. FailureQueryParam

This optional parameter specifies a bind variable to be used with FailureQuery. For more information, see Section 3.8.1. SQL bind variables on page 47.

3.107.5. IgnoreQuery

This optional string specifies the SQL query that is used to log ignored authentication attempts if `LogIgnore` is enabled. `LogIgnore` is not enabled by default. `IgnoreQuery` has no default value. If it is not defined, authentication ignores are not logged. In the query, special formatting characters are permitted:

- %0 is replaced with the message severity level.
- %1 is replaced with the quoted reason message (which is usually empty for successes).
- %2 is replaced with the SQL quoted User-Name.
- %3 is replaced with the SQL quoted decoded plaintext password (if any).
- %4 is replaced with the SQL quoted original user name from the incoming request (before any RewriteUsername rules were applied)

3.107.6. IgnoreQueryParam

This optional stringarray parameter specifies a bind variables to be used with `IgnoreQuery`. For more information on bind variables, see Section 3.8.1. SQL bind variables on page 47.

If you specify one or more `IgnoreQueryParam`s, they are used in order to replace parameters named with a question mark in `IgnoreQuery`. The query is cached by the SQL server for future reuse. Only the first `QueryCacheSize` queries are cached.

3.108. <AuthLog SYSLOG>

This clause indicates to log authentication successes and failures to the syslog logging facility. You can define as many <AuthLog SYSLOG> clauses as you wish at the top level or within Realm or Handler clauses. Each clause can specify different logging conditions and a different log database. This module was contributed by Carlos Canau (Carlos.Canau@KPNQwest.pt).

<AuthLog SYSLOG> understands also the same parameters as all AuthLogs. For more information, see Section 3.105. <AuthLog xxxxxx> on page 369.

3.108.1. Facility

The name of the syslog facility that will be logged to. The default is **user**.

```
# Log to the syslog facility called 'auth'
Facility auth
```

3.108.2. Priority

The syslog priority level that will be used for each log message. Default is **info**.

```
# Increase this logger priority to 'notice'
Priority notice
```

3.108.3. SuccessFormat

This optional parameter specifies the format that is to be used to log authentication successes in Filename when LogFormatHook is not defined. You can use any of the special characters. For more information about special characters, see Section 3.3. Special formatters on page 21. %0 is replaced by the message severity level, %1 by the reason string (usually an empty string for success), and %2 by the tracing identifier. The default is **%l:%U:%P:OK**. This logs time stamp in long format, current User-Name, decoded password and text OK.

CAUTION

The default SuccessFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, SuccessFormat that does not include the %P (decoded password) special character is preferable.

3.108.4. FailureFormat

This optional parameter specifies the format that is to be used to log authentication failures in Filename when LogFormatHook is not defined. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. Also %0 is replaced by the message severity level, %1 by the reason string and %2 by the tracing identifier. The default value is **%l:%U:%P:FAIL**. This logs time stamp in long format, current User-Name, decoded password and text FAIL.

CAUTION

The default FailureFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, FailureFormat that does not include the %P (decoded password) special character is preferable.

3.108.5. IgnoreFormat

This optional parameter specifies the format that is to be used to log ignored authentication requests in Filename when LogFormatHook is not defined. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. Also %0 is replaced by the message severity level, %1 by the reason string and %2 by the tracing identifier. The default value is **%l:%U:%P:IGNORE**. This logs time stamp in long format, current User-Name, decoded password and text IGNORE.

CAUTION

The default IgnoreFormat logs the plaintext password entered by the user. Some organisations prefer that user passwords are not logged. In that case, IgnoreFormat that does not include the %P (decoded password) special character is preferable.

3.108.6. LogSock

This optional parameter specifies what type of socket to use to connect to the syslog server. The possible values are:

- **native**
- **eventlog**
- **unix**
- **inet**

  This means that TCP is tried first, then UDP.
- **tcp**
- **udp**
- **stream**
- **pipe**
- **console**

The default is to use the *Sys::Syslog* default of **native, tcp, udp, unix, pipe, stream, console**.

CAUTION

Due to limitations in the Sys::Syslog Perl module, if you have multiple *<AuthLog SYSLOG>*, *<AcctLog SYSLOG>* or *<Log SYSLOG>* clauses and if any one has *LogSock* defined, all of them must have *LogSock* defined.

Note

If you use TCP, we recommend you to define both *LogHost* and *LogPort*. If you have not defined *LogPort* and you see error "TCP service unavailable", this means Sys::Syslog is unable to find the destination port. To resolve this, either use *LogPort* to define the port or add *syslog/tcp* or *syslogng/tcp* definitions to /etc/services file. For more information about *LogPort*, see Section 3.26.10. LogPort on page 141.

3.108.7. LogPath

When LogSock is set to unix or stream or pipe, this optional parameter specifies the syslog path. Defaults to _PATH_LOG macro (if your system defines it).

```
LogPath /run/mysyslog/log.sock
```

3.108.8. LogHost

When *LogSock* is set to **tcp** or **udp** or **inet**, this optional parameter specifies the name or address of the syslog host. Defaults to the local host. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

Note

The *LogHost* parameter is passed directly to Perl's `Sys::Syslog` module which will likely do a DNS query for each logged message. This can cause performance problems and high number of DNS requests with verbose log levels. It is recommended to not set *LogSock* and let the local syslog to do remote logging.

Note

`Sys::Syslog` does not support IPv6. To log over IPv6, leave *LogSock* unset and let the local syslog do remote logging over IPv6.

```
# Log to a remote host via syslog over udp:
LogSock udp
LogHost your.syslog.host.com
```

3.108.9. LogOpt

This optional parameter allows control over the syslog options passed to `Sys::Syslog::openlog`. *LogOpt* is a comma separated list of words from the set:

- **cons**
- **ndelay**
- **nofatal**
- **nowait**
- **perror**
- **pid**

As described in the Perl `Sys::Syslog` documentation.

Defaults to **pid**. Special characters are supported.

```
LogOpt pid,perror
```

3.108.10. LogIdent

This optional string parameter specifies an alternative `ident` name `Sys::Syslog` prepends to every syslog message. Defaults to the executable name used to run radiusd. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

```
# Also log server farm instance number
LogIdent %h-%O
```

3.108.11. LogPort

This optional parameter specifies an alternative TCP or UDP destination port on the syslog host. There is no default, which means `Sys::Syslog` chooses the port. Here is an example of using *`LogPort`*:

```
LogPort 5514
```

CAUTION

This parameter requires `Sys::Syslog` 0.28 or later.

3.108.12. MaxMessageLength

This optional parameter specifies a maximum message length (in characters) for each message to be logged. If specified, each log message is truncated to the specified number of characters prior to logging. Defaults to 0, which means no truncation.

3.109. <AuthLog EVENTLOG>

This clause logs authentication successes and failures to Microsoft Windows Event Log. There is an example configuration available in `goodies/eventlog.cfg`.

<AuthLog EVENTLOG> understands also the same parameters as all AuthLogs. For more information, see Section 3.105. <AuthLog xxxxxx> on page 369.

Note

<AuthLog EVENTLOG> is only available when Radiator is running on Windows

3.109.1. SuccessFormat

This optional parameter specifies the format that is to be used to log authentication successes. You can use any of the special characters. For more information about special characters, see Section 3.3. Special formatters on page 21. The default value is **`Access-Accept for user %U`**.

3.109.2. FailureFormat

This optional parameter specifies the format that is to be used to log authentication failures. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. The default value is **`Access-Reject for user %U`**.

3.109.3. IgnoreFormat

This string defines the format of an ignore message. You can use any special characters. For more information, see Section 3.3. Special formatters on page 21. The default value is **`Ignore for user %U`**.

3.110. <AcctLog xxxxxx>

This section lists the parameters that are common for all AcctLog modules. Apart from these common parameters, AcctLog modules have parameters that are specific to that AcctLog module. AcctLog modules define how to log accounting data. You can define AcctLog clauses at the top level or within a specific Realm or Handler clause.

For AcctLog examples, see `goodies/logformat.cfg` file in the Radiator distribution.

3.110.1. LogFormatHook

This defines the Perl hook that is run for each log message. It is not set by default. For LogFormatHook examples, see `goodies/logformat.cfg` file in the Radiator distribution.

3.111. <AcctLog EVENTLOG>

This clause logs accounting requests to Microsoft Windows Event Log. You can define as many AcctLog EVENTLOG clauses as you wish at the top level or within Realm or Handler clauses. Each clause can specify different logging conditions and a different log file.

Note

AcctLog EVENTLOG is only available when Radiator is running on Windows

---

3.111.1. LogFormat

This string defines the format for success messages. You can use any special characters. For more information, see Section 3.3. Special formatters on page 21. The bind variables are replaced as follows:

- **`%0`**: the result
- **`%1`**: the reason string, usually this is empty if successful
- **`%2`**: the tracing identifier string

The default value is:

```
LogFormat Accounting for user %n, \
          Accounting-Session-Id: %{Acct-Session-Id}, \
          Framed-IP-Address: %{Framed-IP-Address}, \
          Calling-Station-Id: %{Calling-Station-Id}
```

3.112. <AcctLog FILE>

This clause logs accounting requests to a flat file. You can define as many *`<AcctLog FILE>`* clauses as you wish at the top level or within *`Realm`* or *`Handler`* clauses. Each clause can specify different logging conditions and a different log file.

`<AcctLog FILE>` supports all the common AcctLog configuration parameters. For more information about the AcctLog configuration parameters, see Section 3.110. <AcctLog xxxxxx> on page 377.

3.112.1. LogFormat

This string defines the format for success messages. You can use any special characters. For more information, see Section 3.3. Special formatters on page 21. The bind variables are replaced as follows:

- **`%0`**: the result
- **`%1`**: the reason string, usually this is empty if successful
- **`%2`**: the tracing identifier string

The default value is:

```
LogFormat %l:%n:%{Acct-Session-Id}:%{Framed-IP-Address}:%{Calling-Station-Id}
```

3.112.2. OutputFormat

This optional string parameter defines the format for the log messages. Format specifiers, such as *%{GlobalVar:name}*, are evaluated when the configuration is loaded. Currently supported values are **text** and **json**. Default value is **text**.

> **Tip**
>
> For the best JSON encoding performance, check your Perl installation includes `Cpanel::JSON::XS` or `JSON::XS`.

For extensive customisation of JSON and other formats, see *LogFormatHook* and `Radius::LogFormat` module.

```
# We have switched from LogFormatHook to OutputFormat
#LogFormatHook sub { Radius::LogFormat::format_acctlog_json(@_); }
OutputFormat json
```

3.112.3. AcctLogOutputDef

This optional parameter allows you to change the way Radiator formats accounting log JSON output by mapping RADIUS attributes to JSON fields. Multi-valued RADIUS attributes are added as a JSON array. When this parameter is not defined, all RADIUS attributes are encoded. The general format is:

```
AcctLogOutputDef jsonattributename[,radiusattributename]
```

If **radiusattributename** is omitted, **jsonattributename** is used as RADIUS attribute name. Special variables, such as %n, can also be used in **radiusattributename**. Note that in this case % must be the first character in the **radiusattributename**.

For extensive customisation of JSON and other formats, see *LogFormatHook* and `Radius::LogFormat` module. For a configuration sample, see `goodies/logformat.cfg` file in the Radiator distribution.

```
# Use JSON and define what we want in the JSON output
OutputFormat json

# Use RADIUS attribute as JSON field name
AcctLogOutputDef Connect-Info

# Map attribute to JSON field
AcctLogOutputDef Custom-Event-Timestamp, Event-Timestamp

# Use special variables
AcctLogOutputDef Custom-Original-Username, %u
AcctLogOutputDef Custom-User-Name, %n
```

3.112.4. Filename

This optional parameter specifies the name of the file where the accounting log messages are written. You can use any special characters. For more information, see Section 3.3. Special formatters on page 21. The default value is **%L/accounting.log**.

3.113. <AcctLog SQL>

This clause logs accounting requests to an SQL database. You can define as many AcctLog SQL clauses as you wish at the top level or within Realm or Handler clauses. Each clause can specify different logging conditions and a different log file.

<AcctLog SQL> supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

For an AcctLog SQL configuration sample, see `goodies/sql.cfg` file in the Radiator distribution.

3.113.1. LogQuery

This string defines the SQL query that logs the authentication successes. It is an optional parameter and has no default value. You can use special characters. For more information, see Section 3.3. Special formatters on page 21. If it is not defined, accounting is not logged at all.

The bind variables are replaced as follows:

- **`%0`**: the result
- **`%1`**: the quoted reason string, usually this is empty if successful
- **`%2`**: the SQL-quoted User-Name
- **`%3`**: the SQL-quoted original user name from the incoming request (before any RewriteUsername rules were applied)
- **`%4`**: the tracing identifier string

3.113.2. LogQueryParam

This string array enables the use of bound variables, if they are supported by the SQL server, and query caching. This is an optional parameter. When one or more *`LogQueryParameter`*s are defined, they are used to replace parameter names with question marks in *`LogQuery`* and the query is cached in the SQL server for future reuse. Only the first *`QueryCacheSize`* is cached.

3.114. <AcctLog SYSLOG>

This clause logs accounting requests to the syslog logging facility. You can define as many AcctLog SYSLOG clauses as you wish at the top level or within Realm or Handler clauses. Each clause can specify different logging conditions and a different log file.

3.114.1. Facility

The name of the syslog facility that will be logged to. The default is **`user`**.

```
# Log to the syslog facility called 'auth'
Facility auth
```

3.114.2. Priority

The syslog priority level that will be used for each log message. Default is **`info`**.

```
# Increase this logger priority to 'notice'
Priority notice
```

3.114.3. LogFormat

This string defines the format for success messages. You can use any special characters. For more information, see Section 3.3. Special formatters on page 21. The bind variables are replaced as follows:

- **`%0`**: the result
- **`%1`**: the reason string, usually this is empty if successful
- **`%2`**: the tracing identifier string

The default value is:

```
LogFormat %l:%n:%{Acct-Session-Id}:%{Framed-IP-Address}:%{Calling-Station-Id}
```

3.114.4. LogSock

This optional parameter specifies what type of socket to use to connect to the syslog server. The possible values are:

- **`native`**
- **`eventlog`**
- **`unix`**
- **`inet`**

  This means that TCP is tried first, then UDP.
- **`tcp`**
- **`udp`**
- **`stream`**
- **`pipe`**
- **`console`**

The default is to use the *`Sys::Syslog`* default of **`native, tcp, udp, unix, pipe, stream, console`**.

CAUTION

Due to limitations in the `Sys::Syslog` Perl module, if you have multiple *`<AuthLog SYSLOG>`*, *`<AcctLog SYSLOG>`* or *`<Log SYSLOG>`* clauses and if any one has *`LogSock`* defined, all of them must have *`LogSock`* defined.

Note

If you use TCP, we recommend you to define both *`LogHost`* and *`LogPort`*. If you have not defined *`LogPort`* and you see error "TCP service unavailable", this means `Sys::Syslog` is unable to find the destination port. To resolve this, either use *`LogPort`* to define the port or add *`syslog/tcp`* or *`syslogng/tcp`* definitions to `/etc/services` file. For more information about *`LogPort`*, see Section 3.26.10. LogPort on page 141.

3.114.5. LogPath

When LogSock is set to unix or stream or pipe, this optional parameter specifies the syslog path. Defaults to _PATH_LOG macro (if your system defines it).

```
LogPath /run/mysyslog/log.sock
```

3.114.6. LogPort

This optional parameter specifies an alternative TCP or UDP destination port on the syslog host. There is no default, which means `Sys::Syslog` chooses the port. Here is an example of using *`LogPort`*:

```
LogPort 5514
```

CAUTION

This parameter requires `Sys::Syslog` 0.28 or later.

3.114.7. LogIdent

This optional string parameter specifies an alternative `ident` name `Sys::Syslog` prepends to every syslog message. Defaults to the executable name used to run radiusd. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

```
# Also log server farm instance number
LogIdent %h-%O
```

3.114.8. LogOpt

This optional parameter allows control over the syslog options passed to `Sys::Syslog::openlog`. *`LogOpt`* is a comma separated list of words from the set:

- **`cons`**
- **`ndelay`**
- **`nofatal`**
- **`nowait`**
- **`perror`**
- **`pid`**

As described in the Perl `Sys::Syslog` documentation.

Defaults to **`pid`**. Special characters are supported.

```
LogOpt pid,perror
```

3.114.9. LogHost

When *`LogSock`* is set to **`tcp`** or **`udp`** or **`inet`**, this optional parameter specifies the name or address of the syslog host. Defaults to the local host. Special formatters are supported. For more information, see Section 3.3. Special formatters on page 21

Note

The *LogHost* parameter is passed directly to Perl's Sys::Syslog module which will likely do a DNS query for each logged message. This can cause performance problems and high number of DNS requests with verbose log levels. It is recommended to not set *LogSock* and let the local syslog to do remote logging.

Note

Sys::Syslog does not support IPv6. To log over IPv6, leave *LogSock* unset and let the local syslog do remote logging over IPv6.

```
# Log to a remote host via syslog over udp:
LogSock udp
LogHost your.syslog.host.com
```

3.114.10. MaxMessageLength

This optional parameter specifies a maximum message length (in characters) for each message to be logged. If specified, each log message is truncated to the specified number of characters prior to logging. Defaults to 0, which means no truncation.

3.115. <AddressAllocator SQL>

<AddressAllocator SQL> works in conjunction with *<AuthBy DYNADDRESS>* (see Section 3.53. <AuthBy DYNADDRESS> on page 241) to allocate IPv4 addresses and IPv6 prefixes from an SQL database. During deallocation, the address is marked as unused. Addresses that remain in use for more than *DefaultLeasePeriod* seconds are automatically reclaimed (this protects against lost Accounting Stop requests).

<AddressAllocator SQL> supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46.

The default FindQuery fetches the oldest unused address from the RADPOOL table. Change the allocation strategy by customising the SQL queries.

Starting from Radiator 4.14, you can define UpdateQuery to refresh address TIME_STAMP and EXPIRY when accounting Alive requests are received. This keeps the addresses from being automatically reclaimed while they are periodically updated by Alive requests.

The table definition of a sample RADPOOL table for a range of SQL databases can be found in the goodies/*.sql files in the Radiator distribution.

<AddressAllocator SQL> uses the PoolHint to determine which pool to use. It uses the pool hint in the SQL select statement that is used to find an available address. It does an exact match on the POOL column of the RADPOOL table.

When an address is allocated for a user, it is ‘leased’ to the user for a fixed period. It is available exclusively for that user until either they terminate their session (when an Accounting Stop is received) or until the lease expires. The purpose of this is to protect against ‘lost’ Accounting Stops such as might occur with poor network connectivity, or a crashed NAS. However it also means that you should set the lease period to be longer than the longest legitimate session time, otherwise users may find their addresses being reallocated to another user.

Since Radiator 4.14 you can use shorter lease period if accounting Alive messages are used and *UpdateQuery* is configured.

Starting with Radiator 4.18, you can use asynchronous SQL queries with some database servers. See AsynchronousSQL on page 52 and `goodies/addressallocator.sql` for the details.

The default lease period is 1 day, and you can control this with *DefaultLeasePeriod*. Every *LeaseReclaimInterval* seconds, expired leases are reclaimed and made available for allocation again.

Acct-Status-Type values Accounting-On and Accounting-Off are by default accepted and replied to with no other action. You can configure AddressAllocator SQL with DeallocateByNASQuery to, for example, release all leases for the NAS when Accounting-On or Accounting-Off is received.

<*AddressAllocator SQL*> makes the following allocation variables available for replies. These names can be used in MapAttribute in <*AuthBy DYNADDRESS*>.

- *yiaddr*, the allocated IPv4 address or IPv6 prefix from the YIADDR column of the RADPOOL table
- *subnetmask*, the IPv4 subnet mask or IPv6 mask length from the SUBNETMASK column of the RADPOOL table
- *dnsserver*, the DNS server address from the DNSSERVER column of the RADPOOL table

<*AddressAllocator SQL*> can also optionally populate the RADPOOL table at startup, by defining <*AddressPool xxxxxx*> clauses inside the <*AddressAllocator SQL*> clause.

3.115.1. Identifier

This mandatory parameter specifies an symbolic name for this AddressAllocator clause. It must exactly match the Allocator parameter in an AuthBy DYNADDRESS in order for it to be used to allocate addresses.

```
Identifier mySQLallocator
```

3.115.2. DefaultLeasePeriod

This optional parameter defines how long the default lease period is in seconds. The default is 86400 seconds (24 hours).

```
# We have long sessions, set the lease to be 2 days
DefaultLeasePeriod 172800
```

3.115.3. LeaseReclaimInterval

This optional parameter defines how often expired leases are looked for. Every LeaseReclaimInterval seconds, AddressAllocator SQL checks the RADPOOL table for expired leases, using the ReclaimQuery. If expired leases are found, they are marked as available again. The default is 86400 seconds (24 hours).

```
# Look for expired leases every hour
LeaseReclaimInterval 3600
```

3.115.4. FindQuery

This optional parameter allows you to define a custom SQL query to find an available address or prefix. The default is:

```
select TIME_STAMP, YIADDR, SUBNETMASK, DNSSERVER from RADPOOL where POOL='%0'
       and STATE=0 order by TIME_STAMP
```

The variables are replaced as follows:

- %0 by the pool hint
- %1 by the user name
- %2 with the lease expiry time
- %3 by SQL quoted NasId value

Tip

You can get a substantial speedup during address allocation with MySQL by adding 'limit 1' to the end of the FindQuery when using a database engine that supports LIMIT clause.

This example uses NAS_ID column to allocate an IPv6 prefix from the range assigned to the user's NAS. The pool has been previously initialised with NAS_IDs belonging to known NASes.

```
# Our NASes all send NAS-IPv6-Address, e.g. 2001:db8:2::2
NasId %{NAS-IPv6-Address}
FindQuery select TIME_STAMP, YIADDR, SUBNETMASK, DNSSERVER from RADPOOL
          where POOL='%0' and STATE=0 and NAS_ID=%3 order by TIME_STAMP
```

3.115.5. FindQueryBindVar

This optional parameter specifies a bind variable to be used with FindQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.6. AllocateQuery

This optional parameter allows you to define a custom SQL query to allocate an address or prefix that was found with *`FindQuery`*. The default is:

```
update RADPOOL set STATE=1,TIME_STAMP=%0, EXPIRY=%1, USERNAME=%2
       where YIADDR='%3' and STATE=0 and TIME_STAMP %4
```

The variables are replaces as follows:

- %0 by the current time in Unix epoch seconds
- %1 by the lease expiry time (the current time + the lease period)
- %2 by the user name
- %3 by the address or prefix
- %4 by a timestamp comparison operator
- %5 by the SQL quoted NasId

If the *`AllocateQuery`* is an empty string, the query is not executed. This can be useful if the *`FindQuery`* does all the allocation work.

3.115.7. AllocateQueryBindVar

This optional parameter specifies a bind variable to be used with AllocateQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.8. UpdateQuery

This optional parameter allows you to define a custom SQL query to update the address or prefix when accounting `Start` or `Alive` requests are received. There is no default.

```
UpdateQuery update RADPOOL set TIME_STAMP=?, EXPIRY=? where YIADDR=?
UpdateQueryBindVar %t
UpdateQueryBindVar %0
UpdateQueryBindVar %1
```

The variables are replaced as follows:

- %0 by the lease expiry time (the current time + the lease period)
- %1 by the address
- %2 by the SQL quoted NasId

If the *`UpdateQuery`* is an empty string, the query is not executed.

3.115.9. UpdateQueryBindVar

This optional parameter specifies a bind variable to be used with UpdateQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.10. CheckPoolQuery

This optional parameter allows you to define a custom SQL query to check whether an address or prefix exists in the pool. The default is:

```
select STATE from RADPOOL where YIADDR='%0'
```

%0 is replaced by the address.

If CheckPoolQuery is set to an empty string, no pool checking is done at startup. Delay the start of pool check with DelayedPoolCheckTime. For more information, see Section 3.115.22. DelayedPoolCheckTime on page 388.

3.115.11. CheckPoolQueryBindVar

This optional parameter specifies a bind variable to be used with *`CheckPoolQuery`*. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.12. AddAddressQuery

This optional parameter allows you to define a custom SQL query to add an address to the pool if it is not found by CheckPoolQuery. The default is:

```
insert into RADPOOL (STATE, TIME_STAMP, POOL, YIADDR, SUBNETMASK, DNSSERVER)
       values (0, %t, %0, '%1', '%2', '%3')
```

The variables are replaced as follows:

- %t by the current time in Unix epoch seconds
- %0 by the pool name
- %1 by the address
- %2 by the subnet mask

- %3 by the DNS server address
- %4 by the pool's PoolGroup
- %5 by the pool's Priority
- %6 by the pool's NasIdentifier

If AddAddressQuery is set to an empty string, addresses are not automatically added to the pool.

3.115.13. AddAddressQueryBindVar

This optional parameter specifies a bind variable to be used with AddAddressQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.14. DeallocateQuery

This optional parameter allows you to define a custom SQL query to deallocate a previously allocated address. If the DeallocateQuery is an empty string, no deallocation query is executed. The default value is:

```
update RADPOOL set STATE=0,TIME_STAMP=%t where YIADDR='%0'
```

%0 is replaced the address and %1 by the SQL quoted NasId.

3.115.15. DeallocateQueryBindVar

This optional parameter specifies a bind variable to be used with DeallocateQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.16. DeallocateByNASQuery

This optional parameter allows you to define a custom SQL query to deallocate all previously allocated addresses for a NAS. If the DeallocateByNASQuery is an empty string or not defined, no deallocation query is executed. This query is not defined by default currently. %0 is replaced by the SQL quoted NasId.

```
# Process Accounting-On and Accounting-Off
DeallocateByNASQuery update RADPOOL set STATE=0, TIME_STAMP=%t where NAS_ID=%0
```

3.115.17. DeallocateByNASQueryBindVar

This optional parameter specifies a bind variable to be used with DeallocateByNASQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.18. ReclaimQuery

This optional parameter allows you to define a custom SQL query to reclaim an address whose lease has expired. If the ReclaimQuery is an empty string, no reclaim query will be executed. Defaults to

```
update RADPOOL set STATE=0 where STATE!=0 and EXPIRY < %0
```

%0 is replaced by the current time in Unix epoch seconds. By default reclaim query runs immediately when AddressAllocator SQL module is loaded. You can delay the start of reclaim check with DelayedPoolCheckTime. For more information, see Section 3.115.22. DelayedPoolCheckTime on page 388.

3.115.19. ReclaimQueryBindVar

This optional parameter specifies a bind variable to be used with ReclaimQuery. For more information about bind variables, see Section 3.8.1. SQL bind variables on page 47.

3.115.20. NoAddressHook

NoAddressHook is called when there are no addresses left or the allocation fails because of too many simultaneous tries.

NoAddressHook is passed with the following arguments:

- Reference to the current request object
- Reference to the current reply object
- Reference to current authentication result
- Reference to a string variable holding the reason for a reject
- String variable holding value of pool hint

To change the type of reply, change the third argument from $main::REJECT to the desired value.

3.115.21. NasId

NasId defines the name of attribute in request or other value typically used for NAS_ID column. It is available as SQL quoted special value for the AddressAllocator SQL queries. The default is %{NAS-Identifier}.

```
# Our NASes use IPv6 addresses
NasId %{NAS-IPv6-Address}
```

3.115.22. DelayedPoolCheckTime

This parameter defines time in seconds Radiator delays checking address pools when AddressAllocatorSQL is activated. This is not set by default, which mean Radiator checks the pool immediately when AddressAllocator SQL loaded during the configuration. During the pool check CheckPoolQuery and ReclaimQuery are run to update the pool state. For more information, see Section 3.115.10. CheckPoolQuery on page 386 and Section 3.115.18. ReclaimQuery on page 387.

```
# Postpone the check. Do it when the config has been loaded
DelayedPoolCheckTime 1
```

3.115.23. AddressPool

The AddressPool clause allows you to define which address pools are available. When Radiator is started, each AddressPool ensures there is an entry for each of its addresses and prefixes in the RADPOOL table.

AddressPool is a simple alternative for maintaining the contents of the RADPOOL table through other method. You can use another method for initialising and maintaining the RADPOOL table, in this case it is not necessary to have any AddressPool clauses.

AddressPool defines a range of available addresses or prefixes. Each address or prefix in the range has the same Subnet mask and DNS Server address. The Subnetmask and the DNS server address specify the values to use if an address or prefix is allocated from a range. The default for Subnetmask is 255.255.255.255. There is no default for DNSServer.

IPv4 address ranges can be specified either as lower and upper addresses (inclusive) within a class C block or as a CIDR block. IPv6 prefix ranges are specified as a CIDR block.

The step size between consecutive addresses is controlled with the Step parameter, which defaults to 1. Step of other than 1 can be useful where you need to allocate subnets of more than one address, rather than individual host addresses.

Advanced configuration for address pools is supported with optional parameters PoolGroup and Priority. A PoolGroup defines a name to group multiple pools with different priorities set by Priority. These parameters are available for AddAddressQuery (for more information, see Section 3.115.6. AllocateQuery on page 385) and the values inserted in SQL can later be used by customised allocation queries and procedures.

The following example defines two pools of addresses. The first pool is called 'pool1'. It contains addresses in the ranges 192.1.1.1 to 192.1.1.50 (inclusive) and 192.1.1.60 to 192.1.1.120 (inclusive) and the entire 192.1.2.0 class C block. The IP Subnet mask for each address is 255.255.255.255. The second pool is called ‘pool2’ and contains addresses in the range 192.2.2.62 to 192.2.2.99 (inclusive). The third pool is called 'pool3' and contains 256 IPv6 prefixes with prefix length of 64 bits.

PoolGroup, PoolPriority and NasIdentifier are made available for AddAddressQuery (for more information, see Section 3.115.12. AddAddressQuery on page 386) and can later be used by FindQuery (for more information, see Section 3.115.4. FindQuery on page 384) and other queries that need to up suitable addresses.

```
<AddressAllocator SQL>
      .....
      # Defines the addresses that we are prepared
      # to allocate:
      <AddressPool pool1>
            Subnetmask 255.255.255.0
            DNSServer 10.1.1.1
            Range 192.1.1.1 192.1.1.50
            Range 192.1.1.60 192.1.1.120
            Range 192.1.2.0/24
      </AddressPool>
      <AddressPool pool2>
            Subnetmask 255.255.255.127
            Range 192.2.2.62 192.2.2.99
      </AddressPool>
      <AddressPool pool3>
            Subnetmask /56
            Range 2001:db8:100::/48
            #PoolGroup group1
            #PoolPriority 1
            NasIdentifier 2001:db8:2::2
      </AddressPool>
</AddressAllocator>
```

3.116. <AddressAllocator DHCP>

AddressAllocator DHCP works in conjunction with <AuthBy DYNADDRESS> (see Section 3.53. <AuthBy DYNADDRESS> on page 241) and a DHCP server to dynamically allocate IP addresses.

AddressAllocator DHCP has been tested with a number of DHCP servers, however it works best with the latest versions of the DHCP server from the Internet Software Consortium (ISC) that support the Subnet Selection Option. For more information, see ISC website [http://www.isc.org/].

The RFC 3011 Subnet Selection Option allows server-to-server operation such as that supported by Radiator.

Tip

The PoolHint supplied in the AuthBy DYNADDRESS clause must be a subnet definition that is understood by the DHCP server for the purposes of address allocation.

AddressAllocator DHCP makes the following allocation variables available for replies. These names can be used in MapAttribute in AuthBy DYNADDRESS.

- yiaddr, the allocated IP address from DHCP
- subnetmask, the subnet mask from DHCP
- dnsserver, the DNS server address from DHCP

An example AddressAllocator DHCP configuration file can be found in the distribution in the file `goodies/addressallocatordhcp.cfg`.

CAUTION

Because AddressAllocator DHCP binds to the DHCP server address, it is not possible to run the DHCP server on the same host as Radiator.

Because the DHCP address allocator binds to port 67, Radiator must be run as root, or at least with root privileges.

3.116.1. Identifier

This mandatory parameter specifies a symbolic name for this AddressAllocator clause. It must exactly match the Allocator parameter in an AuthBy DYNADDRESS in order for it to be used to allocate addresses.

```
Identifier        DHCPallocator
```

3.116.2. Host

The host name or IP address of the DHCP server. The address may be either a single host address or a broadcast address. If a broadcast address is specified, Radiator will allocate an address from the first DHCP host to answer a discover request.

```
Host 1.2.3.4
```

3.116.3. Port

The DHCP port number on the DHCP host. Defaults to port 67.

```
Port 67
```

3.116.4. LocalAddress

The local IP address to bind to. Defaults to the main IP address of the Radiator host.

```
LocalAddress 1.1.1.1
```

3.116.5. DHCPClientIdentifier

The Client Identifier to be used by the DHCP server to track allocations. Defaults to the user name in the current request.

```
DHCPClientIdentifier %{User-Name}
```

3.116.6. DefaultLease

Specifies the default lease period requested. Defaults to one day (86400 seconds).

```
DefaultLease 86400
```

3.116.7. TimeoutMinimum

The DHCP RFC specifies timeout and retry behaviour while attempting to contact a DHCP server. The TimeoutMinimum specifies the initial timeout in seconds. Defaults to 2 seconds.

```
TimeoutMinimum 4
```

3.116.8. TimeoutMaximum

The TimeoutMaximum specifies how long to keep retrying a request in seconds. Defaults to 16 seconds.

```
TimeoutMaximum 64
```

3.116.9. TimeoutFactor

The TimeoutFactor indicates how to recalculate the timeout for each attempt. Defaults to multiplying the timeout by 2 for each attempt.

```
TimeoutFactor 2
```

3.116.10. ServerPort

Specifies the local DHCP server port. Defaults to port 67.

```
ServerPort 67
```

3.116.11. Synchronous

Normally DHCP requests are processed asynchronously. This optional parameter can be used to make the requests synchronous.

```
Synchronous
```

3.116.12. SubnetSelectionOption

This is an optional integer parameter that is used to enable the DHCP "Subnet Selection Option". The official RFC option number is 118 which is used as the default if this parameter does not specify other value. Early versions of the ISC DHCP server used option 211.

If the *SubnetSelectionOption* parameter is not defined, no Subnet Selection Option is added and any *PoolHint* value provided is used in the giaddr field in the DHCP request, instead of *LocalAddress*.

Here are examples of using *SubnetSelectionOption*:

```
# This sets the value to 211.
SubnetSelectionOption 211

# If the parameter is set but no value is defined, 118 is used.
SubnetSelectionOption

# Do not add any Subnet Selection Option.
#SubnetSelectionOption
```

3.116.13. UserClass

This optional parameter can be used to specify a User Class to be passed to the DHCP server to assist with address allocation. Any text and/or Radiator special characters are permitted. Refer to the DHCP server documentation for further details regarding the use of classes.

```
UserClass %{Client:Identifier}
```

3.116.14. ClientHardwareAddress

ClientHardwareAddress is an attribute in the incoming address which contains the hex encoded MAC address of the client. If present, it will be used as CHADDR in the DHCP request. If not present, and fake CHADDR based on the request XID will be used. The DHCP server may use this when allocating an address for the client. The MAC address can contain extraneous characters such as '.' or ':' as long as it contains the 12 hex characters (case insensitive) of the MAC address. Special characters are permitted.

```
# Use contents of an attribute to tell the DHCP server the
# hardware address of the client
ClientHardwareAddress %{Unisphere-Dhcp-Mac-Addr}
```

3.116.15. UseClassForAllocationInfo

When this optional parameter is set, information about the allocation is stored in reply attribute Class instead of in memory. Required for server farm where different farm members are likely to handle the allocation and deallocation. Defaults to off.

3.116.16. DHCPHostName

This string sets the value for DHCP option 12 "Host name". This has no default value and no Host Name is sent with DHCP request by default.

Here is an example of using *`DHCPHostName`*:

```
DHCPHostName Radiator-%{Calling-Station-Id}
```

3.116.17. DHCPVendorClass

This string parameter sets the value that is used for DHCP option 61 "Class Identifier", also known as "Vendor class identifier". This has no default value and no Class Identifier is sent with DHCP request by default.

Here is an example of using *`DHCPVendorClass`*:

```
DHCPVendorClass RadiatorAllocations
```

3.117. <AddressAllocator DHCPv6>

AddressAllocator DHCPv6 works in conjunction with <AuthBy DYNADDRESS> (see Section 3.53. <AuthBy DYNADDRESS> on page 241) and a DHCPv6 server to dynamically allocate IPv6 addresses, IPv6 delegated prefixes and to request other configuration information from the DHCPv6 server.

AddressAllocator DHCPv6 has been tested with a number of DHCPv6 servers, however it has been mostly tested the DHCP server from the Internet Software Consortium (ISC). For more information, see ISC website [http://www.isc.org/]. Version 4.3.3 or later is recommended.

AddressAllocator DHCPv6 makes the following allocation variables available for replies. The allocation variables names are case insensitive DHCPv6 option names without the leading 'OPTION_'. These names can be by default used in MapAttribute in AuthBy DYNADDRESS.

- iaaddr, the allocated IPv6 address from DHCPv6
- iaprefix, the delegated IPv6 prefix from DHCPv6
- dns_servers, the DNS servers' IPv6 addresses from DHCPv6
- serverid, the SERVERID option in hex format from DHCPv6
- clientid, the CLIENTID option in hex format as replied by DHCPv6

An example AddressAllocator DHCPv6 configuration file can be found in the distribution in the file `goodies/addressallocatordhcpv6.cfg`.

An example ISC DHCP server configuration file can be found in the distribution in the file `goodies/addressallocatordhcpv6-dhcpd.conf`.

CAUTION

Because AddressAllocator DHCPv6 binds to the DHCPv6 server address, it is not possible to run the DHCPv6 server on the same host as Radiator.

Note

PoolHint supplied in the AuthBy DYNADDRESS clause for the purposes of allocation is sent to the DHCPv6 server with OPTION_USER_CLASS. Some DHCPv6 servers may not be able to use this option. For this reason PoolHint is also sent with OPTION_CLIENTID. For more information, see Section 3.117.8. DHCPClientIdentifier on page 395.

Because the DHCPv6 address allocator binds to DHCPv6 client port 546, Radiator must be run as root, or at least with suitable privileges.

3.117.1. Identifier

This mandatory parameter specifies a symbolic name for this AddressAllocator clause. It must exactly match the Allocator parameter in an AuthBy DYNADDRESS in order for it to be used to allocate addresses.

```
Identifier        dhcpv6allocator
```

3.117.2. Host

The host name or IP address of the DHCP server. Hostname of the DHCP server to use. Defaults to All_DHCP_Relay_Agents_and_Servers (ff02::1:2). Radiator will allocate an address from the first DHCPv6 server to answer a solicit request.

```
# Use ALL_DHCP_Servers address
Host ff05::1:3
```

3.117.3. Port

The UDP port number on the DHCPv6 server. Defaults to port 547.

```
Port 67
```

3.117.4. ClientPort

The local UDP port number to send DHCPv6 messages from. Defaults to port 546.

```
Port 68
```

3.117.5. LocalAddress

The local IP address to bind to. Defaults to the IPv6 wildcard address. Consider using interface's link-local address.

Note

You can always use the link-local address of the interface for LocalAddress. Using link-local address is preferred and limits the set of addresses Radiator listens on to the smallest number of possibilities. The default value for LocalAddress is ::, the IPv6 wildcard address.

On Linux and macOS, you can configure 'LocalAddress ::'. On Windows, you need to set LocalAddress to the link-local address of the interface Radiator shares with the DHCPv6 server. Using global address may also work on all operating systems.

```
LocalAddress fe80::dead:beff:feef:1234
```

3.117.6. InterfaceName

InterfaceName defines the interface for sending DHCPv6 requests. There is no default.

InterfaceName currently works only on Linux. On Linux, InterfaceName value must be the interface shared with the DHCPv6 server.

InterfaceName or InterfaceIndex is needed to select the correct interface for link-local communication with the DHCPv6 server.

```
InterfaceName eth0
```

3.117.7. InterfaceIndex

On macOS and Windows you need to set InterfaceIndex. On macOS, the interface index numbering typically follows ‘ifconfig -a’ output with the first interface having index number 1. On Windows, run ‘netstat -nr’ to display the interface index numbers.

InterfaceName or InterfaceIndex is needed to select the correct interface for link-local communication with the DHCPv6 server.

```
InterfaceIndex 4
```

3.117.8. DHCPClientIdentifier

The Client Identifier to be used by the DHCPv6 server to track allocations. Sent by Radiator with OPTION_CLIENTID. Defaults to the user name in the current request followed by PoolHint: %{User-Name}-%{Reply:PoolHint}.

```
DHCPClientIdentifier %{User-Name}-pool-%{Reply:PoolHint}
```

3.117.9. DefaultLease

Specifies the default lease period requested. If SessionTimeout is set by a previous AuthBy then that is used as the expiry time. Defaults to one day (86400 seconds).

```
DefaultLease 86400
```

3.117.10. Synchronous

Normally DHCPv6 requests are processed asynchronously. This optional parameter can be used to make the requests synchronous.

```
Synchronous
```

3.117.11. UserClass

This optional parameter can be used to specify a User Class to be passed to the DHCP server to assist with address allocation. Any text and/or Radiator special characters are permitted. Refer to the DHCP server documentation for further details regarding the use of classes.

```
UserClass %{Client:Identifier}
```

3.117.12. AllocateDoneHook

AllocateDoneHook provides access to the received DHPCPv6 reply and allows modifying the results after Radiator has processed them, but before they are passed to AuthBy DYNADDRESS and made available for MapAttribute. The option values are in their native format. See the relevant RFCs for the details.

Note

This example requires domain_list is specified with MapAttribute. Radiator will not otherwise request OPTION_DOMAIN_LIST from DHCPv6.

```
# Domain list is in RFC 1035 format. Format it later.
AllocateDoneHook sub { my $result = $_[0]; my $reply = $_[1]; \
my @o = $reply->option($Radius::DHCPv6::OPTION_DOMAIN_LIST); \
```

```
$result->{domain_list} = $o[1] if $o[0]; }
```

3.118. <Resolver>

<Resolver> provides DNS and name resolution services for the *<AuthBy DNSROAM>* clause. For more information, see Section 3.75. <AuthBy DNSROAM> on page 303. Use *<Resolver>* only if you use *<AuthBy DNSROAM>* in your configuration. Other AuthBys do not require using *<Resolver>*.

<AuthBy DNSROAM> uses *<Resolver>* to do NAPTR, SRV, A, and AAAA lookups on a DNS name server in order to discover the name, address, and other possible attributes, such as protocol and whether to use TLS encryption, of a server that is used to handle requests for a certain Realm.

By default, *<Resolver>* consults DNS using the standard resolver configuration for your host. On Unix or Linux systems, it finds the resolver details by consulting `/etc/ resolv.conf`, `$HOME/.resolv.conf or ./.resolv.conf`. You can override these defaults and specify the used DNS name server, search path, and other options by using parameters in the Resolver clause.

<Resolver> requires the `Net::DNS` Perl module. Depending on the `Net::DNS` and Perl version, it may require the `Socket6` module and the `IO::Socket::INET6` module if you want to consult a DNS server via IPv6. These are all available as source from CPAN, or possibly as pre-built packages for your operating system or Perl distribution. For more information, see Section 2.1.2. CPAN on page 3.

<Resolver> uses the following algorithm to discover server names and addresses for a given Realm:

1. Look for NAPTR records for the Realm.
2. For each found NAPTR record, examine the Service field and use it to determine the transport protocol and TLS requirements for the server. The Service field starts with 'AAA' for insecure and 'AAAS' for TLS secured. The Service field contains '+RADSECS' for RadSec over SCTP, '+RADSECT' for RadSec over TCP or '+RADIUS' for RADIUS protocol over UDP. The most common Service field is 'AAAS +RADSECT' for TLS secured RadSec over TCP.
3. If the NAPTR has the 'S' flag, look for SRV records for the name. For each SRV record found, note the Port number and look for A and AAAA records corresponding to the name in the SRV record.
4. If the NAPTR has the 'A' flag, look for A and AAAA records for the name.
5. If no NAPTR records are found and DirectAddressLookup is enabled, look for A and AAAA records based directly on the realm name. For example, if the realm is 'examplerealm.edu', it looks for records such as '_radsec._tcp.examplerealm.edu', '_radsec._sctp.examplerealm.edu' and '_radius._udp.examplerealm.edu'.
6. All A and AAAA records found are ordered according to their Order and Preference fields. The most preferable server address is used as the target server address, along with any other server attributes discovered from DNS. If no SRV records was found for the address, the DNSROAM configured Port is used.

For example, if the user name is fred@example.com, the Realm is 'example.com', and DNS contains the following records:

```
example.com.IN NAPTR 50 50 "s" "AAAS+RADSECT" "" _radsec._tcp.example.com.
_radsec._tcp.example.com. IN SRV 0 10 2083 radsec.example.com.
radsec.example.com. IN AAAA 2001:db8::202:44ff:fe0a:f704
```

In the previous example, the selected target is a RadSec server on port 2083 at IPv6 address 2001:db8::202:44ff:fe0a:f704. The connection is made over TCP/IP, and TLS encryption is used. This complete specification of the realm is the most flexible and is recommended.

More concise DNS configurations are possible, too:

```
example.com. IN NAPTR 50 50 "a" "AAAS+RADSECS" "" radsec.example.com.
radsec.example.com. IN AAAA 2001:db8::202:44ff:fe0a:f704
```

In this case, the selected target is a RadSec server at IPv6 address 2001:db8::202:44ff:fe0a:f704. The connection is made over SCTP, and TLS encryption is used. The port used is the default Port configured into *<AuthBy DNSROAM>*.

The DNS can contain just this record:

```
_radius._udp.example.com. IN A 203.0.113.10
```

In this case, the selected target is a RADIUS server at IPv4 address 203.0.113.10. The connection is made over UDP. The Port and Secret used are the defaults configured into *<AuthBy DNSROAM>*.

Tip

The simplest Resolver clause you can have is:

```
<Resolver>
</Resolver>
```

This definition gets all its configuration from `/etc/resolv.conf` or the equivalent on your platform.

3.118.1. Nameservers

This optional parameter specifies the name or address of one or more DNS Name Servers passed to Net::DNS. Nameservers with IPv6 addresses are supported if your system supports IPv6. Defaults to the value of nameserver in `resolv.conf`. For more information, see Section 3.118. <Resolver> on page 396.

Note

Current Net::DNS implementations appear not to use multiple nameservers with the bgsend method Radiator uses. Only the first server is used for the query.

```
# Look first on one server and then another if it times out:
Nameservers 203.63.154.100
Nameservers 203.63.154.101
```

3.118.2. Debug

This optional flag enables debugging within the Net::DNS module. It will print to stdout the details of all DNS requests sent and replies received. Defaults to no debugging.

```
# Enable debug logging in Net::DNS:
Debug
```

If Radiator starts with `systemd`, you can make a local `systemd` configuration change to redirect `stdout` and `stderr` to a file. For the details, see Section 5.1. Systemd service unit file on page 444

3.118.3. Recurse

This optional flag enables recursive DNS lookups. Defaults to no recurse.

3.118.4. TCPTimeout

This optional flag specifies the timeout (in seconds) for DNS lookups over TCP connections. Defaults to 5 seconds.

3.118.5. UDPTimeout

This optional flag specifies the timeout (in seconds) for DNS lookups over UDP connections. Defaults to 5 seconds.

3.118.6. TCPPersistent

Note

This parameter is obsolete since Radiator 4.20. Multiple outstanding queries are not supported by Net::DNS when using persistent sockets.

This optional flag tells Net::DNS to keep a TCP socket open for each *`host:port`* to which it connects. This is useful if you are using TCP and need to make a lot of queries to the same nameserver. The default value is **`false`**.

3.118.7. UDPPersistent

Note

This parameter is obsolete since Radiator 4.20 Multiple outstanding queries are not supported by Net::DND when using persistent sockets.

This optional flag tells Net::DNS to keep a single UDP socket open for all DNS queries. This is useful if you are using UDP and need to make a lot of queries to the same nameserver. The default value is **`false`**.

3.118.8. GetIPV4

This optional flag specifies whether Resolver will attempt to find an IPv4 (A) address for any names it discovers. Defaults to true.

3.118.9. GetIPV6

This optional flag specifies whether Resolver will attempt to find an IPv6 (AAAA) address for any names it discovers. Defaults to true.

3.118.10. GetRadius

This optional parameter specifies whether Resolver is required to attempt to discover RADIUS servers from

- NAPTR records of type AAA with +RADIUS
- IPv4 (A) or IPv6 (AAAA) records for '_radius._udp.realmname'

Defaults to true.

3.118.11. GetRadSec

This optional parameter specifies whether Resolver is required to attempt to discover RADSEC servers from

- NAPTR records of type AAA or AAAS with +RADSECS or +RADSECT or
- IPv4 (A) or IPv6 (AAAA) records for '_radsec._sctp.realmname'
- IPv4 (A) or IPv6 (AAAA) records for '_radsec._tcp.realmname'

Defaults to true.

3.118.12. DirectAddressLookup

If DirectAddressLookup is enabled, and if there are no NAPTR records for the requested Realm, Resolver will attempt lookups of A and AAAA records for _radsec._ sctp.<REALM>, _radsec._tcp.<REALM> and _radius._udp.<REALM> Enabled by default.

3.118.13. NAPTR-Pattern

NAPTR-Pattern is an optional parameter that specifies a regexp that will be used to match the contents of NAPTR records during Resolver service discovery.

If NAPTR-Pattern is defined and matches a NAPTR DNS record, it will be used to determine the protocol and transport to be used. The regexp is expected to match 2 substrings. The first is the protocol and can be 'radsec' or 'radius'. The second is the transport to use, and can be 'tls', 'tcp' or 'udp'.

3.118.14. NegativeCacheTtl

This optional value specifies how long a negative lookup (ie failure to resolve the realm) will be cached until another lookup will be made. Defaults to 21600 seconds (6 hours).

3.118.15. FailureBackoffTime

If the lookup failed to discover any results and there was a timeout when waiting for the nameserver, this optional value specifies how long Radiator will wait before another lookup will be made. This will give the target server time to become available without resorting to potentially long NegativeCacheTtl. Defaults to 3 seconds.

3.119. <ServerDIAMETER>

This clause tells Radiator to act as a Diameter to RADIUS gateway.

Diameter is an AAA protocol described in RFC 6733 (and others). It provides for TCPIP or SCTP transport and TLS encryption. It makes specific provision for carrying RADIUS compatible requests and defines Diameter to RADIUS gateways. ServerDIAMETER implements such a Diameter to RADIUS gateway.

Incoming Diameter requests are converted as far as possible into RADIUS requests and then dispatched internally within Radiator. A Realm or Handler can be configured to handle the request either locally or proxy it (as a RADIUS request) to another RADIUS server. RADIUS replies are sent back to the originating Diameter peer. Handlers may use the Client-Identifier to match requests received by a particular `<ServerDIAMETER>` clause.

By default, `<ServerDIAMETER>` listens for connections from Diameter peers on TCP port 3868. By default it does not require TLS encryption of the Diameter connection. `<ServerDIAMETER>` never contacts a Diameter peer by itself: it always acts only as a Diameter server.

By default, *<ServerDIAMETER>* uses a hardwired internal dictionary to translate Diameter requests into readable parameters. You can use the global configuration parameter DiameterDictionaryFile to alter the hardwired internal dictionary.

<ServerDIAMETER> supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

3.119.1. Port

This optional parameter specifies which network port ServerDIAMETER will listen on for connections from Diameter peers. Defaults to 3868, the official IANA port number for Diameter. May be a numeric port number or symbolic port/service name.

3.119.2. BindAddress

This optional parameter specifies one or more network interface addresses to listen for incoming connections. It is only useful if you are running Radiator on a multi-homed host (a host that has more than one network address). The default value is the global *BindAddress*, which defaults to **0.0.0.0**. It listens to all networks connected to the host. For more information, see Section 3.7. Global parameters on page 29.

Using this parameter, you can run multiple instances of Radiator on the one computer, where each instance listens to connections directed to a different host address. *BindAddress* can include special formatting characters, and multiple comma separated IPv4 and IPv6 addresses.

When SCTP multihoming is supported, all addresses defined with *BindAddress* must be either IPv4 or IPv6 addresses. Radiator binds all addresses to one listen socket instead of creating multiple listen sockets.

Here is an example of using *BindAddress*:

```
# Only listen on one IPv4 address and the IPv6 loopback
BindAddress 203.63.154.1, ::1
```

3.119.3. Protocol

This optional parameter specifies which Stream protocol will be used to carry Diameter. Options are 'tcp' for TCP/IP or 'sctp' for SCTP (Stream Control Transmission Protocol). Defaults to 'tcp'. Not all hosts are able to support 'sctp': consult your vendor. The protocol setting must be the same as that being used by connecting Diameter peers.

```
Protocol sctp
```

Tip

On modern Linux hosts, SCTP support is in a loadable module, and can be enabled with:

```
modprobe sctp
```

3.119.4. ReadTimeout

This optional parameter specifies the maximum time to wait for incoming Diameter connections to complete their initial handshaking. Defaults to 10 seconds. If a Diameter CER message is not received from the peer by ServerDIAMETER within this time period, the connection will be shut down.

3.119.5. OriginHost

This parameter specifies the name that <ServerDIAMETER> uses to identify itself to any connecting Diameter peers. It is sent to the peer in the Diameter CER message. It is not optional and must be specified in the <ServerDIAMETER> clause. Diameter peers may use OriginHost to determine whether they have connected to the correct peer, so it may be critical that it be configured correctly. Special formatting characters are supported.

3.119.6. OriginRealm

This parameter specifies the name of the user Realm that <ServerDIAMETER> is willing to handle. It is sent to connecting Diameter peers in the CER message, and the peer uses it to determine which requests are routed to this <ServerDIAMETER>. It is not optional an must be specified in the <ServerDIAMETER> clause. Special formatting characters are supported.

3.119.7. ProductName

This optional parameter is used to identify the product name of this Diameter peer. It is sent to connecting Diameter peers in the CER message. It defaults to 'Radiator'.

3.119.8. AddToRequest

This optional parameter is used to add extra RADIUS attributes to the RADIUS request generated from each incoming Diameter request. It can be used to tag requests arriving from ServerDIAMETER for special handling within Radiator or in remote RADIUS servers.

```
AddToRequest NAS-Identifier=DIAMETER
```

3.119.9. DefaultRealm

This optional parameter can be used to specify a default realm to use for received Diameter requests that have a user name that does not include a realm. If the incoming user name does not have a realm (i.e. there is no @something following the user name) and if DefaultRealm is specified, the User-Name in the resulting RADIUS request will have @defaultrealm appended to it. The realm can then be used to trigger a specific <Realm> or <Handler> clause. This is useful if you operate a number of Diameter peers for different customer groups and where some or all of your customers log in without specifying a realm.

```
# Realmless logins to this NAS will be treated
# as if they are for realm open.com.au
<ServerDIAMETER>
      OriginHost ....
      DefaultRealm open.com.au
</ServerDIAMETER>
<Realm open.com.au>
      .....
</Realm>
```

3.119.10. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  PreHandlerHook is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

    PreHandlerHook specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

    If *EAP_LEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-MSCHAPv2

    If *EAP_PEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-GTC

    If *EAP_GTC_PAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  PreHandlerHook is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  PreHandlerHook is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  PreHandlerHook is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  PreHandlerHook is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's *PrehandlerHook* is run before ServerTACASPLUS's *PreHandlerHook*. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

PreHandlerHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *PreHandlerHook*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.119.11. SupportedVendorIds

This optional parameter allows you to define the Supported Vendor Ids announced in CER. There is no default and no Supported-Vendor-Id is announced by default. Keyword "DictVendors" is an alias group for all vendors in the default dictionary and the dictionary file configured with DiameterDictionaryFile.

```
# Tell the peer we support all the vendors in our
# default and DiameterDictionaryFile dictionaries
SupportedVendorIds DictVendors
```

3.119.12. ConvertCommand

This optional parameter defines a conversion from Diameter Command Code to RADIUS Code. Currently the only conversion available is converting STR to Accounting-Request with Acct-Status-Type set to Stop. By default no optional conversions are done. Any unknown ConvertCommand conversions are logged, and later ignored, when this module is loaded.

```
# Specify one ConvertCommand for each conversion
ConvertCommand STR,Accounting-Request
```

3.119.13. AuthApplicationIds

This optional parameter allows you to define the Auth Application Ids announced in CER. Defaults to '0, 1, 5' (i.e. DIAMETER BASE, NASREQ and Diameter-EAP).

```
AuthApplicationIds 0, 1
```

3.119.14. AcctApplicationIds

This optional parameter allows you to define the Acct Application Ids announced in CER. Defaults to ‘3’ (i.e. BASE_ACCOUNTING).

```
AcctApplicationIds 3
```

3.119.15. MaxBufferSize

This optional advanced parameter specifies the maximum number of octets that are output and input buffered by Radiator's Stream modules. This advanced parameter usually does not need adjusting.

3.119.16. DisconnectTraceLevel

This optional parameter specifies log trace level for peer initiated disconnects. The default value is error level **`0`**. When connections are known to be short-lived, a non-default value may be useful. This parameter is available for all Stream based modules, such as *`<ServerDIAMETER>`* and *`<AuthBy RADSEC>`*.

```
# Debug logging is enough for peer disconnects
DisconnectTraceLevel 4
```

3.119.17. StreamStateChangeHook

This optional parameter allows you to define a Perl function that will be called when a Stream connection state to a peer changes. RadSec, Diameter, TACACS+, HTTP and SIGTRAN clients and servers use Stream connections in Radiator. This is a low layer hook that runs before, for example, RadSec TLS handshake or Diameter Capabilities Exchange messages are exchanged. The following arguments are passed in the following order:

- Reference to this `Radius::Stream` derived type. Examples of derived types are `Radius::RadsecHost` and `Radius::DiameterConnection`.
- New stream state. One of integer constant values:
  - `$Radius::Stream::STREAM_STATE::CONNECTED`
  - `$Radius::Stream::STREAM_STATE::DISCONNECTED`

`StreamStateChangeHook` can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. IP address and other specific information is not passed to the hook. Their format and type depends on the derived type. For example, SCTP streams may have multiple source and destination addresses. The following example shows how to log information from the hook no matter what the type of the stream object is.

```
StreamStateChangeHook sub { \
    my ($self, $new_state) = @_; \
    my $state = ($new_state == $Radius::Stream::STREAM_STATE::CONNECTED) ? \
                 'connected' : 'disconnected'; \
    main::log($main::LOG_INFO, "StreamStateChangeHook: State change to $state"); \
    return; }
```

3.119.18. StreamMaxClients

This optional parameter specifies the maximum number of accepted connections for each listen socket. This parameter is not set by default. When the parameter is set to zero or unset, no restrictions apply. This parameter is available for all StreamServer based modules, such as *<ServerDIAMETER>* and *<ServerRADSEC>*.

With server farm, see FarmSize on page 42, this parameter allows distributing all incoming connections equally between worker processes. For example, if the number of connections is known not to exceed 10, a farm of 12 workers will have 2 spare workers with each of the 10 remaining workers handling one connection each.

Note

Currently this parameter is used as follows: All values larger than 0 are treated as 1.

```
# Allow one connection for each listen socket
StreamMaxClients 1
```

3.119.19. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with *`PacketTrace`* set off or **`0`**.

`PacketTrace` is available for the following clauses:

- *`Client`*
- *`Handler`*
- *`Realm`*
- *`AuthBy`*
- *`ServerDIAMETER`*

- *ServerRADSEC*
- *ServerTACACSPLUS*

Here is an example of using *PacketTrace*:

```
# Debug any packets that pass through here
PacketTrace
```

3.119.20. PostDiaToRadiusConversionHook

This optional parameter allows you to define a Perl function that will be called during packet processing. PostDiaToRadiusConversionHook is called after an incoming Diameter request has been converted to its equivalent RADIUS request, allowing you to alter or add to attribute conversions etc. It is passed references to the incoming Diameter request and the converted RADIUS request.

3.119.21. PostRadiusToDiaConversionHook

This optional parameter allows you to define a Perl function that will be called during packet processing. PostRadiusToDiaConversionHook is called after an RADIUS reply has been converted to its equivalent Diameter reply, prior to being sent back to the Diameter client. It is passed references to the RADIUS reply and the converted Diameter reply.

3.119.22. Clients

This optional parameter specifies a list of IP addresses that connections will be accepted from. You can specify one or more comma or space separated IP addresses on each Client line. You can specify multiple Client parameters. Only exact matches are supported at present. The default is to accept connections from any and all clients.

If Clients is specified and a client attempts to connect from an IP address that is not named, Radiator will log a WARNING level message then reject and close the connection.

```
# Only accept connections from some addresses
Clients 127.0.0.1, 203.63.154.29
Clients 203.63.154.27
```

3.120. <ServerTACACSPLUS>

This clause tells Radiator to act as a TACACS+ server as well as a RADIUS server.

TACACS+ is an Authentication, Authorisation and Accounting (AAA) protocol developed by Cisco and supported by devices from Cisco, Juniper, HP and other vendors. It uses TCP connections between the client device and the TACACS+ server. Newer Cisco devices generally support both RADIUS and TACACS+ or just RADIUS. Some older Cisco devices only support TACACS+. In addition, there are some Cisco security facilities that are only available through TACACS+.

The *<ServerTACACSPLUS>* clause handles TACACS+ AAA requests in the following way:

- Incoming TACACS+ requests are decrypted using a key from a matching Client or this *<ServerTACACSPLUS>* clause. The search function looks for a matching Client using this order of preference:
  1. Exact IP address
  2. CIDR address
  3. DEFAULT Client clause

Note

Radiator 4.13 and earlier looked for DEFAULT Client clause before CIDR match.

When the Client search has finished, the decryption key is chosen using this order of preference. See Key on page 408 for additional details:

1. If a Client is found and its `TACACSPLUSKey` is defined, it is used as the key.
2. If `<ServerTACACSPLUS> Key` is defined, it is used as the key.
3. If a Client was found, and its `Secret` is defined, it is used as the key.
4. If there still is no key found, a warning is logged and TACACS+ message decryption fails.

- TACACS+ authentication requests are converted into RADIUS Access-Request requests. TACACS+ ASCII, PAP, CHAP and MSCHAP are supported, and the resulting passwords are converted into the appropriate RADIUS attributes. The `action`, `priv_lvl`, `authen_type`, and `service` values in the TACACS+ authentication request are converted to `OSC-TACACS-Action`, `OSC-TACACS-Privilege-Level`, `OSC-TACACS-Authen-Type`, and `OSC-TACACS-Service` RADIUS attributes. The RADIUS Access-Request is dispatched to a matching `<Realm>` or `<Handler>` clause, which serve the request locally using any of the Radiator AuthBy clauses or proxy the request to another RADIUS server.
- TACACS+ authorisation requests are approved subject to any `AuthorizeGroupAttr` reply items from the previous RADIUS Access-Accept and `AuthorizeGroup` parameters from the configuration file. In addition, any cisco-avpair reply items from the previous RADIUS Access-Accept are used as authorisation attribute-value pairs. TACACS+ authorisation requests can optionally be converted to RADIUS Access-Requests.
- TACACS+ accounting requests are converted into RADIUS Accounting-Requests.
- Global `RewriteUsername` rules are applied. For more information, see Section 3.7.29. RewriteUsername on page 37. Client `RewriteUsername` can also be used. See below for more information about how `<ServerTACASCSPLUS>` uses Client clauses.
- `%{Client:name}` format and `Client-Identifier` check item are made available. The values are from a matching Client clause or ServerTACACSPLUS clause when no Client clause matched.
- The converted RADIUS requests are dispatched to a matching `<Realm>` or `<Handler>` clause, using the same rules as RADIUS requests.

Tip

By default, Radiator requires that the user has previously authenticated with this Radiator instance before accepting any TACACS+ authorisation requests. The authentication reply provides the authorisation information Radiator needs to handle the subsequent TACACS+ authorisation requests. If the optional parameter `AllowAuthorizeOnly` is enabled, Radiator requests authorisation information even if the user has not previously authenticated with this Radiator instance. For more information, see Section 3.120.13. AllowAuthorizeOnly on page 413.

Tip

If `<ServerTACACSPLUS>` is used to authenticate administrator access to a Cisco device, you need to add specific authorisation attributes to allow administrative access. For example, to get administrative access

to a Cisco Aironet wireless Access Point requires that the authorisation include a TACACS+ attribute-value pair like:

```
aironet:admin-capability=ident+admin
```

You can achieve this by having a suitable cisco-avpair reply item for the relevant user in your Radiator database:

```
ciscouser User-Password=fred
      cisco-avpair="aironet:admin-capability=ident+admin"
```

You can also achieve this by having an AuthorizationAdd parameter in your <*ServerTACACSPLUS*> clause:

```
AuthorizationAdd aironet:admin-capability=ident+admin
```

This example enables only some levels of administrative access (ident and admin). See your Cisco device documentation for more details.

Tip

Devices from other vendors than Cisco, such as Juniper, may also accept ciscoavpair attributes.

<*ServerTACACSPLUS*> can be used with any Radiator authentication method that understands plain text passwords, such as FILE, SQL, LDAP2, DBFILE, and also with any method that challenge the user for additional authentication data,such as DIGIPASS, ACE, OTP, INTERNAL.

Tip

You can use the TACACS+ test client `goodies/tacacsplustest` in your distribution to send test TACACS+ requests.

During request processing, <*Server TACACSPLUS*> looks for a Client clause that matches the origin of the TACACS+ request, as described above. If found, a number of parameters from the Client clause are used during processing:

- *TACACSPLUSKey*
- *Secret*
- *RewriteUsername*
- *StripFromRequest*
- *AddToRequest*
- *AddToRequestIfNotExist*
- *PreHandlerHook*
- *DefaultRealm* (overrides *DefaultRealm* in <*ServerTACACSPLUS*>)

3.120.1. Key

This parameter specifies the default shared secret to be used to decrypt TACACS+ messages. When a new connection from a TACACS+ client is received, *<ServerTACACSPLUS>* tries to find a key to use for decrypting that connection. It first looks for a matching Client and then for a key until it finds one that has been defined:

- If a matching Client is found: *EncryptedTACACSPLUSKey* parameter is preferred over *TACACSPLUSKey* parameter
- *EncryptedKey*
- This *Key* parameter
- If a matching Client is found: *EncryptedSecret* parameter is preferred over *Secret* parameter

---

Note

EncryptedTACACSPLUSKey and *EncryptedSecret* are currently experimental and will be documented later.

Tip

If all your TACACS+ devices use the same key, use this *Key* parameter. If some or all of your TACACS + devices use different keys, define a Client and *TACACSPLUSKey* for each differing one and set this *Key* as the default for the rest. If some TACACS+ clients are also RADIUS clients, define a Client clause for each one, specifying the RADIUS secret in *Secret*, and the TACACS+ key in *TACACSPLUSKey*.

```
Key mysecret
```

3.120.2. EncryptedKey

This parameter specifies the default shared secret to be used to decrypt Tacacs+ messages. When a new connection from a Tacacs+ client is received, ServerTACACSPLUS tries to find a Key to use for decrypting that connection. EncryptedKey is in encrypted format and is preferred over Key. For more information, see Section 3.120.1. Key on page 408.

EncryptedKey is currently experimental and will be fully documented later.

3.120.3. Port

This optional parameter specifies which TCP port the server will listen on for incoming Tacacs+ connections. Defaults to 49 (which generally requires root or other privileged access) Any valid port number or service name can be used.

```
Port 1024
```

3.120.4. BindAddress

This optional parameter specifies one or more network interface addresses to listen for incoming Tacacs+ connections on. It is only useful if you are running Radiator on a multi-homed host (i.e. a host that has more than one network address). Defaults to the global BindAddress, which defaults to 0.0.0.0 (i.e. listens on all networks connected to the host). For more information, see Section 3.7.9. BindAddress on page 33.

Using this parameter, you can run multiple instances of Radiator on the one computer, where each Radiator listens to Tacacs+ connections directed to a different host address. BindAddress can include special formatting characters, and multiple comma separated IPv4 and IPv6 addresses.

```
# Only listen on one IPv4 address and the IPv6 loopback
BindAddress 203.63.154.1, ::1
```

3.120.5. AuthorizationReplace

This optional parameter specifies Tacacs+ authorisation attribute-value pairs that are to replace those suggested by the Tacacs+ client. It effectively overrides the default authorisation that the client would use.

You can have as many *`AuthorizationReplace`* parameters as you wish, one for each Tacacs+ authorisation attribute-value pair:

```
AuthorizationReplace service=aironet
AuthorizationReplace protocol=shell
AuthorizationReplace aironet:admin-capability=ident+admin
```

3.120.6. AuthorizationAdd

This optional parameter specifies Tacacs+ authorisation attribute-value pairs that are to be added to those suggested by the Tacacs+ client. It effectively increases the default authorisation that the client would use.

You can have as many *`AuthorizationAdd`* parameters as you wish, one for each Tacacs+ authorisation attribute-value pair:

```
AuthorizationAdd aironet:admin-capability=ident+admin
```

3.120.7. AddToRequest

This optional parameter adds any number of RADIUS attributes to the RADIUS requests generated by ServerTACACSPLUS. It can be used to tag requests arriving from Tacacs+ for special handling within Radiator or in remote RADIUS servers.

```
AddToRequest NAS-Identifier=TACACS
```

3.120.8. AuthorizationTimeout

This optional parameter changes the timeout period for handling a complete TACACS+ conversation, including the authentication any subsequent authorisation requests. Defaults to 600 seconds. If the timeout expires, further authorisations for an earlier authentication will not be valid, and will be rejected.

3.120.9. DefaultRealm

This optional parameter can be used to specify a default realm to use for received TACACS requests that have a user name that does not include a realm. If the incoming user name does not have a realm (i.e. there is no @something following the user name) and if DefaultRealm is specified, the User-Name in the resulting RADIUS request will have @defaultrealm appended to it. The realm can then be used to trigger a specific <Realm> or <Handler> clause. This is useful if you operate a number of TACACS clients for different customer groups and where some or all of your customers log in without specifying a realm.

Tip

You can override this on a per-client basis by setting DefaultRealm in the Client clause.

```
# Realmless logins to this NAS will be treated
# as if they are for realm open.com.au
<ServerTACACSPLUS>
      Key ....
      DefaultRealm open.com.au
</ServerTACACSPLUS>
<Realm open.com.au>
      .....
</Realm>
```

3.120.10. GroupMemberAttr

When AuthorizeGroup is use to specify TACACS+ user privileges, GroupMemberAttr specifies the name of the RADIUS reply attribute in the Access-Accept that is expected to contain the name of the TACACS+ users privilege group. This group name will then be used by AuthorizeGroup to determine which privileges can be extended to that user. If there is no such attribute in the Access-Accept, the TACACS+ group name for the user will be assumed to be 'DEFAULT'. If GroupMemberAttr is not defined in the configuration file, then all TACACS+ users will be assumed to have a TACACS+ group name of 'DEFAULT'.

The RADIUS attribute named by GroupMemberAttr may be a real RADIUS attribute received from a remote RADIUS server (in the case where the remote RADIUS server provides the authentication of TACACS+ requests). Or it could be pseudo RADIUS attribute added to the reply by an AuthBy internal to the current Radiator server.

```
# Name of the pseudo attribute containing the TACACS group name
# in RADIUS Access-Accepts:
GroupMemberAttr tacacsgroup
```

3.120.11. AuthorizeGroup

Some TACACS+ clients request per-command authorisation of commands from the TACACS+ server. When this occurs, one or more *AuthorizeGroup* parameters can be used to specify privilege levels, permitted TACACS commands, and TACACS restrictions for various TACACS+ privilege groups. If no *AuthorizeGroup* parameters are specified in the Radiator configuration file, all TACACS+ commands are authorised by *<Server TACSCPLUS>*.

You must have a thorough and complete understanding of TACACS+ command authorisation and how to configure your TACACS+ client for command authorisation before attempting to configure privilege control with *AuthorizeGroup*.

The general format of the *AuthorizeGroup* parameter is this:

```
# Note: Whitespace is not allowed after and before {}
AuthorizeGroup <groupname> <permit|permitreplace|deny> pattern1
pattern2 ... {attr1=val attr2=val ...} {extra_check1=val extra_check2="val 2" ...}
```

If any extra checks exist, they are also evaluated for a match. Currently supported extra checks are:

- *peeraddr*
- Client-Identifier
- Any real attribute from Access-Accept
- Any pseudo attribute from Access-Accept

Each *AuthorizeGroup* parameter specifies a privilege rule for a TACACS+ privilege group. You can specify zero or more *AuthorizeGroup* parameters for each privilege group that is used in your organisation. The *AuthorizeGroup* parameters are considered for a group in the order in which they are given in the Radiator configuration file.

When a TACACS authorisation request is received, the list of *AuthorizeGroup* rules is searched for rules matching the group name identified by the *GroupMemberAttr* attribute. For more information, see Section 3.120.10. GroupMemberAttr on page 410. Each rule is examined in order until a matching rule is found. The patterns are used to do the matching. Each pattern is a Perl Regular Expression (regexp). Pattern1 is matched against the first TACACS+ request argument (usually *service=xyz*), pattern2 is matched against the second TACACS+ request argument and so on. If every pattern matches its TACACS+ argument, then the rule matches.

Tip

For example *service=shell* is a single request argument. The pattern to match two values for service is *service=(shell|exec)*.

If the first matching rule is *deny*, the authorisation is denied.

If the first matching rule is *permit*, the request is authorised, and the list of reply *attr=val* entries are sent back to the TACACS+ client to be added to the user's command arguments.

If the first matching rule is *permitreplace*, the request is authorised, and the list of reply *attr=val* entries are sent back to the TACACS+ client and are used to replace the user's requested command arguments.

Tip

Some Cisco devices have authorisation for optional roles with the * character. This is not a wildcard character, but indicates the following is optional. For example, the following rule automatically selects the right roles depending on the *shell:contextname* in the authorisation request. Only the set of roles corresponding to the requested *contextname* is returned.

```
AuthorizeGroup group1 permit service=shell \
                             {shell:contextname1*"role1 role2 role3" \
                              shell:contextname2*role2}
```

Tip

AuthorizeGroup replaces the old *CommandAuth* parameter. Support for *CommandAuth* will be removed some time in the future.

To provide per-user privilege control for TACACS+ users:

1. Divide your TACACS+ users into groups where all the members in the group have the same privileges.
2. Assign a unique name to each group.
3. Choose a RADIUS Reply attribute that will contain the group name for each user. Set this attribute name in *GroupMemberAttr*.

4. Arrange for each user to have their specific group name (from step 1) in their RADIUS Reply attribute selected in the previous step.
5. Add one or more *AuthorizeGroup* parameters specifying the privileges for all of the possible group names from step 1 above.
6. Configure your TACACS+ client to perform per-command authorisation with its TACACS+ server.
7. Test that your TACACS+ client enforces the correct privileges specified in step 5 for each user group.

Example

This example uses two TACACS groups. Group1 is only permitted to do show commands but group2 is allowed to do anything. Group1 is allowed to start a PPP IP session, which gets an inacl of 101 and an outacl of 102.

```
AuthorizeGroup group1 permit service=shell cmd=show cmd-arg=.*
AuthorizeGroup group1 permit service=ppp protocol=ip {inacl=101 outacl=102}
AuthorizeGroup group1 deny .*
AuthorizeGroup group2 permit .*
```

Example

This example uses extra checks to allow or deny *show* commands for *group1*. Users coming from client that connects from 127.0.0.1 are explicitly permitted. Users coming from client that has identifier *Some-TACACSPLUS-client* are explicitly denied. The rest of the users are permitted.

```
AuthorizeGroup group1 permit service=shell cmd=show cmd-arg=.* {} \
                             {peeraddr=127.0.0.1}
AuthorizeGroup group1 deny service=shell cmd=show cmd-arg=.* {} \
                             {Client-Identifier="Some-TACACSPLUS-client"}
AuthorizeGroup group1 permit service=shell cmd=show cmd-arg=.*
```

Example

This example shows how to match a command with multiple arguments. For example: `logout subscribers username testuser`

Here the last parameter, the target username, varies between commands. Therefore only fixed arguments are configured with *AuthorizeGroup*.

```
AuthorizeGroup group3 permit service=shell cmd=logout \
      cmd-arg=subscribers cmd-arg=username
```

Example

As an alternative to controlling individual command authorisation, you can set a privilege level for the user when they start their exec session. Thereafter, the router limits which command the user can use, depending on the priv-lvl. 0 is the lowest, and permits disable, enable, exit, help, and logout. priv-lvl=1 is the non-privileged user. priv-lvl=15 is the highest privilege level, the level after going into enable mode. Here users in group 3 get a priv-lvl of 15. The start of a session sends the args 'service=shell cmd*'.

```
AuthorizeGroup group3 permit service=shell cmd\* {priv-lvl=15}
```

For more examples, see `goodies/tacacsplusserver.cfg`.

3.120.12. AuthorizeGroupAttr

If this parameter is specified, it specifies the name of an attribute in Access-Accept that will contain per-command authorisation patterns for authorising TACACS+ commands for this user. Multiple attributes are supported with each attribute containing one pattern. The format is the same as the *AuthorizeGroup* parameter above excluding the group name. These patterns are processed before any configured-in *AuthorizeGroup* parameters.

As an example, a users file to grant group1 member ‘mikem’ an additional right to use ping command would look like below. The Radiator <ServerTACACSPLUS> clause is configured with *GroupMemberAttr OSC-Group-Identifier* and *AuthorizeGroupAttr OSC-Authorize-Group*.

```
mikem User-Password=fred
      OSC-Group-Identifier = group1,
      OSC-Authorize-Group = "permit service=shell cmd=ping"
```

3.120.13. AllowAuthorizeOnly

When enabled, this parameter allows Radiator to create a RADIUS Access-Request with Service-Type attribute set to Authorize-Only when TACACS+ authorisation request is received but Radiator has no previous information about the user's authorisation. This can happen if the TACACS+ client does not use TACACS + for authentication, has authenticated against another TACACS+ server, Radiator has been reloaded, or *AuthorizationTimeout* has expired. This is disabled by default.

For example, Cisco 'aaa new model' allows non-TACACS+ authentication with TACACS+ based accounting and authorisation: you can authenticate with local user name, Radius, or kerberos and then do command authorisation over TACACS+.

The default for Radiator is to require TACACS+ authentication first to create the authorisation context before being able to do command authorisation. If *AllowAuthorizeOnly* is enabled, an existing authorisation context is not required.

Before enabling this option, we recommend considering if it is acceptable to trust the TACACS+ client authentication and allow Radiator to do command authorisation without any previous knowledge about the users' authentication.

3.120.14. UsernamePrompt

This optional parameter sets the prompt that ServerTACSPLUS will use to prompt the client for a user name when the Tacacs authen-type of ASCII is used. Defaults to ‘Username: ‘.

3.120.15. PasswordPrompt

This optional parameter sets the prompt that ServerTACSPLUS will use to prompt the client for a password when the Tacacs authen-type of ASCII is used. Defaults to ‘Password: ‘.

3.120.16. IdleTimeout

If a TACACS+ client stays connected for more than this number of seconds without sending any requests it will be disconnected. Defaults to 180 seconds. A Value of 0 means no idle timeout will apply, and clients can stay connected forever.

3.120.17. AuthenticationStartHook

Specifies a Perl hook to run when a TACACS+ Authentication Start is received. It can be used for special processing of TACACS+ Start requests. If the hook returns an empty list, normal processing of the request will continue else no further processing will be done and the hook is expected to handle the request.

The hook is passed the following arguments:

- Pointer to the current `TacacsplusConnection` object
- Pointer to the request object that has been constructed to represent the TACACS+ authentication request
- The action attribute from the incoming TACACS+ request. This describes the authentication action to be performed. One of `$Radius::Tacacsplus::TAC_PLUS_- AUTHEN_*`.
- The authen_type attribute from the incoming TACACS+ request. The type of authentication that is being performed. One of `$Radius::Tacacsplus::TAC_PLUS_- AUTHEN_TYPE_*`.
- The priv_lvl attribute from the incoming TACACS+ request. This indicates the privilege level that the user is authenticating as. One of `$Radius::Tacacsplus:: TAC_PLUS_PRIV_LVL_*`.
- The service attribute from the incoming TACACS+ request. This is the service requesting the authentication. One of `$Radius::Tacacsplus::TAC_PLUS_AUTHEN_SVC_*`.

3.120.18. AuthenticationContinueHook

Specifies a Perl hook to run when a TACACS+ Authentication Continue is received. It can be used for special processing of TACACS+ Continue requests. If the hook returns an empty list, normal processing of the request will continue else no further processing will be done and the hook is expected to handle the request.

The hook is passed the following arguments:

- Pointer to the current `TacacsplusConnection` object
- The body of the request a received in the incoming TACACS+ CONTINUE request. Contains the undecoded flags and fields from the request.

3.120.19. AccountingHook

Specifies a Perl hook to run when a TACACS+ Accounting Request message is received. It can be used for special processing of TACACS+ accounting requests. The hook return value is ignored. See goodies files `create-cisco-cmd.pl` and `createavpairs.pl` for sample hooks.

The hook is passed the following arguments:

- Pointer to the current TacacsplusConnection object
- Pointer to the request object that has been constructed to represent the TACACS+ accounting request

3.120.20. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  `PreHandlerHook` is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

  `PreHandlerHook` specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

If *EAP_LEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

– EAP-MSCHAPv2

If *EAP_PEAP_MSCHAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

– EAP-GTC

If *EAP_GTC_PAP_Convert* flag is set, *PreHandlerHook* specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  PreHandlerHook is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  PreHandlerHook is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  PreHandlerHook is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  PreHandlerHook is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's *PrehandlerHook* is run before ServerTACASPLUS's *PreHandlerHook*. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

PreHandlerHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *PreHandlerHook*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.120.21. SingleSession

This optional parameter tells the server to try to maintain a single TCP connection for all TACACS+ requests from the same client if the client permits. If this flag is set and if the TACACS+ client indicates a willingness to support single connections with the TAC_PLUS_SINGLE_CONNECT_FLAG, the TCP connection will not be closed by Radiator after each TACACS+ session. Defaults to 1.

Single sessions can be disabled regardless of client options by setting the SingleSession flag to 0.

Note

Default behaviour changed in 4.8. In versions of Radiator prior to 4.8, the default behaviour was to always try to keep the session open.

3.120.22. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with *PacketTrace* set off or **0**.

PacketTrace is available for the following clauses:

- *Client*
- *Handler*
- *Realm*
- *AuthBy*
- *ServerDIAMETER*
- *ServerRADSEC*
- *ServerTACACSPLUS*

Here is an example of using *PacketTrace*:

```
# Debug any packets that pass through here
PacketTrace
```

3.120.23. DisconnectTraceLevel

This optional parameter specifies log trace level for peer initiated disconnects. The default value is error level **0**. When connections are known to be short-lived, a non-default value may be useful. This parameter is available for all Stream based modules, such as *<ServerDIAMETER>* and *<AuthBy RADSEC>*.

```
# Debug logging is enough for peer disconnects
DisconnectTraceLevel 4
```

3.120.24. StreamStateChangeHook

This optional parameter allows you to define a Perl function that will be called when a Stream connection state to a peer changes. RadSec, Diameter, TACACS+, HTTP and SIGTRAN clients and servers use Stream connections in Radiator. This is a low layer hook that runs before, for example, RadSec TLS handshake or Diameter Capabilities Exchange messages are exchanged. The following arguments are passed in the following order:

- Reference to this `Radius::Stream` derived type. Examples of derived types are `Radius::RadsecHost` and `Radius::DiameterConnection`.
- New stream state. One of integer constant values:
  - `$Radius::Stream::STREAM_STATE::CONNECTED`
  - `$Radius::Stream::STREAM_STATE::DISCONNECTED`

StreamStateChangeHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. IP address and other specific information is not passed to the hook. Their format and type depends on the derived type. For example, SCTP streams may have multiple source and destination addresses. The following example shows how to log information from the hook no matter what the type of the stream object is.

```
StreamStateChangeHook sub { \
    my ($self, $new_state) = @_; \
    my $state = ($new_state == $Radius::Stream::STREAM_STATE::CONNECTED) ? \
                  'connected' : 'disconnected'; \
    main::log($main::LOG_INFO, "StreamStateChangeHook: State change to $state"); \
    return; }
```

3.120.25. DisconnectWhenIgnore

If the *AuthBy* returns IGNORE message because of an authentication backend database failure, *DisconnectWhenIgnore* flag parameter defines the server behaviour. When this is set, the server disconnects the client without returning TACACS+ error message before disconnecting. This may cause the client to continue with local authentication after the authentication backend failure.

Note

The client behaviour, when the TACACS+ error message is not sent, depends on its implementation.

3.120.26. ContextId

ContextId parameter specifies how to derive a lookup key for TACACS+ authentication context when authorizing TACACS+ requests. Context specific special characters supported are: TACACS+ client IP address (%0), TACACS+ client source TCP port (%1), TACACS+ user (%2), TACACS+ port (%3), TACACS+ rem_addr (%4). The default is fine for the most cases. Defaults to **`%2:%0`**.

```
# Base context lookup only on username
ContextId %2
```

3.120.27. GroupCacheFile

CAUTION

This option is deprecated and no longer does anything. AllowAuthorizeOnly replaces GroupCacheFile. For more information, see Section 3.120.13. AllowAuthorizeOnly on page 413.

3.120.28. Clients

This optional parameter specifies a list of IP addresses that connections will be accepted from. You can specify one or more comma or space separated IP addresses on each Client line. You can specify multiple Client parameters. Only exact matches are supported at present. The default is to accept connections from any and all clients.

If Clients is specified and a client attempts to connect from an IP address that is not named, Radiator will log a WARNING level message then reject and close the connection.

```
# Only accept connections from some addresses
Clients 127.0.0.1, 203.63.154.29
Clients 203.63.154.27
```

3.121. <ServerRADSEC>

This clause accepts RadSec (RFC 6614) connections from *<AuthBy RADSEC>* clauses in other Radiators other RadSec clients and processes RADIUS requests sent over the RadSec connection in a similar way to how a Client clause received conventional UDP RADIUS requests. RadSec can be used to provide secure reliable proxying of RADIUS requests from one Radiator to another, even over insecure networks. For more information, see RadSec white paper [https://files.radiatorsoftware.com/radiator/whitepapers/radsec-whitepaper.pdf].

Both *<ServerRADSEC>* and any RadSec clients that connect to it must have the same Secret configured, otherwise they are not able to exchange RADIUS requests correctly, irrespective of whether any TLS or other configuration parameters are correct. For compliance with RFC 6614, ServerRADSEC defaults to a Secret of **radsec**.

All valid RADIUS requests received by *<ServerRADSEC>* are dispatched to the first matching *<Realm>* or *<Handler>* clause, in the same was as the *<Client>* clause. The reply to any incoming request will be automatically delivered back to the original requesting RadSec client.

<ServerRADSEC> supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80. When you enable TLS, you must configure a server certificate, otherwise a RadSec client is not able to establish a TLS encrypted connection to *<ServerRADSEC>*.

ServerRADSEC has *TLS_RequireClientCert* enabled by default. When a RadSec client presents a certificate to the RadSec Server, the RadSec server performs a number of checks to validate the client certificate. The client certificate is checked for valid start and end dates, and also checks the chain of validity back to the issuing Certificate Authority, using the root certificates specified in *TLS_CAFile* or *TLS_CAPath*. If *TLS_PolicyOID* parameter is defined, the OIDs must be present in the certificate path. Also a client certificate is only accepted if at least one of the following conditions are true:

- The IP address of the client exactly matches a subjectAltName with type IPADD (IP Address) in the certificate. If *subjectAltNameDNS* is configured, its value is used to match the certificate's subjectAltName values with type DNS.
- The Subject in the certificate matches the pattern specified by the *TLS_ExpectedPeerName* parameter in this *<ServerRADSEC>* clause.

and

- If *TLS_SubjectAltNameURI* parameter is defined in the *<ServerRADSEC>* clause, the certificate must contain a *subjectAltName* of type URI that matches the *TLS_SubjectAltNameURI* regular expression.
- If *TLS_CertificateFingerprint* parameter is defined in the *<AuthBy RADSEC>* clause, the certificate's fingerprint must match at least one of the *TLS_CertificateFingerprint* options.

Note that the default *TLS_ExpectedPeerName* pattern is .+, which matches any Subject. This means than in the default configuration, *<ServerRADSEC>* accepts any client whose client certificate can be validated against a root certificate specified by *TLS_CAFile* or *TLS_CAPath*.

These certificate validation rules are consistent with RFC 2595.

3.121.1. Port

This optional parameter specifies which network port *<ServerRADSEC>* listens to for connections from RadSec clients. The default value **2083**, the official IANA port number for RadSec. *Port* can be a numeric port number or symbolic port or service name.

3.121.2. BindAddress

This optional parameter specifies one or more network interface addresses to listen for incoming connections. It is only useful if you are running Radiator on a multi-homed host (a host that has more than one network address). The default value is the global *BindAddress*, which defaults to **0.0.0.0**. It listens to all networks connected to the host. For more information, see Section 3.7. Global parameters on page 29.

Using this parameter, you can run multiple instances of Radiator on the one computer, where each instance listens to connections directed to a different host address. *BindAddress* can include special formatting characters, and multiple comma separated IPv4 and IPv6 addresses.

When SCTP multihoming is supported, all addresses defined with *BindAddress* must be either IPv4 or IPv6 addresses. Radiator binds all addresses to one listen socket instead of creating multiple listen sockets.

Here is an example of using *BindAddress*:

```
# Only listen on one IPv4 address and the IPv6 loopback
BindAddress 203.63.154.1, ::1
```

3.121.3. Secret

This parameter specifies the shared secret that will be used between this *<ServerRADSEC>* and RadSec clients that connect to it. The shared secret is used in the same way as *Secret* parameter in the Client clause: to encrypt passwords and generate message authenticators. The shared secret must be configured identically into *<ServerRADSEC>* and all RadSec clients that connect to it, regardless of whether TLS is enabled or not. An authentication error will occur if the shared secret is not correctly configured. For compliance with RFC 6614, the default value is **radsec**.

3.121.4. RewriteUsername

This is an optional parameter. It enables you to alter the username in authentication and accounting requests. For more details and examples, see Section 8. Rewriting user names on page 484.

3.121.5. StripFromRequest

This optional parameter strips the named RADIUS attributes from the RADIUS requests received by ServerRADSEC before passing them to any authentication modules. The value is a comma separated list of attribute names. StripFromRequest removes attributes from the request before AddToRequest adds any to the request. There is no default.

```
# Remove any NAS-IP-Address,NAS-Port attributes
StripFromRequest NAS-IP-Address,NAS-Port
```

3.121.6. AddToRequest

This optional parameter adds any number of RADIUS attributes to the RADIUS requests that *<ServerRADSEC>* receives. It can be used to tag requests arriving from RadSec for special handling within Radiator or in remote RADIUS servers.

Here is an example of using *AddToRequest*:

```
AddToRequest NAS-Identifier=RADSEC
```

3.121.7. AddToRequestIfNotExist

This string is an optional parameter. It adds attributes to the requests that *<ServerRADSEC>* receives. Unlike *AddToRequest*, an attribute is added only if it does not exist in the request already. The value is a comma-separated list of attribute-value pairs.

3.121.8. AllowInReject

This optional parameter specifies the only attributes that are permitted in an Access-Reject. This can be useful in Handlers with multiple AuthBys where the attributes added before a rejecting AuthBy need to be stripped from the resulting Access-Reject.

AllowInReject is not set by default and does not remove anything from Access-Rejects.

```
# Only permit a limited set of attributes in a reject.
AllowInReject Message-Authenticator, EAP-Message
```

3.121.9. DefaultRealm

This optional parameter can be used to specify a default realm to use for received RadSec requests that have a user name that does not include a realm. If the incoming user name does not have a realm (i.e. there is no @something following the user name) and if DefaultRealm is specified, the User-Name in the resulting RADIUS request will have @defaultrealm appended to it. The realm can then be used to trigger a specific <Realm> or <Handler> clause. This is useful if you operate a number of RadSec clients for different customer groups and where some or all of your customers log in without specifying a realm.

```
# Realmless logins to this NAS will be treated
# as if they are for realm open.com.au
<ServerRADSEC>
      Secret ....
      .....
      DefaultRealm open.com.au
</Client>
<Realm open.com.au>
      .....
</Realm>
```

3.121.10. Protocol

This optional parameter specifies which Stream protocol is used to carry RadSec. Options are **`tcp`** for TCP/IP or **`sctp`** for SCTP (Stream Control Transmission Protocol). The default value is **`tcp`**. Not all hosts are able to support **`sctp`**, consult your vendor. The protocol setting must be the same in each RadSec server and client.

```
Protocol sctp
```

Tip

On modern Linux hosts, SCTP support is in a loadable module, and can be enabled with:

```
modprobe sctp
```

3.121.11. PreHandlerHook

This optional parameter allows you to define a Perl function that is called during packet processing. It can be configured within several types of clauses for which its functionality is slightly different:

- Client clause

  `PreHandlerHook` is called for each request after per-Client user name rewriting and duplicate rejection, and before the request is passed to a Realm or Handler clause.

- AuthBy clause

  The functionality depends on the used EAP authentication type:

  – PEAP, EAP-TTLS, EAP-FAST

    `PreHandlerHook` specifies a Perl hook to be called before the inner request is re-dispatched to a matching Realm or Handler.

  – LEAP

    If `EAP_LEAP_MSCHAP_Convert` flag is set, `PreHandlerHook` specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-MSCHAPv2

    If `EAP_PEAP_MSCHAP_Convert` flag is set, `PreHandlerHook` specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

  – EAP-GTC

    If `EAP_GTC_PAP_Convert` flag is set, `PreHandlerHook` specifies a Perl hook to be called before the converted request is re-dispatched to a matching Realm or Handler.

- AuthBy DYNAUTH clause

  `PreHandlerHook` is called for each request created by the clause before the request is passed to a Realm or Handler clause.

- ServerRADSEC clause

  `PreHandlerHook` is called for each request after global and per-ServerRADSEC user name rewriting and before the request is passed to a Realm or Handler clause.

- ServerDIAMETER clause

  `PreHandlerHook` is called for each request received by ServerDIAMETER before the request is passed to a Realm or Handler clause.

- ServerTACACSPLUS clause

  `PreHandlerHook` is called for each request before it is passed to a Realm or Handler clause. If a Client is found for the request, Client's `PrehandlerHook` is run before ServerTACASPLUS's `PreHandlerHook`. Global and per-Client user name rewriting and other processing is done before the hooks are run.

A reference to the request is passed as the only argument.

The hook code is compiled by Perl when Radiator starts up. Compilation errors in your hook code are reported to the log file at start-up time. Runtime errors in your hook are also reported to the log file when your hook executes. Multiline hooks with trailing backslashes (\) are parsed by Radiator into one long line. Therefore, do not use trailing comments in your hook.

PreHandlerHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. Here is an example of using *PreHandlerHook*:

```
# Fake a new attribute into the request
PreHandlerHook sub { ${$_[0]}->add_attr('test-attr', \
      'test-value');}
```

3.121.12. Identifier

This optional parameter can be used to match Client-Identifier check items in Handler and other clauses, so that you can trigger special behaviour for requests received by a particular ServerRADSEC clause.

3.121.13. PacketTrace

This optional flag forces all packets that pass through this module to be logged at trace level 5 until they have been completely processed. This is useful for logging packets that pass through this clause in more detail than other clauses during testing or debugging. The packet tracing stays in effect until it passes through another clause with *PacketTrace* set off or **0**.

PacketTrace is available for the following clauses:

- *Client*
- *Handler*
- *Realm*
- *AuthBy*
- *ServerDIAMETER*
- *ServerRADSEC*
- *ServerTACACSPLUS*

Here is an example of using *PacketTrace*:

```
# Debug any packets that pass through here
PacketTrace
```

3.121.14. StatusServer

Normally, when a Status-Server request is received, Radiator replies with some statistics including the total number of requests handled, the current request rate and so on. You can control Status-Server response by setting *StatusServer* to one of the following values:

- **off**

  Status-Server requests are ignored.

- **minimal**

  Reply without any attributes.

- **uptime**

  Reply without server uptime and, in case of RADIUS/TCP and RADIUS/TLS (RadSec), connection uptime .

- **default**

Reply with statistics.

3.121.15. MaxBufferSize

This optional advanced parameter specifies the maximum number of octets that are output and input buffered by Radiator's Stream modules. This advanced parameter usually does not need adjusting.

3.121.16. DisconnectTraceLevel

This optional parameter specifies log trace level for peer initiated disconnects. The default value is error level **0**. When connections are known to be short-lived, a non-default value may be useful. This parameter is available for all Stream based modules, such as *<ServerDIAMETER>* and *<AuthBy RADSEC>*.

```
# Debug logging is enough for peer disconnects
DisconnectTraceLevel 4
```

3.121.17. StreamStateChangeHook

This optional parameter allows you to define a Perl function that will be called when a Stream connection state to a peer changes. RadSec, Diameter, TACACS+, HTTP and SIGTRAN clients and servers use Stream connections in Radiator. This is a low layer hook that runs before, for example, RadSec TLS handshake or Diameter Capabilities Exchange messages are exchanged. The following arguments are passed in the following order:

- Reference to this `Radius::Stream` derived type. Examples of derived types are `Radius::RadsecHost` and `Radius::DiameterConnection`.
- New stream state. One of integer constant values:
  - `$Radius::Stream::STREAM_STATE::CONNECTED`
  - `$Radius::Stream::STREAM_STATE::DISCONNECTED`

StreamStateChangeHook can be an arbitrarily complicated Perl function, that might run external processes, consult databases, change the contents of the current request or many other things. IP address and other specific information is not passed to the hook. Their format and type depends on the derived type. For example, SCTP streams may have multiple source and destination addresses. The following example shows how to log information from the hook no matter what the type of the stream object is.

```
StreamStateChangeHook sub { \
    my ($self, $new_state) = @_; \
    my $state = ($new_state == $Radius::Stream::STREAM_STATE::CONNECTED) ? \
                  'connected' : 'disconnected'; \
    main::log($main::LOG_INFO, "StreamStateChangeHook: State change to $state"); \
    return; }
```

3.121.18. StreamMaxClients

This optional parameter specifies the maximum number of accepted connections for each listen socket. This parameter is not set by default. When the parameter is set to zero or unset, no restrictions apply. This parameter is available for all StreamServer based modules, such as *<ServerDIAMETER>* and *<ServerRADSEC>*.

With server farm, see FarmSize on page 42, this parameter allows distributing all incoming connections equally between worker processes. For example, if the number of connections is known not to exceed 10, a farm of 12 workers will have 2 spare workers with each of the 10 remaining workers handling one connection each.

Note

Currently this parameter is used as follows: All values larger than 0 are treated as 1.

```
# Allow one connection for each listen socket
StreamMaxClients 1
```

3.121.19. Clients

This optional parameter specifies a list of IP addresses that connections will be accepted from. You can specify one or more comma or space separated IP addresses on each Client line. You can specify multiple Client parameters. Only exact matches are supported at present. The default is to accept connections from any and all clients.

If Clients is specified and a client attempts to connect from an IP address that is not named, Radiator will log a WARNING level message then reject and close the connection.

```
# Only accept connections from some addresses
Clients 127.0.0.1, 203.63.154.29
Clients 203.63.154.27
```

3.122. <ServerHTTP>

The *<ServerHTTP>* clause presents a HTTP interface that allows Radiator to be monitored, configured and reconfigured through a standard web browser. The Graphical User Interface (GUI) that it presents is designed to be easy to use and intuitive, and to allow access to the full range of detailed configuration options that are usually access directly by editing the configuration file.

The GUI presented by *<ServerHTTP>* is a useful alternative to the more traditional editing of the Radiator configuration file. Further it allows access to other useful information about the host that Radiator is running on, the details of the version of Perl installed, and details about the versions and modules of Radiator installed on that host.

The GUI presented by this interface is described in Section 10. Configuring Radiator with GUI on page 493.

<ServerHTTP> supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

Authentication

Any user attempting to connect to *<ServerHTTP>* is subject to authentication. If authentication does not succeed, then the user is unable to access any web pages. Once logged in, the information the user is permitted to see, and the actions the user is permitted to do are controlled by the user's Privilege Level. The authentication steps are:

1. Check all the clauses in the AuthBy list, if any, continuing until the AuthByPolicy is met.
2. If no AuthBy clause succeeds (or if there are no AuthBy clauses), authenticate against the hardwired User name and Password in this clause.
3. If the hardwired User name is not defined permit authentication as the user ’anonymous’ without a password.

If the last AuthBy returns ACCEPT, the connection is accepted. If the last AuthBy returns IGNORE, or there are no AuthBys, then fall back to the hardwired User name and Password parameters is done.

Note

If you plan to use `<AuthBy RADIUS>`, you need to configure the AuthBy with the Synchronous parameter. Otherwise `<AuthBy RADIUS>` returns immediately with IGNORE.

An authentication lasts for the time period given by `SessionTimeout`, after which the user will be required to log in again.

The users Privilege Level is determined in the following way:

1. If the successful authentication was from an AuthBy clause, and the user had a Management- Policy-Id reply item, then the Privilege Level is given by the integer in the Management-Policy-Id.
2. Otherwise the Privilege Level is given by the DefaultPrivilegeLevel parameter.

Privilege Level

The information the user is permitted to see, and the actions the user is permitted to do are controlled by the user's Privilege Level. The Privilege Level is a number from 0 to 15, where 0 is the lowest privilege, (and which does not even permit logging in), and 15 is the highest, which allows all actions.

The Privilege Level is a bitmask obtained by adding together the following numbers:

- 1: Permission to view basic (non-security-critical) status only.
- 2: Permission to reset the server
- 4: Permission to edit and change the running configuration (but not save it)
- 8: Permission to save changes to the configuration file

For example, to grant privilege to view status and to reset the server, the Privilege Level should be set to 3 (1 + 2). To grant all privileges, the Privilege Level should be set to 15 (1 + 2 + 4 + 8).

CAUTION

Careless configuration of this clause can open security holes in your RADIUS host. In order to limit the possibility of security compromise, It is recommended that you:

1. Limit the clients that can connect with the Clients parameter.
2. Make sure the Radiator configuration file is only readable by root.
3. Consider making Radiator run as a non-privileged user.
4. Use secure user names and password to authenticate access to this server.
5. Enable SSL connections only with the UseSSL flag.
6. Disable this clause when not required.

3.122.1. Port

Port number to listen for connections. Service names or integer port numbers are permitted.

3.122.2. Username

Fallback username to log in as. If there is no Username and no AuthBy clauses, users will not be required to authenticate in order to use the web interface.

3.122.3. Password

Fallback password for Username. Password can be plaintext or any of the encrypted formats such as {crypt}....., {nthash}....., {SHA}...., {SSHA}....., {mysql}...., {msssql}...., {dechpwd}...., {MD5}....., {clear}.... etc.

3.122.4. AuthBy

List of AuthBy clauses to use to handle authentication for new web connections. Requests are processed by each AuthBy in order until AuthByPolicy is satisfied. If there are no AuthBy clauses, the fallback Username and Password will be used.

3.122.5. AuthByPolicy

Specifies whether and how to continue authenticating after each AuthBy.

3.122.6. AuthLog

List of AuthLog clauses which will be used to log authentication of users logging in the Server HTTP interface.

3.122.7. AuditTrail

Optional filename where all editing and configuration changes will be logged. Special characters are supported.

3.122.8. LogMicroseconds

When logging, include microseconds in the time.

3.122.9. SessionTimeout

Maximum time in seconds a session is permitted to continue without logging in again.

3.122.10. LogMaxLines

Maximum number of recent log lines which will be displayed on the Log page.

3.122.11. Trace

Logging trace level. Only messages with the specified or higher priority will be logged.

3.122.12. DefaultPrivilegeLevel

Privilege level to be used if a per-user Management-Policy-Id is not available from a successful authentication from the AuthBy list. The privilege level is a bitmask described above.

3.122.13. PageNotFoundHook

If a page is requested but not found in the set of built-in pages PageNotFoundHook is called to try to handle the request. PageNotFoundHook is passed the requested URI and a reference to the ServerHTTP connection. If it can handle the request, it returns an array of ($httpcode, $content, @httpheaders), else undef.

```
PageNotFoundHook sub {return (200, "your HTML content");}
```

3.122.14. Clients

This optional parameter specifies a list of IP addresses that connections will be accepted from. You can specify one or more comma or space separated IP addresses on each Client line. You can specify multiple Client

parameters. Only exact matches are supported at present. The default is to accept connections from any and all clients.

If Clients is specified and a client attempts to connect from an IP address that is not named, Radiator will log a WARNING level message then reject and close the connection.

```
# Only accept connections from some addresses
Clients 127.0.0.1, 203.63.154.29
Clients 203.63.154.27
```

3.123. <StatsLog xxxxxx>

This marks the beginning of a StatsLog clause, which defines how to log statistics from Radiator internal objects. The xxxxxx is the name of a specific StatsLog module. For more about Diameter statistics and how to configure logging them, see Section 3.127. <DiaStatsLog xxxxxx> on page 432. This section lists the configuration parameters all StatsLogs and DiaStatsLogs use and describes StatsLog statistics in detail.

StatsLogs may also use additional parameters that are specific for that StatsLog. StatsLog clauses are defined at the top level configuration.

Radiator collects statistics for the server as a whole, as well as for each Client, Realm, Handler, AuthBy, and Host that a packet passes through. The following statistics are collected for each object or clause:

Table 10. Statistics collected within Radiator and usage in Format parameter

Symbolic name	Description	Format character
requests	Total requests	%22
droppedRequests	Total dropped requests	%14
duplicateRequests	Total duplicate requests	%17
proxiedRequests	Total proxied requests	%21
proxiedNoReply	Total proxied requests with no reply	%20
badAuthRequests	Total Bad authenticators in requests	%11
responseTime	Average response time (seconds). This is a cumulative moving average of the per-request elapsed processing time for the last 100 requests.	%23
accessRequests	Access requests	%6
dupAccessRequests	Duplicate access requests	%15
accessAccepts	Access accepts	%3
accessRejects	Access rejects	%5
accessChallenges	Access challenges	%4
malformedAccessRequests	Malformed access requests	%18

Symbolic name	Description	Format character
badAuthAccessRequests	Bad authenticators in authentication requests	%9
droppedAccessRequests	Dropped access requests	%12
accountingRequests	Accounting requests	%7
dupAccountingRequests	Duplicate accounting requests	%16
accountingResponses	Accounting responses	%8
malformedAccountingRequests	Malformed accounting requests	%19
badAuthAccountingRequests	Bad authenticators in accounting requests	%10
droppedAccountingRequests	Dropped accounting requests	%13

CAUTION

In the case of AuthBy clauses you should set a unique Identifier in each clause, otherwise you will not necessarily be able to distinguish between each AuthBy clause in StatsLog statistics.

Tip

responseTime measures the per-request processing time, the time required for Radiator to fully process a single request. It does not measure the Access-Request/ Access-Accept delay. For more information about how responseTime is calculated, see Table 10. Statistics collected within Radiator and usage in Format parameter on page 427.

Tip

responseTime will be computed based on time measurements accurate to 1 microsecond when Time::HiRes module is available. This should be the case with all modern Perl versions.

3.123.1. Interval

This is the time interval (in seconds) between each set of statistics. The default value is 600 seconds (10 minutes).

```
# Log once per day
Interval 86400
```

3.123.2. StatsType

By default statistics produce counter, for example cumulative, values. You can change this with *StatsType* parameter. Format specifiers, such as *%{GlobalVar:name}*, are evaluated when the configuration is loaded. The parameter value is the output type and can be one of:

- **`cumulative`**: cumulative counter shows the number of processed packets.
- **`derivative`**: derivative is the difference (delta) between two counter values in time interval.
- **`packet_rate`**: packet rate is the amount of packets transferred within time interval (packets per second).
- **`all`**: all produces output from all available statistic types (cumulative, derivative and packet_rate).

The default output type is **`cumulative`**.

```
# We are interested in the difference (delta)
StatsType derivative
```

3.123.3. RateCalculationInterval

This is used to calculate packet rates that are different from the value of Interval. RateCalculationInterval is an optional parameter that defines the time interval (in seconds) in which the packet rate is calculated. For example, if Interval is set to 600 seconds and RateCalculationInterval is set to 60, statistics is dumped in every 600 seconds but packet rate shows the average amount of packets in 60 second interval.

Calculated example: Amount of packets in Interval (600 s) is 1000 pkts and RateCalculationInterval is one minute (60 s). (1000 pkts/10 min) * (60 s/min) / (600 s/10 min) = 100 pkts/min.

RateCalculationInterval defaults to 1 (packets per second).

```
# We are interested in packets per minute
RateCalculationInterval 60
```

3.123.4. OutputFormat

This string defines the format of statistics entries. Format specifiers, such as *`%{GlobalVar:name}`*, are evaluated when the configuration is loaded. The value can be either **`text`** or **`json`**. The default value is **`text`** unless otherwise documented by a clause.

```
<StatsLog FILE>
    # Log as json instead of text
    OutputFormat json

    # Other configuration options
</StatsLog>
```

3.123.5. StatsExcludeObject

By default statistics include all Client, Handler and other objects that support statistics collection. *`StatsExcludeObject`* supports a comma separated list of the following object types:

- *`Client`*: Skip all Clients, including those loaded with ClientList clauses.
- *`Handler`*: Skip all Handler clauses.
- *`Realm`*: Skip all Realm clauses.

Currently all AuthBys within skipped Handler and Realm clauses are also skipped. How AuthBys are excluded may change in the future releases. To define exceptions, see Section 3.123.6. StatsInclude on page 430.

```
# We don't want to see any client statistics
StatsExcludeObject Client
```

3.123.6. StatsInclude

StatsInclude allows defining exceptions for statistics objects that would be otherwise skipped by *StatsExcludeObject* configuration parameter. StatsInclude value specifies a type and value separated by a comma. The following types are supported:

- *Client, name*: Include *Client* clause that matches <*Client name*> configuration clause
- *Client-Identifier, identifier*: Include *Client* clause with *Identifier identifier* configured
- *Handler, identifier*: Include *Handler* clause with *Identifier identifier* configured.
- *Realm, identifier*: Include *Realm* clause with *Identifier identifier* configured.

For more about excluding objects from statistics, see Section 3.123.5. StatsExcludeObject on page 429. *StatsInclude* is ignored when the type is not defined with *StatsExcludeObject*.

```
# We don't want to see client statistics, except for these two
StatsExcludeObject Client
StatsInclude Client, 10.20.30.40
StatsInclude Client-Identifier, bgn-gateway
```

3.123.7. FarmWorkerSpacing

If this optional parameter is set to non-zero, it specifies the time in seconds for spacing out logging done by server farm workers. Defaults to not set which causes all farm workers to log statistics at the same moment. This parameter has only effect when both *Interval* and global *FarmSize* on page 42 parameters are configured.

```
# Log once per day, use 30 second offset between each worker
Interval 86400
FarmWorkerSpacing 30
```

3.124. <StatsLog FILE>

This clause logs statistics from all Radiator internal objects to a flat file. There is an example configuration file in `goodies/statslog.cfg`.

In StatsLogFILE, the specified filename is opened every Interval seconds, and the statistics for each object are logged, one line per object. An optional header line is also logged. For each object logged, the log contains a timestamp (in seconds since 1970), the type of the object and a unique identifier, if available, followed by the value of each statistic in alphabetical order by their symbolic name (see Table 10. Statistics collected within Radiator and usage in Format parameter on page 427). It will use the Identifier field of available, otherwise the object Name.

Statistics for the server as a whole (ServerConfig) will be logged first, followed by each Client, each Realm, each Handler. Within each Realm and Handler, each AuthBy used by that clause will be logged (recursively for AuthBy GROUP). Within any AuthBy RADIUS, statistics for Host clauses will be logged. You can find an example of the log format in `goodies/statslog.cfg`.

3.124.1. Filename

This is the name of the file to log statistics to. Defaults to `%L/statistics`. The file name can include special formatting characters, although data from the current request or reply are never available (logging is never done in the context of a current request. For more information, see Section 3.3. Special formatters on page 21.

```
Filename /path/to/my/stats.dat
```

CAUTION

Logging to STDOUT by specifying Filename as a single dash - is no longer supported. Support for this was removed in Radiator 4.18.

3.124.2. Format

This optional parameter specifies the format for each logging line. You can use this to control exactly what statistics are logged and in what order they appear.

If Format is not defined, statistics data will be logged in the following order (also shown is the special character that is available for that data in the Format specification). All fields will be colon separated.

- Current timestamp in seconds since 1970 (%0)
- Type of object whose data is being logged (%1)
- Identifier of the object (if present) else its Name (%2)
- Each statistic in alphabetical order of its symbolic name (%3 to %23). See Table 10. Statistics collected within Radiator and usage in Format parameter on page 427.

```
# Log the timestamp, objtype, name and average response time
Format %0:%1:%2:%23
```

3.124.3. Header

This optional parameter allows you to customise the Header line that is logged before each set of statistics. It can be useful for describing the contents of each column when importing into Excel etc.

The default is a single line beginning with a hash, followed a name for each column, colon separated.

If Header is defined as an empty string, no Header lines will be written.

```
# No headers
Header
```

3.125. <StatsLog SQL>

This clause logs statistics from all Radiator internal objects to an SQL database. There is an example configuration file in `goodies/statslog.cfg`.

For more information about what data is available for logging, see Table 10. Statistics collected within Radiator and usage in Format parameter on page 427.

By default, *<StatsLog SQL>* executes an SQL insert statement for each object to be logged, which includes a timestamp, object type, object name, and each statistic available for logging. *<StatsLog SQL>* supports all the common SQL configuration parameters. For more information about the SQL configuration parameters, see Section 3.8. SQL configuration on page 46. This clause ignores *OutputFormat* configuration parameter.

3.125.1. InsertQuery

This optional parameter specifies the SQL query to be used for each log. It can include special formatting characters. For more information, see Section 3.3. Special formatters on page 21. %0 to %23 are replaced by statistics data. For more information, see Section 3.124.2. Format on page 431.

The default is an SQL query something like this, which is compatible with the example RADSTATSLOG table created by the example database schemas in goodies.

```
insert into RADSTATSLOG (TIME_STAMP, TYPE, IDENTIFER, <symbolic
stats names>) values (9999999, 'objectname', 'objectidentifier',
<statistics values>)
```

Here is an example that only logs a subset of the statistics:

```
InsertQuery insert into MYTABLE (TIME_STAMP, TYPE, ID, \
      RESPONSETIME) values (%0, %1, %2, %23)
```

3.126. <StatsLog REDIS>

This clause logs statistics to Redis from all Radiator internal objects. Statistics can later fetched from Redis for management applications, log transport agents, such as logstash Redis input plugin, or other use. The statistics are currently logged in JSON format. StatsLog REDIS uses GossipRedis for communicating with a Redis server. For more information, see Section 3.134. <GossipRedis> on page 439.

There is an example configuration file in `goodies/statslog.cfg`.

For more information about what data is available for logging, see Table 10. Statistics collected within Radiator and usage in Format parameter on page 427.

3.126.1. StatsKey

This is a Redis key to store stats. The default is **`Radiator::StatsLogREDIS`**. Special formatting characters are supported.

```
# Our key has identifier and farm instance
StatsKey Radiator:%{Server:Identifier}:%O:stats
```

3.127. <DiaStatsLog xxxxxx>

This marks the beginning of a DiaStatsLog clause, which defines how to log Diameter statistics. The xxxxxx is the name of a specific DiaStatsLog module. This section lists the parameters all DiaStatsLogs use.

DiaStatsLogs may also use other parameters that are specific for that DiaStatsLog. DiaStatsLog clauses are defined at the top level configuration and they understand also the same parameters as *`<StatsLog xxxxxx>`*. For more information, see Section 3.123. <StatsLog xxxxxx> on page 427.

There is an example configuration file for the different *`<DiaStatsLog>`* clauses in `goodies/diastatslog.cfg`.

3.127.1. PeerAliveDetectionInterval

This integer defines the time interval during which the counter update must happen, otherwise the peer is considered to be inactive. The time interval is given in seconds. The default value is **`60`** seconds. *`PeerAliveDetectionInterval`* must be twice as long as the watchdog trigger time so that at least DWR/DWA are updated. Twinit default for watchdog trigger time is 30 seconds according to RFC 6733, thus *`PeerAliveDetectionInterval`* must be minimum 60 seconds. If this parameter is set to **`0`**, it disables the peer alive checks.

3.127.2. PeerRemovalThreshold

This integer defines the number of StatsLog intervals how long an inactive peer is kept in memory. The inactive peer is removed from the memory when *`PeerRemovalThreshold`* * *`PeerAliveDetectionInterval`* seconds is elapsed. The default value is 1. If this parameter is set to **`0`**, inactive peer cleanup is disabled.

3.128. <DiaStatsLog FILE>

This clause logs statistics from all Radiator internal Diameter objects to a flat file.

3.128.1. Filename

This string defines the name of the file where the log statistics are saved. The default value is **`%L/statistics`**. You can use special characters but data from the current request or reply is not available as logging is never done in the context of a current request. For more information about special characters, see Section 3.3. Special formatters on page 21.

3.129. <DiaStatsLog REDIS>

This clause logs statistics from all Radiator internal Diameter objects to Redis. Statistics can later be fetched from Redis for management applications, log transport agents, such as logstash Redis input plugin, or any later use. The statistics are currently logged in JSON format. <DiaStatsLog REDIS> uses GossipRedis for communicating with a Redis server. For more information, see Section 3.134. <GossipRedis> on page 439

<DiaStatsLog REDIS> is currently experimental and will be documented later.

3.129.1. StatsKey

This defines the Redis key to store Diameter stats. The default is **`Radiator::DiaStatsLogREDIS`**. You can use the special characters. For more information, see Section 3.3. Special formatters on page 21.

```
# Our key has identifier and farm instance
StatsKey Radiator:%{Server:Identifier}:%O:diastats
```

3.129.2. OutputFormat

This string defines the format for the statistics entries. Currently the only supported value is **`JSON`**, it is also set as a default.

3.130. <DiaStatsLog SQL>

This clause logs statistics from all Radiator internal Diameter objects to an SQL database. This clause ignores *`OutputFormat`* configuration parameter.

3.130.1. InsertQuery

This string defines the SQL query that is used for each log. This is an optional parameter. If this is not defined, only the standard set of statistics is logged. You can use special characters. For more information, see Section 3.3. Special formatters on page 21.

3.130.2. TableName

This string defines the name of the SQL table where the log data is inserted. The default value is **`RADDIASTATSLOG`**.

3.131. <MessageLog xxxxxx>

This marks the beginning of a *MessageLog* clause, which defines how to log sent and received RADIUS, Diameter, and TACACS+ messages. The xxxxxx is the name of a specific *MessageLog* module. This section lists the parameters all *MessageLog*s use. *MessageLog*s can also use other parameters that are specific for that *MessageLog*. *MessageLog* clauses must be defined at the top level configuration.

You can have more than one *MessageLog* clause. For example RADIUS, Diameter, and TACACS+ typically have separate loggers. Message logging is not enabled by default. For more information about how to enable message logging, see top level configuration parameter in Section 3.7.51. GlobalMessageLog on page 42.

See `goodies/logformat.cfg` for a configuration sample.

3.131.1. Identifier

This specifies an optional symbolic name that can be used to refer to the logger from somewhere else.

```
# Log all incoming RADIUS message with MessageLog FILE
GlobalMessageLog radius,messagelograd

<MessageLog FILE>
      Identifier messagelograd
      Format text
      Filename %L/messagelog-%0-%1
</MessageLog>
```

3.131.2. LogSelectHook

This parameter defines an optional Perl hook that is run for each log message. If it returns true, the message is logged, otherwise logging is skipped. This is not defined by default and all messages are logged.

LogSelectHook has the following arguments:

- *direction*

  This argument defines the direction and is either *in* or *out*.

- *protocol*

  This argument defines the used protocol. The supported values are:

  - *radius*
  - *radsec*
  - *diameter*
  - *tacacsplus*

- *from_ip*
- *from_port*
- *to_ip*
- *to_port*
- *request_object*

The following example logs only the messages from/to 10.10.10.10:

```
LogSelectHook sub { return 1 if ($_[2] eq '10.10.10.10' || $_[4] eq '10.10.10.10') }
```

3.132. <MessageLog FILE>

This clause logs RADIUS, Diameter, or TACACS+ messages to a file. You can define as many <MessageLog FILE> clauses as you wish at the top level configuration. Each clause can specify different logging conditions and a different log file.

```
# This message logger logs to a file
<MessageLog FILE>
      Identifier myauthlogger
      Format text
      #Encoding hex
      Filename %L/messagelog-%0-%1
</MessageLog>
```

<MessageLog FILE> understands also the same parameters as all MessageLogs. For more information, see Section 3.131. <MessageLog xxxxxx> on page 434.

3.132.1. Filename

This optional parameter specifies the name of the file where messages log entries are written. You can use any of the special characters defined. For more information about special characters, see Section 3.3. Special formatters on page 21. The default value is `%L/messagelog`.

The special characters are replaced as follows:

- *%0* is replaced by the protocol.

  For the supported protocols, see Section 3.7.51. GlobalMessageLog on page 42 .

- *%1* is replaced by the log format value.

  For the supported log format values, see Section 3.132.2. Format on page 435.

- *%2* is replaced by the encoding.

  For the supported encodings, see Section 3.132.3. Encoding on page 435 .

If the Filename parameter starts with a vertical bar character (‘|’), the rest of the filename is assumed to be a program to which the output is to be piped. Otherwise the output appends to the named file:

```
# Example file name: messagelog-radius-text-hex
Filename %L/messagelog-%0-%1-%2
```

3.132.2. Format

This optional parameter specifies the format for the message log entries. Possible values are: **`text`** and **`text2pcap`**. Unknown values default to **`text`**, which is also the default value. *`Format`* value supports format specifiers, such as *`%{GlobalVar:name}`*, which are evaluated when the configuration is loaded.

```
# Prepare for possible conversion to pcap format
Format text2pcap
```

3.132.3. Encoding

This optional parameter specifies the encoding for the message log entries. Encoding is done after the message has been formatted. Possible values are: **none** and **hex**. Unknown values default to **none**, which is also the default value. *`Encoding`* value supports format specifiers, such as *`%{GlobalVar:name}`*, which are evaluated when the configuration is loaded.

```
# Might be useful if using a binary Format
Encoding hex
```

3.133. <Monitor>

This clause enables external client programs to make an authenticated TCP connection to Radiator, and use that connection to monitor, probe, modify, and collect statistics from Radiator. One such external client program is Radar, a real-time interactive GUI that permits monitoring, plotting of statistics and much more. For more information, see Radar website [https://radiatorsoftware.com/products/radar/].

Monitor permits the telnet connections and implements a simple command syntax that allows various actions to be executed. For more information about the command language that Monitor implements, see Section 18. Monitor command language on page 523. Monitor permits multiple simultaneous independent connections. Radiator also permits multiple Monitor clauses, each listening on a different *`Port`* or *`BindAddress`*.

Monitor authenticates incoming connections. Only if the connection submits a valid user name and password Monitor honours the requests on that connection. You can configure Monitor with either a hardwired user name and password, or with a standard Radiator AuthBy clause. You can specify one or more AuthBy parameters or AuthBy clauses and an AuthByPolicy similar to *`<AuthBy GROUP>`*. For more information, see Section 3.38.1. AuthByPolicy on page 185. As a security measure, if a Monitor connection fails authentication 5 times, the connection is automatically disconnected.

CAUTION

Careless configuration of this clause can open security holes in your RADIUS host. To avoid this, we recommend you to take the following actions:

- Limit the clients that can connect with the Clients parameter.
- Make sure the configuration file is only readable by root.
- Consider making radiusd run as a non-privileged user.
- Use secure user names and passwords to authenticate access to this server.
- Disable this clause when not required.

`<Monitor>` supports TLS. For more information about TLS parameters, see Section 3.11. TLS configuration on page 80.

3.133.1. Port

This optional parameter specifies the TCP port number to listen on. The argument may be either a numeric port number or an alphanumeric service name as specified in `/etc/services` (or its equivalent on your system). Defaults to 9048.

Tip

You can pass the port number as a command line argument to radiusd with a configuration like this:

```
Port %{GlobalVar:monitorport}
```

and then run radiusd with an argument to set the port number like this:

```
radiusd monitorport=9000 ....
```

3.133.2. Clients

This optional parameter specifies a list of IP addresses that connections will be accepted from. You can specify one or more comma or space separated IP addresses on each Client line. You can specify multiple Client parameters. Only exact matches are supported at present. The default is to accept connections from any and all clients.

If Clients is specified and a client attempts to connect from an IP address that is not named, Radiator will log a WARNING level message then reject and close the connection.

```
# Only accept connections from some addresses
Clients 127.0.0.1, 203.63.154.29
Clients 203.63.154.27
```

3.133.3. BindAddress

This optional parameter specifies a single host address to listen for Monitor connections on. It is only useful if you are running Radiator on a multi-homed host (i.e. a host that has more than one network address). Defaults to 0.0.0.0 (i.e. listens on all networks connected to the host). For more information, see Section 3.5. Address binding on page 27.

Using this parameter, you can run multiple instances of Radiator on the one computer, where each Radiator listens to Monitor requests directed to a different host address. BindAddress can include special formatting characters.

```
# Only listen on one network address
BindAddress 203.63.154.1
```

3.133.4. AuthBy, <AuthBy xxxxxx> and AuthByPolicy

Monitor supports either a hardwired user name and password, or one or more AuthBy parameters or <AuthBy xxxxxx> clauses in a similar way to <AuthBy GROUP>. For more information, see Section 3.38. <AuthBy GROUP> on page 184. If there are no AuthBy clauses, or if they all IGNORE the authentication, the hardwired Username and Password will be tried.

Tip

If you are configuring Monitor in order to accept connections from Radar, or any other application that uses the Monitor {chap} passwords, the AuthBy needs to be one that contains plaintext passwords in its database.

3.133.5. Username

This optional parameter specifies the user name that must authenticate any connection through this Monitor clause. Username and Password will be checked if there are no AuthBy clauses, or if they all IGNORE the authentication.

```
# Hardwired username and password
Username mikem
Password fred
```

3.133.6. Password

This optional parameter specifies the password that must authenticate any connection through this Monitor clause. Username and Password will be checked if there are no AuthBy clauses, or if they all IGNORE the authentication.

3.133.7. TraceOnly

This optional parameter prevents connections through this Monitor from getting statistics, getting or setting configuration data, or restarting the server. It inhibits the following Monitor commands:

- STATS
- DESCRIBE
- SET
- LIST
- RESTART
- GET

This flag is useful for limiting access to privileged data by certain staff.

Tip

You can have multiple Monitor clauses on different Ports, so it is possible to have one with TraceOnly and one without. This would allow you to permit some Radar users to get access only to Trace data, and some to have access to all functions:

```
# This one is restricted
<Monitor>
      Port 9001
      TraceOnly
      Username xxx
      Password xxx
</Monitor>
# This one allows a superuser to get access to all data
<Monitor>
      Port 9002
      Username yyy
      Password yyy
</Monitor>
```

3.133.8. StatisticsOnly

This optional parameter prevents connections through this Monitor from tracing, getting or setting configuration data, or restarting the server. It inhibits the following Monitor commands:

- TRACE
- TRACE_USERNAME
- DESCRIBE
- SET
- LIST

- RESTART
- GET

This flag is useful for limiting access to privileged data by certain staff.

3.133.9. LogMicroseconds

This optional parameter makes Monitor log the current microseconds at the end of the time string.

3.133.10. LogTraceId

This is a flag parameter. When set, *<Monitor>* logging includes a tracing identifier. When *LogTraceId* is set within a *<Monitor>* clause, the tracing identifier is not added to other Radiator logging messages, it affects only the *<Monitor>* logging. Otherwise the functionality of *LogTraceId* within *<Monitor>* clause is similar to global Section 3.7.4. LogTraceId on page 30.

3.133.11. LogFarmInstance

This is a flag parameter. When set, *<Monitor>* logging includes server farm instance number. When *LogFarmInstance* is set within a *<Monitor>* clause, the farm instance number is not added to other Radiator logging messages, it affects only the *<Monitor>* logging. Otherwise the functionality of *LogFarmInstance* within *<Monitor>* clause is similar to global Section 3.7.53. LogFarmInstance on page 43.

3.134. <GossipRedis>

<GossipRedis> clause is used to configure a Redis based Gossip instance. For more information about Gossip framework, see Section 11. Using Gossip framework on page 507.

Radiator's Gossip framework requires Data::MessagePack Perl module. *<GossipRedis>* requires Redis Perl module.

For a configuration sample, see `goodies/farmsize.cfg`.

3.134.1. Host

This parameter specifies the Redis server to connect to. The default value is **`127.0.0.1`**.

Here is an example of using *Host*:

```
# IPv6 is not supported yet
Host 172.16.20.3
```

3.134.2. Port

This optional parameter specifies the symbolic service name or port number of the Redis server. The default value is **`6379`**.

Here is an example of using *Port*:

```
Port 6380
```

3.134.3. Sock

This specifies Unix domain socket to connect to Redis server. There is no default.

Here is an example of using *Sock*:

```
Sock /var/run/redis/redis.sock
```

3.134.4. Sentinels

This specifies Redis high availability Sentinels to connect to. There is no default and Sentinel service is not used by default.

Here is an example of using *`Sentinels`*:

```
Sentinels 10.20.30.3, 10.20.40.4
```

3.134.5. SentinelService

This specifies the name of the service to connect to via sentinels when *`Sentinels`* is set. The default value is **`mymaster`**.

Here is an example of using *`SentinelService`*:

```
SentinelService radiusmaster
```

3.134.6. SentinelPort

This specifies the symbolic service name or port number of the *`Sentinels`*. The default value is **`26379`**.

Here is an example of using *`SentinelPort`*:

```
SentinelPort 26380
```

3.134.7. Password

This specifies a password to login to the Redis server. There is no default.

Here is an example of using *`Password`*:

```
Password secret
```

3.134.8. Prefix

This specifies the prefix for the Redis connection name, pubsub channels, and get/set keys. The default value is **`radiator`**.

Here is an example of using *`Prefix`*:

```
Prefix radius
```

3.134.9. InstanceId

This specifies the instance-specific part of the Redis connection name and pubsub channels. Special formatting characters are supported. The default value is **`%h.%O`** (hostname and farm instance number).

Here is an example of using *`InstanceId`*:

```
InstanceId devel-%h.%O
```

3.134.10. DbIndex

This specifies the Redis DB index. The default value is **`0`**.

Here is an example of using *`DbIndex`*:

```
DbIndex 3
```

3.134.11. Timeout

This sets the connection and command timeout period in seconds for the connection to the Redis server, sentinel connections, sentinel reads, sentinel writes, server read, and server write. The default value is **1** second.

Here is an example of using *Timeout*:

```
Timeout 2
```

3.134.12. FailureBackoffTime

This specifies the reconnect interval when trying to login to the Redis server. The default value is **10** seconds.

Here is an example of using *FailureBackoffTime*:

```
FailureBackoffTime 30
```

3.135. <GossipUDP>

<*GossipUDP*> clause is used to configure a UDP-based Gossip instance. For more information about Radiator Gossip framework, see Gossip framework on page 507.

<*GossipUDP*> provides support for direct UDP communication between Gossip peers. <*GossipUDP*> is currently experimental.

4. Running radiusd

radiusd is the Radiator RADIUS server. At start-up, *radiusd* tries to open its configuration file. If the file can not be opened, *radiusd* exits. After the configuration has been read, *radiusd* finishes its start-up. It can change its user and group, create a PID file and do other work depending on the configuration. The location of configuration file and PID file, if PID file is written at all, and many other settings can be changed with command line and configuration file options.

Signals

If *radiusd* is signalled with SIGHUP, it reinitialises by rereading the configuration file. All *Client*s, *Realm*s, *Handler*s, and the others defined in the old configuration file are lost, and new ones are configured. The effect of SIGHUP is expected to be the same as if you killed and then restarted *radiusd*.

If *radiusd* is signalled with SIGTERM, it exits gracefully.

If *radiusd* is signalled with SIGUSR1, it increases its current Trace level by 1. SIGUSR2 decreases it by one.

Environment variables

Environment variables can be used to add Radiator location to Perl module search path. It's possible to add new paths before and after the default search paths. See also *-I* command line option for adding to module path.

- PERL5LIB is processed by the Perl interpreter. These paths are prepended to the module search path before *radiusd* is started.
- RADIATOR_INC, RADIATOR_INC_CARRIER, RADIATOR_INC_DIAMETER, RADIATOR_INC_GBA_BSF and RADIATOR_INC_SIM are appended to the module search path. They are processed immediately by *radiusd* when it starts.

PERL5LIB is the recommended environment variable for modifying module search path. Radiator packages that come in RPM, deb and other formats, do not set this variable. Radiator packages use environment variables starting with RADIATOR_INC. The value of these should be a single directory currently.

Tip

Value of PERL5LIB is a list of directories. The list separator depends on the platform, and it is a colon ':' on unix or a semicolon ';' on Windows.

Command line arguments

Command line arguments given to *radiusd* override global configuration parameter settings in the configuration file. For more information, see Section 3.7. Global parameters on page 29.

The arguments are:

```
radiusd [-I dirname] [-h] [-v] [-c]
    [-auth_port port,...] [-acct_port port,...]
    [-db_dir dirname] [-log_dir dirname]
    [-bind_address address,...]
    [-log_file filename] [-config_file filename] [-dictionary_file file,...]
    [-foreground] [-daemon] [-log_stdout] [-trace n]
    [-pid_file filename] [-no_pid_file]
    [-service] [-installservice] [-uninstallservice] [-servicename name]
    [-serviceperlargs perlargs]
    [globalvarname=value]
```

- *-h*

  This prints usage information and exits.

- *-I dirname*

  This prepends *dirname* to the module search path. This parameter is processed by *radiusd* before any environment variables that start with RADIATOR_INC.

- *-v*

  This prints version information and exits.

- *-c*

  This parses the configuration file, reports errors in the usual way, then exits.

- *-auth_port port,...*

  This specifies the ports to listen for Access-Requests and overrides *AuthPort*.

- *-acct_port port,...*

  This specifies the ports to listen for Accounting-Requests and overrides *AcctPort*.

- *-db_dir dirname*

  This specifies the database directory and overrides *DbDir*.

- *-log_dir dirname*

  This specifies the log file directory and overrides *LogDir*.

- *-log_file filename*

  This specifies the name of the log file and overrides *LogFile*.

- *-config_file filename*

This reads the filename as the configuration file. The default value depends on the operating system:

  - Unix: `/etc/radiator/radius.cfg`
  - Windows: `C:\Program Files\Radiator\radius.cfg`

- *`-dictionary_file filename,...`*

  This specifies the name of one or more dictionary file and overrides *`DictionaryFile`*.

- *`-foreground`*

  This runs *`radiusd`* in the foreground, not as a daemon. The default behaviour is to run as a daemon.

- *`-daemon`*

  This forces *`radiusd`* to run as a daemon in the background, regardless of the setting of *`Foreground`* in the configuration file..

- *`-log_stdout`*

  This logs to STDOUT as well as to *`LogFile`*, if running in the foreground.

- *`-trace n`*

  This sets the trace level to *`n`* and overrides *`Trace`*.

- *`-pid_file filename`*

  This writes the PID to `filename` and overrides the global *`PidFile`* parameter setting. If this is not set, the *`PidFile`* configuration parameter value is used. For more information, see Section 3.7.15. PidFile on page 35.

- *`-no_pid_file`*

  This prevents creating a PID file.

- *`-bind_address address,...`*

  This specifies one or more IPv4 or IPv6 addresses to listen on and overrides *`BindAddress`*. For more information, see Section 3.5. Address binding on page 27.

- *`-service`*

  This is for specialised use on Windows only. It tells Radiator to run as a Windows Service. It requires Win32::Daemon, and requires that the service have been previously installed with *`-installservice`*.

- *`-installservice`*

  On Windows, this installs or reinstalls Radiator to run as a Windows Service. The service is configured to run Radiator with all the same arguments as was passed with *`-installservice`*, and it adds the *`-service`* flag. After this, the Radiator service appears in the Windows Service list as 'Radiator Radius Server'. The Service automatically starts next time the host is booted. Using this requires Win32::Daemon.

  On Windows 7 and others, the command prompt needs to be started with right click -> "Run as administrator". Otherwise the service does not get installed, even if the user has administrator privileges.

- *`-uninstallservice`*

  On Windows, this removes Radiator from Running as a Windows Service. Ensure the service is stopped before uninstalling it. This requires Win32::Daemon. If a *`-servicename`* argument was used with *`-installservice`*, then the same *`-servicename`* argument must be used to uninstall the service.

- *`-servicename`*

On Windows, if Radiator is being installed or uninstalled as a service, this argument specifies the name that the service is installed under. The default value is 'Radiator'. Specifying a different service name allows multiple Radiator services to be run at the same time.

- *-serviceperlargs args*

  On Windows, if Radiator is being installed or uninstalled as a service, this argument specifies extra arguments to pass to Perl when the service starts. This is useful for specifying an alternative install directory for the Radiator Perl modules:

  ```
  perl c:/Radiator/radiusd -installservice -serviceperlargs "-I
  c:/Radiator"
  ```

- *globalvarname=value*

  Defines the global variable called *globalvarname* to be defined as the string "value". The value can be accessed anywhere special formatting characters are permitted with *%{GlobalVar:globalvarname}*. This argument has exactly the same effect as the following setting in the configuration file:

  ```
  DefineFormattedGlobalVar globalvarname value
  ```

5. Details on starting Radiator during system start-up

Radiator AAA server is very reliable but it is possible that it is accidentally killed, or that a system problem causes it to exit. In a Unix environment, the preferred method is to use Linux **deb** or **PKG** packages and start and restart *radiusd* automatically with the *systemd* command. On Windows, you can run Radiator as a system service that will automatically start and restart.

For source code, custom and other advanced installations on Unix systems, see the subtopics in this chapter. There are a number of ways to make Radiator to start at system start-up and make it restart in case of problems.

Sometimes you may need to update Radiator start-up, for example, to enable advanced debugging. See the subtopics for more details on how to do this.

Attention

The commands to set up the different start-up methods vary between systems. Most likely you need to customise the methods for your system.

5.1. Systemd service unit file

The Radiator deb and RPM distributions are designed to use *systemd* service unit files. The default installation and its *systemd* configuration is described in installation section on page 3. Copies of the service unit files are also available in directory `/opt/radiator/radiator/goodies/`

When you need to locally customise the unit file settings, do not directly edit Radiator's files in `/usr/lib/systemd/system`. The local customisations, also called drop-in files, should go into `/etc/systemd/system/radiator.service.d/` directory. For example:

```
% cat /etc/systemd/system/radiator.service.d/stdout-stderr.conf
[Service]
# Standard output and error can be connected (redirected) to a file
# instead the usual default of journal
StandardOutput=file:/var/log/radiator/radiator-stdout.log
```

```
StandardError=file:/var/log/radiator/radiator-stderr.log
```

The above changes *systemd* behaviour to capture `stdout` and *stderr* and write them to separate files in the log directory. This allows for easier capture of LDAP level debugging that is not visible for Radiator, as described in Section 3.9.11. Debug on page 57 and Section 3.9.12. DebugTLS on page 57

The locally created drop-in file must end with the suffix `.conf`. Before using a new drop-in file, *systemd* likely requires '**`systemctl daemon-reload`**' command.

See the copies of service unit files in directory `/opt/radiator/radiator/goodies/` for additional customisation options, such as allowing Radiator to access the privileged `winbidd` socket for *`AuthBy NTLM`*.

5.2. System Service on Windows

On Windows you can configure Radiator to run as a Windows Service. It is automatically started at boot time, and you can start, stop, and pause it is the Services window, also know as the MMC Services snap-in. This is the preferred way to get Radiator to run automatically every time the Windows server is started.

1. Review the Windows Perl and Radiator installation instructions on page 11. Make sure you have `Win32::Daemon` Perl module installed.
2. Start a Command Prompt window. Adjust the paths below for your Perl installation. Create the Radiator service with:

   ```
   perl c:\perl64\bin\radiusd -install
   ```

   This installs Radiator as a Windows Service so that it uses the default configuration file in `C:\Program Files\Radiator\radius.cfg`
3. Open the Services snap-in. You see Radiator Radius Server as one of the available services. Using the Services window you can start, stop, pause, and disable the Radiator service.
4. Start the Service.
5. Test that the Service is running correctly by sending a test request with

   ```
   perl c:\perl64\bin\radpwtst -user mikem -password fred
   ```
6. Edit and test the configuration in `C:\Program Files\Radiator\radius.cfg`. You need to restart the Radiator service using the Services control panel after making any changes to the configuration file in order for the change to take effect.
7. Next time the computer reboots, the Radiator Service starts automatically.

Tip

Radiator runs successfully as a service if it runs properly using the same command line that you used during the installation testing. Try running Radiator from the root directory of the C: drive with something like **`perl c:\perl64\bin\radiusd`**. Since a Windows service has no “current directory” or “Current drive”, you must be very sure that your Radiator configuration file contains no relative file names. Every file name mentioned must be a fully qualified path name, including the drive name, such as.:

```
DbDir       C:\Program Files\Radiator
```

Tip

A Windows Service usually runs as the System User, not as a logged-in user. To run as a Service, configure your Radiator so that it does not rely on remote shares and other things that may not be accessible to the System User. Generally, this means that all Radiator configuration files, the Radiator program, and the like must be on the local disk.

5.3. Unix SYSV startup script

The Radiator distribution contains a Unix/Linux SYSV compatible startup script in `goodies/linux-radiator.init`.

To use this script to start and stop Radiator by hand or automatically at boot time, on Linux as root. Note that the exact location and command to enable the script may vary between different systems:

```
cp goodies/linux-radiator.init /etc/init.d/radiator
chmod 755 /etc/init.d/radiator
chkconfig -add radiator
```

This startup script responds to the following commands:

- **`/etc/init.d/radiator start`**

  Starts the Radiator server running in the background as a service. This is the default command which will be used to start Radiator automatically at boot time when the startup script is automatically run.

- **`/etc/init.d/radiator stop`**

  Stop the Radiator server.

- **`/etc/init.d/radiator restart`**

  Stop then restart the Radiator server.

- **`/etc/init.d/radiator reload`**

  Forces the Radiator server to reread its configuration file. It does this by sending a HUP signal to the server.

- **`/etc/init.d/radiator status`**

  Prints the current status of the Radiator server process.

- **`/etc/init.d/radiator traceup`**

  Increases the current Radiator server Trace level by one. It does this by sending a USR1 signal to the server.

- **`/etc/init.d/radiator tracedown`**

  Decreases the current Radiator server Trace level by one. It does this by sending a USR2 signal to the server.

5.4. Using restartWrapper

In a Unix environment, you can arrange for *`radiusd`* to be restarted automatically if it exits unexpectedly by using the *`restartWrapper`* script. *`restartWrapper`* is included in the `goodies/` directory of the Radiator distribution. It is not installed automatically, so if you want to use it, you will probably want to copy it to your local binaries directory. Radiator must be run in the foreground with the Foreground parameter or the -foreground argument. For more information, see Section 3.7.1. Foreground on page 29.

restartWrapper never terminates, so you will probably want to run it in the background with an ampersand (&), especially if you are calling it from a system boot script.

You will probably want to put a call to *restartWrapper* in your Unix system boot script so that *radiusd* is started automatically at boot time by *restartWrapper*. This will usually involve modifying `/etc/rc.local` or adding a new script to `/etc/rc2.d`, depending on what type of Unix you are running. See your system documentation for more details about system start-up scripts.

The arguments are:

```
restartWrapper [-h] [-delay n] [-mail address]
      [-min_interval n (default: 0)]
      [-sendmail path-to-sendmail]
      [-syslog facility.level (default: user.err)]
      [-logger path-to-logger (default: /usr/bin/logger)]
      "command to run"
```

- *-h*

  Print usage help message.

- *-delay n*

  Number of seconds to wait before restarting the command. Defaults to 10 seconds.

- *-min_interval n*

  Minimum interval in seconds between successive restart. Defaults to 0 seconds.

- *-mail address*

  The email address to send a message to when the command exits. By default, no email is sent.

- *-sendmail path-to-sendmail*

  Specifies an alternate path to the sendmail program which will be used to send email if the -mail argument is specified. Defaults to `/usr/lib/sendmail`.

- *-syslog facility.level*

  Specifies an optional syslog facility and level to be used to log messages using syslog. If this is not specified, syslog will not be used to log messages.

- *-logger path-to-logger*

  Specifies the syslog logger program which will be used to log syslog messages. Defaults to `/usr/bin/logger`.

- *"command to run"*

  This is the complete command that is to be run, including arguments if any. You should enclose the entire command in double quotes, especially if the command contains arguments that might be mistaken for arguments to restartWrapper. You will probably want to specify the full path to the command.

Example

Run *radiusd* with a specified config file. If it stops, send email to mikem@open.com.au and wait 2 seconds before restarting it.

```
restartWrapper -mail mikem@open.com.au -delay 2 \
      "/bin/radiusd -config_file /etc/radius.cfg \
      -foreground" &
```

Tip

Make sure that Radiator is running in “foreground” mode, either with -foreground in the command line arguments, or with Foreground the configuration file.

Tip

If you are starting restartWrapper from inside a Unix startup script, you will need to follow the command line with an ampersand (&), otherwise the unix startup script will never complete.

5.5. Using inetd

If you do not wish to use *restartWrapper* or *init*, you can instead arrange for the Unix *inetd(1)* super server to start *radiusd* the first time it is required (and to restart it if it stops unexpectedly). In order to do this, you must add a new line to the *inetd* configuration file (usually /etc/inetd.conf). You must also ensure that the radius port number you wish to use is configured into the /etc/services file. You must also ensure that Radiator is configured to run in the foreground with the Foreground parameter or the -foreground argument. For more information, see Section 3.7.1. Foreground on page 29.

The *inetd* line you add will look something like this (the line has been wrapped due to its length in this example):

```
# Start Radiator on demand
radius dgram udp wait root /bin/radiusd radiusd
      -config_file /etc/radius.cfg
      -foreground
```

After changing /etc/inetd.conf, you will need to tell *inetd* to reread its configuration file by sending it a HUP signal with something like

```
kill -HUP pid-of-inetd
```

Whenever a radius request is received and *radiusd* is not already running, *inetd* will automatically start *radiusd*. If *radiusd* stops some time later, *inetd* will restart it when the next request arrives. For more details on using and configuring *inetd*, consult your Unix vendor's documentation.

Tip

Make sure that Radiator is running in “foreground” mode, either with -foreground in the command line arguments, or with Foreground the configuration file.

5.6. Using init

On Unix systems that support it, you can start and restart Radiator automatically with *init(1)*. Add something like this to /etc/inittab:

```
ra:2345:respawn:/usr/bin/radiusd -config_file \
      /etc/raddb/radius.cfg -foreground
```

Tip

Make sure that Radiator is running in “foreground” mode, either with -foreground in the command line arguments, or with Foreground in the configuration file.

6. Testing tools

Radiator download package multiple tools for testing:

- *radpwtst* on page 449 for sending Radius requests
- *tacacsplustest* in goodies directory for sending TACACS+ requests
- *diapwtst* in goodies directory for sending Diameter requests
- *builddbm* on page 458
- *buildsql* on page 460

6.1. radpwtst

radpwtst is a testing tool that sends requests to a RADIUS server, such as Radiator, and waits for a reply. Use it to send Access-Request, Accounting-Request (Stop and Start), and Status-Server requests. *radpwtst* checks that your Radiator server, or any other RADIUS server, is configured and behaving correctly, and also for checking if user's password is correct.

By default, *radpwtst* sends an Access-Request, waits up to 5 seconds for a reply, sends an Accounting-Request (Start), waits for a reply, then sends an Accounting-Request (Stop), and waits for a reply. You can change this behaviour with the command line flags.

Tip

When *radpwtst* is run with *-interactive* flag, it queries the password without local echo. This allows you to specify the password without the command line *-password* option.

Tip

If a Framed-IP-Address is received in an Access Accept, then *radpwtst* uses the address in subsequent Accounting Starts and Stops, unless the *-framed_ip_address* flag has been set. This allows simple testing of address allocation modules and such.

Tip

A fundamental requirement of the RADIUS protocol is that the RADIUS Client (in this case *radpwtst*) and the RADIUS Server (in this case Radiator) must use the same shared secret. If Radiator keeps rejecting your request with a “Bad Password” message, even though you are sure the password is correct, it may be because the shared secrets are not correct. Check the "Secret" line in your Radiator config file, and perhaps use the *-secret* command line argument to *radpwtst*. If you do not specify a *-secret* argument to *radpwtst*, it uses **mysecret** by default

The arguments to *radpwtst* are:

```
radpwtst  [-h] [-time] [-iterations n] [-iteration_delay f] [-timestamps]
          [-log_microseconds] [-trace [level]] [-notrace] [-onlyfailed]
          [-print_stats] [-user username] [-password password]
          [-s server] [-secret secret] [-auth_port port] [-acct_port port]
          [-noauth] [-noacct] [-nostart] [-nostop] [-alive] [-status]
          [-chap] [-chap_nc] [-mschap] [-mschapv2] [-sip]
          [-eapmd5] [-eapotp] [-eapgtc] [-eapfastgtc] [-leap]
          [-motp_secret xxxxxxxxxxxxxxx] [-eaphex xxxxxxxxxxxx]
          [-interactive] [-code requestcode] [-accton] [-acctoff]
          [-identifier n] [-no_random] [-framed_ip_address address]
          [-state state] [-useoldascendpasswords] [-incrementuser]
          [-nas_ip_address address] [-nas_identifier string]
          [-nas_port port] [-nas_port_type type] [-service_type service]
          [-calling_station_id string] [-called_station_id string]
          [-session_id string] [-session_time n]
          [-delay_time n] [-input_octets n] [-output_octets n]
          [-timeout n] [-noreply] [-retries n] [-dictionary file,file]
          [-class string] [-message_authenticator] [-no_message_authenticator]
          [-raw data] [-rawfile filename] [-rawfileseq filename]
          [-outport port] [-bind_address address]
          [-options optionfile] [-gui]
          [attribute=value]...
```

- *-h*

  This prints the usage log and exits.

- *-time*

  This is an alias for *-print_stats*.

- *-iterations n*

  This sends all the selected requests n times, instead of just once.

- *-iteration_delay f*

  This option makes *radpwtst* to wait for specified amount of time between iterations. For example, when setting *iteration_delay* to **0.01**, *radpwtst* waits 0.01 seconds between iterations. This option is useful in testing purposes or when packet rate needs to be limited. Note that delay can be specified as float.

- *-timestamps*

  This includes a time stamp in announce messages. This option is automatically enabled when *iterations* is set to a value larger than **1**.

- *-log_microseconds*

  Using this, the timestamps are logged in microseconds instead of seconds.

- *-trace [n]*

  This prints useful trace information, including the full contents of all transmitted and received requests. The default is to print limited information from the reply. Trace level 5 produces hex packet dumps of requests and replies. The trace level is optional and the default value is **1**.

- *-notrace*

  The trace information is not printed. The default is to print limited information from the reply.

- *-onlyfailed*

  This shows only the failed requests.

- *-print_stats*

  Using this, *radpwtst* prints the the statistics of all requests and elapsed time taken to send and receives all iterations when it is finished and calculates packet rate for packets sent.

  This is useful for testing purposes, since it measures how fast the RADIUS server handles requests. If Perl `Time::HiRes` module is available, the elapsed time is printed with sub-second resolution. This module is available in all recent Perl distributions.

  To get useful values, the number of iterations must be large enough, for example, **8000**.

- *-user username*

  This tags the requests with User-Name of *username*. The default value is **mikem**.

- *-password password*

  In Access-Requests, the password is *password*. The default value is **fred**.

- *-s server*

  This sends all the requests to the server, which can be either the IP address or the DNS name of the host where the destination RADIUS server runs. The default value is **localhost**.

- *-secret secret*

  This uses secret as the shared secret. The default value is **mysecret**.

- *-auth_port port*

  This is the port to use for authentication requests . The default value is **1645**.

- *-acct_port port*

  This is the port to use for accounting requests. The default value is **1646**.

- *-noauth*

  Access-Request is not sent.

- *-noacct*

  Accounting-Request is not sent.

- *-nostart*

  Accounting-Request Start is not sent.

- *-nostop*

  Accounting-Request Stop is not sent.

- *-alive*

  This sends an Accounting-Request with Acct-Status-Type of Alive.

- *-status*

  This sends a Server-Status. The contents of the reply are printed.

Note

The Status-Server RFC requires Message-Authenticator. In most cases you need the `-message_authenticator` option also.

- `-chap`

  Authentication is done with CHAP, instead of PAP.

- `-chap_nc`

  Authenticate with CHAP, instead of PAP, with the CHAP Challenge in the authenticator, and not in a separate CHAP-Challenge attribute.

- `-mschap`

  Authentication is done with MSCHAP, instead of PAP or CHAP. Requires Digest-MD4-1.0 or better from CPAN. For more information about CPAN, see Section 2.1.2. CPAN on page 3.

- `-mschapv2`

  Authentication is done with MSCHAP V2, instead of MSCHAP, PAP or CHAP. This requires Digest-MD4 version 1.1 or better and Digest-SHA version 5.0 or better from CPAN. For more information about CPAN, see Section 2.1.2. CPAN on page 3.

- `-sip`

  SIP Digest is done authentication as per `draft-sterman-aaa-sip-00.txt`. This requires special attributes in the additional `dictionary.sip` in your distribution, so it should be used with `-dictionary dictionary,dictionary.sip`.

- `-eapmd5`

  Authentication is done with EAP-MD5. This usually involves 2 requests being sent to the server. The first is the EAP Identity, the second is the EAP-MD5 response.

- `-eapotp`

  Authentication is done with EAP-One Time Password. This usually involves 2 requests being sent to the server. The first is the EAP Identity, the second is the EAP-One Time Password response.

- `-eapgtc`

  Authentication is done with EAP-Generic Token Card. This usually involves 2 requests being sent to the server. The first is the EAP Identity, the second is the EAP-Generic Token Card response.

- `-eapfastgtc`

  This is similar as `-eapgtc` argument: authentication is done with EAP-Generic Token Card. This usually involves 2 requests being sent to the server. The first is the EAP Identity, the second is the EAP-Generic Token Card response. However, this uses RFC 5421 EAP-FAST-GTC response format.

- `-leap`

  EAP-LEAP authentication is done. This usually involves 3 requests being sent to the server. The first is the EAP Identity, the second is the LEAP client response and the third is the LEAP Access Point Challenge.

- `-motp_secret xxxxxxxxxxxxxxx`

  This makes Mobile OTP request using the password as PIN and motp_secret as the MOTP secret key.

- `-eaphex xxxxxxxxxxx`

This adds an EAP-Message attribute to the request. Argument is the message contents in hex. The correct Message-Authenticator is automatically added.

- *-interactive*

  This displays the Reply-Message, reads a new password from STDIN, and sends a new Access-Request, automatically copying any State attribute to the new request. This flag is useful for testing methods like <AuthBy ACE> which use Access-Challenge to prompt the user during a series of steps in an authentication conversation.

  This flag is also useful if password needs to be kept secret. When -interactive is set, password is read without local echo.

  Requires Perl module Term::Readkey on Windows. Some Unix-based systems are supported directly but Term::ReadKey is recommended for cross platform support.

- *-code requestcode*

  This tells *radpwtst* to send (in addition to any other request required) a RADIUS request with the given code name. Code names such as Ascend-Access-Next-Code, Disconnect- Request and Change-Filter-Request are all supported. Note that *-code Status- Server* is identical in meaning to *-status*.

- *-accton*

  This sends Accounting-On request.

- *-acctoff*

  This sends Accounting-Off request.

- *-identifier n*

  This is the identifier number of a single RADIUS packet.

- *-no_random*

  This forces *radpwtst* to use fixed values for RADIUS authenticator with different CHAP methods. This allows repeating tests with known values.

- *-framed_ip_address address*

  Access requests are sent with the given Framed-IP-Address. The default value is **0.0.0.0**. If the address is **0.0.0.0**, it is sent in the request. By default, *radpwtst* takes notice of any Framed-IP-Address returned in an Access-Accept, and uses it in subsequent Accounting Stops and Starts. Setting *-framed_ip_address* causes the same address to be used for all Accounting Stops and Starts.

- *-state state*

  This adds the string as State attribute.

- *-useoldascendpasswords*

  This makes *radpwtst* to encode passwords using the old (non-RFC compliant) method that Ascend used to use for some NASs. The default is to use RFC2865-compliant algorithm.

- *-incrementuser*

  This increments the user name on each round. If the user name on the first round is **mikem001**, it changes automatically to **mikem002** on the second round.

- *-nas_ip_address address*

  Access and Accounting requests have NAS-IP-Address of address. The default value is **203.63.154.1**.

- *-nas_identifier identifier*

Access and Accounting requests have NAS-Identifier of identifier. The default value is **203.63.154.1**.

- *-nas_port port*

  Access and Accounting request have NAS-Port of *port*. The default value is **1234**.

- *-nas_port_type type*

  Access and Accounting request have NAS-Port-Type of *type*. The default value is **Async**.

- *-service_type service*

  Access and Accounting request have Service-Type of *service*. The default value is **Framed-User**.

- *-called_station_id string*

  Access and Accounting requests have Called-Station-Id of *string*. The default value is **123456789**. If set to an empty string, Called-Station-Id is not included in the request.

- *-calling_station_id string*

  Access and Accounting requests have Calling-Station-Id of *string*. The default value is **987654321**. If set to an empty string, Calling-Station-Id is not included in the request.

- *-session_id string*

  Accounting request has Acct-Session-ID of *string*. The default value is **00001234**.

- *-session_time n*

  Accounting request has Acct-Session-Time of n. The default value is **1000**.

- *-delay_time n*

  Accounting request has Acct-Delay-Time of *n*. The default value is **0**.

- *-input_octets n*

  Accounting request has Acct-Input-Octets of *n*. The default value is **20000**.

- *-output_octets n*

  Accounting request has Acct-Output-Octets of *n*. The default value is **30000**.

- *-timeout n*

  This specifies the time in seconds that *radpwtst* waits for a reply. The default value is **5** seconds. If you specify **0**, it does not wait for a reply at all.

- *-noreply*

  When using this, no reply is waited before sending another request.

- *-retries n*

  If there is no reply, send up to *n* retries . The default value is **0** and no retries are sent.

- *-dictionary file,file*

  This uses *file* as the dictionary file. Multiple dictionary files can be specified as comma-separated file names. If *-dictionary* is not specified, *radpwtst* loads automatically for the first file that exists from this list ($radpwtstdir is the location where *radpwtst* resides):

  – `$radpwtstdir/dictionary`

  – `/etc/radiator/dictionary`

  – `/usr/local/etc/raddb/dictionary`

  - `/usr/local/etc/radiator/dictionary`
  - `/opt/radiator/radiator/dictionary`
  - `C:\Program Files\Radiator\dictionary`

- *`-class string`*

  This makes *`radpwtst`* to send string as the Class attribute in any accounting requests. Class defaults to the Class returned by any previous access-accepts.

- *`-message_authenticator`*

  This sends a correctly calculated Message-Authenticator attribute with the request.

  Note

  Some requests automatically add a Message-Authenticator. For example, EAP requires Message-Authenticator.

- *`-no_message_authenticator`*

  Do not automatically add a Message-Authenticator attribute.

- *`-raw data`*

  This sends raw data literally. An example of suitable raw data is trace 5 packet dump output. White space in the data is ignored.

- *`-rawfile filename`*

  This reads raw data from file called *`filename`* and send it literally. Raw data can be split to multiple lines

- *`-rawfileseq filename`*

  Read a sequence of raw data from file called *`filename`* and send it literally. The requests are separated with delimiter 'NewPacket'.

- *`-outport port`*

  This reads *`radpwtst`* to send requests from the given port. Port can be a port number or a port service name as used in `/etc/services` or it equivalent on your system. The default value is **`0`**, meaning allocate a random port.

- *`-bind_address address`*

  This m *`radpwtst`* to send requests through the network interface for the given IP address. Requests appear to originate from the specified IP address The default value is **`0.0.0.0`**, which means the default address of the default network interface. If the destination address (such as the *`-s`* flag) is an IPv6 address and *`-bind_address`* is specified, *`bind_address`* must also be an IPv6 address.

- *`-options optionfile`*

  This reads the command line options from the file specified with *`optionfile`*. For more information, see Section 6.1.1. radpwtst option file on page 456.

- *`-gui`*

  This presents A GUI that allows easy interactive testing. This GUI runs on Unix, it is not yet available on Windows hosts. Requests are sent when the Send button is pressed, and the GUI stays up after the requests have been sent, so you can send more. Requires Perl Tk module.

- *attribute=value*

  You can force any number of additional attributes to be sent in each request by naming them with their values on the command line. *attribute* must be the name of an attribute in your dictionary, and value must be a valid value for that attribute.

6.1.1. radpwtst option file

radpwtst options can be placed in an option file. The file format is one option per line. Leading and trailing white space is ignored. To preserve leading or trailing white space, use double quotes (") to surround the value. Empty lines and lines with # as first nonwhite space character are ignored. Use only one space between option name and value. See the example below.

```
# Set some defaults. Note: just one space between
# the option name and value
-trace 4
-nas_identifier " name with leading and trailing spaces "

# Attributes can have whitespace in the value only. Surround
# the value with " if there is trailing or leading space
Connect-Info=52100/31200 V91
```

radpwtst looks for options in `/etc/radpwtstrc` and `$HOME/.radpwtstrc`. It loads first the options file and then processes command line options. You can override options specified in the options file using command line options. For example, if the default trace level is 4 in the options file, you can temporarily use trace level 3 by using command line option *-trace 3*. This allows you to define the default parameters in the options file, but you can still temporarily use different values with command line parameters.

6.1.2. radpwtst examples

Send Access-Request and Accounting Start to server oscar.open.com.au for user mikem@your.realm and password jim. Authenticate with CHAP. Make sure the request includes Connect-Info of 52100/31200 V91:

```
radpwtst -s oscar.open.com.au -nostop -chap
      -user mikem@your.realm -password jim
      Connect-Info="52100/31200 V91"
```

Send Server-Status request to fred.open.com.au, print the reply:

```
radpwtst -status -noauth -noacct -s fred.open.com.au
      -trace -message_authenticator
```

Send 1000 Access-Requests to 1.2.3.4 for user fred, password jim, and print out how long it took:

```
radpwtst -iterations 1000 -noacct -user fred
      -password jim -time
```

Send a Disconnect-Request:

```
radpwtst -noacct -noauth -code Disconnect-Request NAS-Port=1234
```

Make the GUI appear for extended interactive testing:

```
radpwtst -gui
```

Send an Access-Request to an IPv6 address:

```
radpwtst -noacct -s 2001:db8:100:f101::1
```

Send an Access-Request to the localhost loopback address using IPv6:

```
radpwtst -noacct -s ::1
```

6.1.3. radpwtst GUI

The *radpwtst* program will present a Graphical User Interface (GUI) when it is started with the *-gui* flag. If you select this option, the GUI will stay visible until dismissed. This allows you to send a number of different requests with a single mouse click, and to quickly and easily change the attributes to be sent. These features allow easy testing of Radiator (or any other RADIUS server for that matter), particularly when configuring a new realm or user.

Note

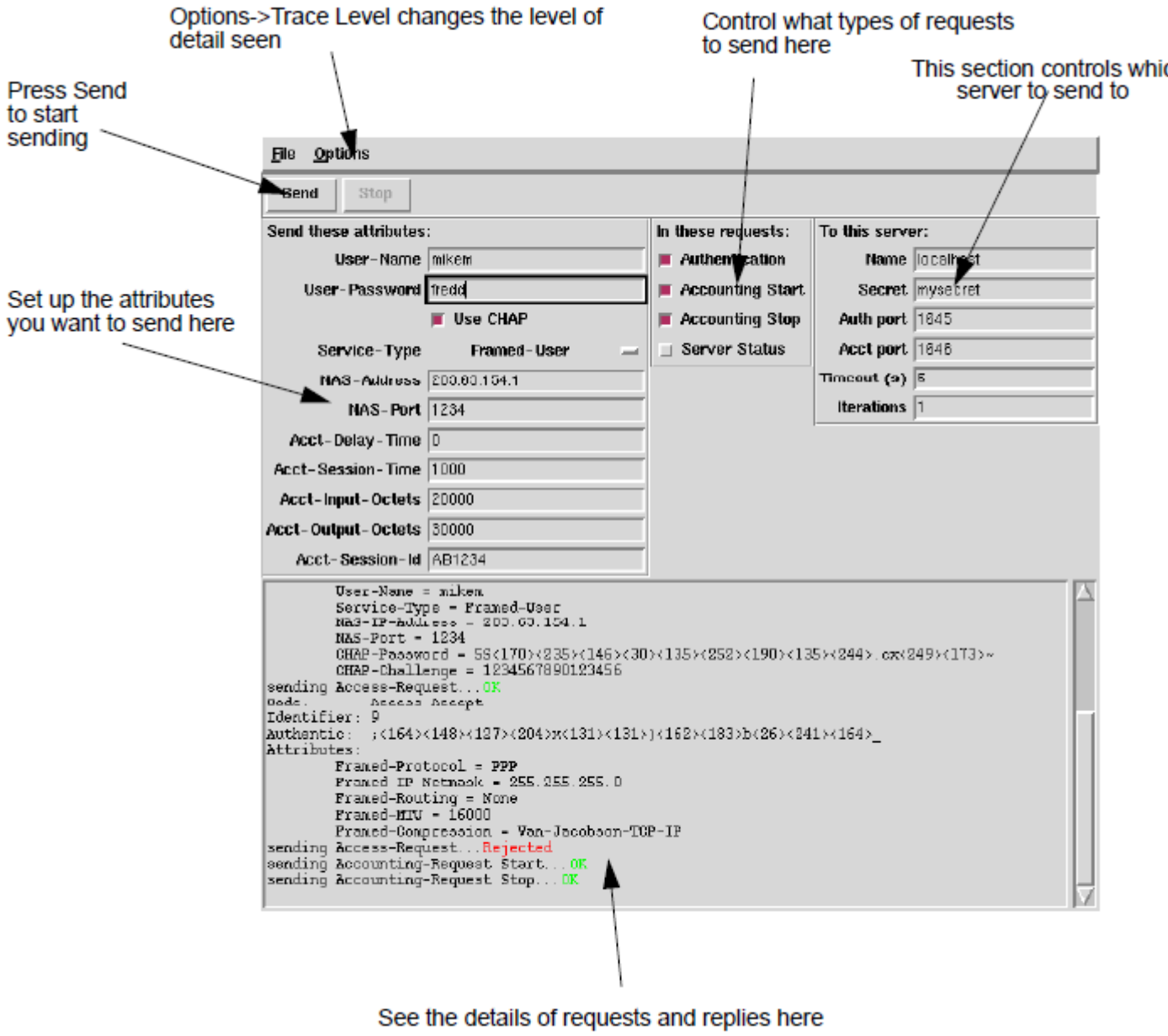
radpwtst -gui requires the Perl Tk module to be installed on your host machine.

The command line flags given to *radpwtst* are used to preconfigure the values you see in the interface. You can change the values at any time to affect the next request to be sent. Press the Start button to commence sending, and the Stop button to stop sending.

On each iteration, radpwtst will send one of each of the requests selected in the middle panel. The requests will contain attributes and values configured in the first panel, and the requests will be sent to the server and ports configured in the third panel. Not all attributes are sent in every request type: only the usual ones for that request type.

In the lower panel, you can trace the progress of each request and the reply, You can control the level of detail seen for each request and reply with the Options->Trace Level menu.

Figure 4. radpwtst Graphical User Interface



6.2. builddbm

builddbm is a testing tool that creates and updates DBM format user database files from flat file format user database files. For more information, see Section 9.3. DBM user database on page 491 and Section 9.2. Flat file user database on page 490. It can also be used to print or delete the information for a single user from a DBM file.

DBM files should be used with Radiator if you require fast authentication, but also need to change your user database on the fly while Radiator is running, and you do not or cannot use <AuthBy SQL> or <AuthBy FILE> in Nocache mode.

Radiator will choose the best format of DBM file available to you, depending on which DBM modules are installed on your machine.

Tip

You can force it to choose a particular DBM file format by using the -t flag.

The arguments are:

```
builddbm [-z] [-u] [-f flatfile] [-d user] [-l user]
[-p]
[-t type] dbfile
```

- *-z*

  This deletes all entries from the database before processing other commands.

- *-u*

  This updates the mode and replaces user entries that already exist in the DBM file rather than showing an error.

- *-f flatfile*

  The name of the flat file format user database to be used to populate the DBM file. The default value is **dbfile**, i.e. the name of the DBM file, without the `.pag` or `.dir` suffixes.

- *-d user*

  This deletes the entry for *user* from the DBM file.

- *-l user*

  This prints out the entry for *user* in a format that could be re-imported into *builddbm*.

- *-p*

  This prints the contents of the DBM file in standard Livingston flat file format. The check and reply attributes of all users in the database are printed out in no particular order. The printed output goes to **stdout**.

- *-t dbmtype*

  This forces *builddbm* to use a particular format of DBM file. The value of *dbmtype* can be AnyDBM_File, NDBM_File, DB_File, GDBM_File, SDBM_File or ODBM_File. Defaults to AnyDBM_File, which selects the best format on the host machine.

- *dbfile*

  The base name of the DBM files to create or use. The actual filenames depend on the DBM module that Perl has selected, but, it is usually something like `dbfile.dir` and `dbfile.pag`. This is a mandatory parameter.

Example

Rebuild the entire DBM database in `users.dir` and `users.pag` from the users file, clearing old entries first:

```
builddbm -z users
```

Example

Update the Berkeley DB format database files `all.db` with the users in the file *users*:

```
builddbm -u -f users -t DB_File all.db
```

Example

Print the entry associated with the user *mikem* in the DBM files staff.dir, staff.pag:

```
builddbm -l mikem staff
```

6.3. buildsql

buildsql is a testing tool that creates or updates an SQL authentication table from the contents of a Unix password file or from a (Livingston) standard RADIUS users file. It can also be used to print or delete the information for a single user from an SQL authentication table.

Use an SQL database with Radiator if you require fast authentication, large user populations, and also need to change your user database on the fly while Radiator is running, and you do not or cannot use <AuthBy DBM> or <AuthBy FILE> in Nocache mode.

By default, *buildsql* connects to the SQL database specified by the *-dbsource*, *-dbusername*, and *-dbauth* command line flags. You must specify these flags. For more information about setting these flags for different database vendors, see Section 19. Using SQL with various database vendors on page 527.

By default, *buildsql* inserts or updates records in a table called SUBSCRIBERS, but you can change this with a command line flag. By default, it only affects four columns in the table: USERNAME, PASSWORD, CHECKATTR, REPLYATTR, but you can change this with command line arguments. All other columns are unaffected by *buildsql*, so you can have arbitrarily complicated tables. You can change the names of the columns that *buildsql* uses with command line arguments. The default names are compatible with the default names used by the SQL authentication module.

The arguments are:

```
buildsql [-z] [-u] [-f] [-d user] [-l user] [-v]
-dbsource dbi:drivername:option
[-dbusername dbusername] [-dbauth auth]
[-password | -dbm | -flat]
[-tablename name]
[-username_column columnname]
[-password_column columnname]
[encryptedpassword]
[-checkattr_column columnname]
[-replyattr_column columnname] file ....
```

- *-z*

  This deletes all user entries from the database before processing other commands.

- *-u*

  This is the update mode. It replaces user entries that already exist in the database rather than shows error about constraint violations.

- *-f*

  This forces database update for non-defined fields.

- *-d user*

  This deletes *user* from the SQL database

- *-l user*

  This prints out the entry for *user* in a format that could be re-imported into *buildsql*.

- *-v*

  This prints out every SQL statement being issued before its executed

- *-dbsource dbi:drivername:option*

  This specifies the data source name of the database to connect to. Must be specified.

- *-dbusername username*

  This specifies the user name to use to connect to the SQL database.

- *dbauth password*

  This specifies the password for *dbusername*. Not required for some database types.

- *-password*

  The source files are in Unix password file format. For more information, see Section 9.4. Unix password file on page 491.

- *-dbm*

  The source files are in DB file format. For more information, see Section 9.3. DBM user database on page 491.

- *-t dbmtype*

  Forces *buildsql* to use a particular format of DBM file. The value of *dbmtype* can be *AnyDBM_File*, *NDBM_File*, *DB_File*, *GDBM_File*, *SDBM_File*, or *ODBM_File*. The default value is *AnyDBM_File*, which selects the best format on the host machine.

- *-flat*

  The source files are in standard RADIUS flat file format. For more information, see Section 9.2. Flat file user database on page 490. This is the default. If no input file type is specified, *-flat* is assumed.

- *-tablename name*

  This specifies the name of the database table to use. The default value is **SUBSCRIBERS**.

- *-username_column columnname*

  This specifies the name of the column where the user name are stored. The default value is **USERNAME**.

- *-password_column columnname*

  This specifies the name of the column where the passwords are stored. The default value is **PASSWORD**.

- *-checkattr_column columnname*

  This specifies the name of the column where the Check Items are stored. The default value is **CHECKATTR**.

- *-replyattr_column columnname*

  This specifies the name of the column where the Reply Items are stored. The default value is **REPLYATTR**.

- *-encryptedpassword*

  This handles and prints all passwords as if they were encrypted. When printing passwords with *-l*, the password is given with Encrypted-Password.

If neither *-password* or *-dbm* is specified, the input files are assumed to be in flat file format. For more information, see Section 9.2. Flat file user database on page 490.

Example

Rebuild the entire SQL database from the `/etc/passwd` file, clearing the old entries first. Then connect to an Oracle database SID called *`osc`* as user system and password manager. Use the default table and column names:

```
buildsql -z -dbsource dbi:Oracle:osc \
      -dbusername system -dbauth manager \
      -password /etc/passwd
```

Example

Print out the attributes for user *`mikem`* in the same database:

```
buildsql dbsource dbi:Oracle:osc \
      -dbusername system -dbauth manager -l mikem
```

Example

Delete user *`mikem`* from the same database:

```
buildsql dbsource dbi:Oracle:osc \
      -dbusername system -dbauth manager -d mikem
```

7. Check and reply items

Check items and reply items are used to authenticate users when an Access-Request message is received. Different AuthBy modules use different methods to store the check and reply items, but regardless of the module and how the user information is stored, all such items are used in the same way.

The general form of a check and reply item is

```
attribute-name = value
```

Attribute-name must be an attribute defined in your dictionary. Value may be surrounded by double quotes ("). Value must be surrounded by double quotes if it contains a comma. If a value is surrounded by double quotes use backslash (\) to escape embedded double quotes. You can have binary characters in a quoted string by specifying the octal code, preceded by a backslash. The spaces around the equals sign are optional.

Multiple check or reply items can be combined on a single line if they are separated by commas. Thus the following are all legal:

```
User-Password = fred
User-Password="fred"
User-Password = "fred",Service-Type = Framed-User
Reply-Message="this, has commas, and quotes\" in it"
Tunnel-Server-Endpoint = "\000191.165.126.240 fr:20"
```

The RADIUS attributes in check and reply items must be a defined in your dictionary.

7.1. Check items

Check items is the name given to the RADIUS attributes in the Access-Request that is checked before the user is granted an Access-Accept. All the check items for a user are checked. If any check item does not match the access is denied. You can have multiple check items for the same attribute, and the check only passes if all check items pass.

Most check items permit exact match, alternation, or regular expression matches. For more information about using them, see Section 7.1.50. Any other attribute defined in your dictionary on page 479.

These check items can also be used in the request selection expression in a *<Handler>* clause. For more information, see Section 3.31. <Handler attribute=value,attribute=value, ...> on page 149.

Usually this is used to limit the conditions under which a user is permitted to log on to your system. The most common case is to require a User-Password or Encrypted-Password, and you may also want to limit access via certain NASs, or at certain times. You can use any RADIUS attribute as a check item, and there are also some special attributes that are handled within Radiator in order to provide extra ways of controlling user access.

Since different brands and models of NAS implement different subsets of the RADIUS specification, it is not possible here to describe all the things you can configure in your NAS with reply items. For more information, see your NAS vendor's documentation.

7.1.1. User-Password, Password

A (usually) plaintext password. Passes only if the given password matches that sent in the Access-Request. If CHAP-Password attribute appears in the request then CHAP authentication will be attempted. If MS-CHAP-Challenge and MS-CHAP-Response attributes appears in the request then MSCHAP authentication will be attempted. CHAP and MSCHAP authentication is only supported with plaintext or Rcrypt encrypted passwords. You may user either Password or User-Password as the attribute name, the effect is the same.

Radiator also supports HTTP Digest password authentication with plaintext passwords. Digest authentication is supported by some web servers (e.g. Apache) and some web proxies (e.g. squid).

User-Password can be in a number of formats, not necessarily in plaintext. Radiator looks for some special format passwords and interprets them as special encryptions. The following formats are supported, along with example versions of the password "fred".

- Standard Unix crypt.This format is also compatible with Unix password encryption as used in Netscape LDAP server. Passwords starting with a leading {crypt} or {CRYPT} are interpreted as a standard Unix crypt password, using the native version of crypt() on your platform.

  ```
  User-Password = {crypt}1xMKc0GIVUNbE
  ```

- Linux MD5 password hashing. Passwords starting with "1" are interpreted as hashed with Linux MD5 password hashing.

  ```
  User-Password = $1$cTpht$Obu9PLSMst1TDou.mN5bk0
  ```

- Linux SHA256 and SHA512 crypt. Passwords starting with “5” or “6” are interpreted as hashed with Linux SHA256 or SHA512 password hashing, respectively.

  ```
  User-Password =
        $5$cTpht$i4ihNcS7lC1orrwWu/IfHrhxdDIkjBu095szYO4AucD
  User-Password = $6$cTpht$Z2pSYxleRWK8IrsynFzHcrnPlpUhA7N9AM/
  8O8se885W45WHyJ2K6bXsygHI46.cjqgl2hucmKtX1shWTL1zU1
  ```

- Linux Blowfish crypt. Passwords starting with $2a$, $2x$ or $2y$ are interpreted as hashed with Linux Blowfish password hashing. Support for these algorithms depends on the system crypt() implementation support. See the system documentation on crypt() about caveats with these hashes.

- Netscape SHA password hashing as used in Netscape LDAP server. Passwords starting with {SHA}, {SSHA}, {sha}, or {ssha} are interpreted as being hashed with Netscape SHA hashing. (Requires Digest-SHA version 5.0 or later, and also Mime::Base64 from MIME-Base64-2.11.tar.gz).

  ```
  User-Password = {SHA}MQF6ciZl5K/OWGlQ9ClEptMx2r8=
  ```

```
User-Password = {SSHA}k1qAjger6rE9fhCrig+QPZ/HTrJhYWE=
```

- SHA-2 hashes SHA 256, 384 and 512. These are similar to {SHA} and {SSHA} above. See `goodies/sha.pl` and `goodies/ssha.pl` for a utility to generate hashes.

```
User-Password =
      {SHA256}0M/C5TGbgs3HGjOHPoJsk9fuETY/iskcT6Oiz80ihuU=
User-Password =
      {SSHA256}abN9UTbhi3evQvdk7uYNML+UMZn8/BnWdxJUApQ0NzGkLQTd
User-Password ={SHA384}QoAkviNtBCtNyjN+yAkEEL6ChjtUVFKDTKHrIlx/
YqIHrDG7Tx2eJhbBPKAX0mo5
User-Password =
{SSHA384}DLZqetLxS6JPok1QcugKji0U8lxt6Zq7SYoGoK5JRVeeOqCuGHwxXf
1ZYGLg8pXqgms3jw==
User-Password = {SHA512}NWbDPDXFm6JYe6wqgVJs8z6gkoER7Z4WFqpD/
P+8P10H4FjICJjNKGCVt1h61e3TUR/ZQ/19d0Ox3tckJiAm8w==
User-Password =
{SSHA512}u+34y2JyCKoVRty0ADABlzhETpPv1HnShr2427qjsn7tgSoOaP8cHB
J95GT28ENlA7vySsjBVOMiuPqk2qgPvJOV4IM=
```

- MD5 Hex digest. Passwords starting with {MD5} or {md5}. Note that all hex digits are required to be lower case.

```
User-Password = "{MD5}570a90bfbf8c7eab5dc5d4e26832d5b1"
```

- MD5 with Mime, as used in some other LDAP servers.

```
User-Password = "{MD5}qP0OV/oViFka8YbFMWEWeg=="
```

- Rcrypt reversibly encrypted passwords. This reversible encryption format depends on a secret key. Radiator includes a reusable code module (Radius::Rcrypt) with encrypt and decrypt routines that you can call from third party programs. Rcrypt passwords require that you define the RcryptKey attribute in the AuthBy clause. The leading string can be {rcrypt} or {RCRYPT}.

```
User-Password = "{rcrypt}nYXkJKLrxm/e3RfU0aT7w4al"
```

- A predigested hex MD5 signature of the concatenation of the user name, a realm and the correct password. This is only valid for Digest and SIP authentication. In this example, the user name is "mikem", the realm is "open.com.au" and the correct password is "fred".

```
User-Password = \
      "{digest-md5-hex}884663db69c36190cf4c05c068a1a303"
```

- MySQL hashed password, as produced by the MySQL password() function.

```
User-Password = "{mysql}0569ef75321b8fed"
```

- MD5 hashed password in the format used by old Netscape Mail Servers.

```
{NS-MTAMD5}
b6b49e37d494a09bfde663033274bc83cd1bf318fa32c5866166a7edcb1
e1c87
```

- A Django style password. For more information, see User authentication in Django [https://docs.djangoproject.com/en/dev/topics/auth/#passwords]

```
User-Password =\
      "sha1$a1976$065f52b49153328da76e13c2b462b860a70eb78b"
User-Password = "md5$a1976$e67d1ca20e9c28321b86e34076cc48ab"
```

- DEC Hashed Password as used by DEC VMS and maybe others. Contains the algorithm type number, a salt and the hashed password, separated by vertical bars. The valid algorithm numbers are 1 (PURDY), 2 (PURDY_V) or 3 (PURDY_S) The hashed password depends on the user name as well as the algorithm and salt, so the hashed passwords are not portable between users. In VMS, user names are by convention all uppercase, and passwords are case sensitive. In this example for user name MIKEM, the algorithm type number is 3 (PURDY_S), the salt is 1234, and the hashed password (fred) is 85ad61e72a41dec4.

```
User-Password = {dechpwd}3|1234|85ad61e72a41dec4
```

- An NT Hashed Password, as used by Microsoft, Samba and others. This format is compatible with Samba SMB passwords (either in a flat file or in LDAP). Such password hashes can be generated with the Samba *`mkntpwd`* program.

```
User-Password = {nthash}DCB8E94AC7D0AADC8A81D9C895ACE5F4
```

- A password encrypted with the Microsoft SQL `pwdencrypt()` function. Passwords encrypted with `pwdencrypt()` are case insensitive. Requires Digest::SHA module. Note that the encrypted password produced by `pwdencrypt()` is a 46 octet binary string. Radiator recognises the encrypted password as either 46 octets of binary or 92 octets of ASCII Hex characters. You can use the MS SQL '+' string concatenation operator to prepend the '{mssql}' to the encrypted password, e.g.

```
select '{mssql}' + password, ..... from .....
User-Password =
{mssql}01003A54FC73501798169BEC84C05CA0D2FBB70009C2556313DA7959
C1A798ECD34514694A13D29ED57BE9CBE5DA
```

- A flagged plaintext password.

```
User-Password = {clear}fred
```

- A PBKDF2 (Password-Based Key Derivation Function 2) derived password. Radiator currently supports password derivation with Pseudo Random Function (PRF) HMAC-SHA1 and the following password format (PRF:rounds:salt:hash). See `goodies/pbkdf2.pl` for the format details. Requires `MIME::Base64`.

```
User-Password = {PBKDF2}HMACSHA1:
9000:h9Pwh4tcu0w=:iN9vitCZ1mqBKEu21dlc0RW2tlc=
```

- Custom format for CheckPasswordHook. For more information, see Section 3.32.23. CheckPasswordHook on page 172.

```
User-Password = {OSC-pw-hook}.......
```

- Plaintext. Any other format is interpreted as a plain text password.

```
User-Password = fred
User-Password = "password with spaces"
```

7.1.2. Encrypted-Password

An encrypted password. Passes only if the password sent in the Access-Request matches the given encrypted password. Most types of encrypted password only support PAP, not CHAP, MSCHAP or MSCHAPV2 authentication. Passwords encrypted with NT Hashed passwords can support PAP, MSCHAP and MSCHAPV2 authentication.

Encrypted-Password understands a number of encrypted formats: SHA, MD5, MD5 Mime, DEC Hashed passwords, NT Hashed passwords and standard Unix crypt. All the following match the plaintext password "fred":

```
Encrypted-Password = "{SHA}k1qAjger6rE9fhCrig+QPZ/HTrJhYWE="
Encrypted-Password = "{crypt}1xMKc0GIVUNbE"
# This next one is also crypt:
Encrypted-Password = "1xMKc0GIVUNbE"
Encrypted-Password = "$1$cTpht$Obu9PLSMst1TDou.mN5bk0"
Encrypted-Password = "1xMKc0GIVUNbE"
Encrypted-Password = "{MD5}qP0OV/oViFka8YbFMWEWeg=="
Encrypted-Password = "{MD5}570a90bfbf8c7eab5dc5d4e26832d5b1"
Encrypted-Password = "{dechpwd}3|1234|85ad61e72a41dec4"
Encrypted-Password = "{nthash}DCB8E94AC7D0AADC8A81D9C895ACE5F4"
# This next one is also nthash:
Encrypted-Password = DCB8E94AC7D0AADC8A81D9C895ACE5F4
Encrypted-Password =
{mssql}01003A54FC73501798169BEC84C05CA0D2FBB70009C2556313DA7959
C1A798ECD34514694A13D29ED57BE9CBE5DA
```

If there is no indication of the encryption type in an Encrypted-Password, Radiator will assume it is a Unix crypt(3) password if it is 13 or 20 bytes long (20 bytes is the BSD/ OS DES extended format for crypt(3)), a binary NT hashed password if it is 16 bytes long and a hex encoded NT hashed password if it is 32 bytes long.

```
# Unix Crypt:
Encrypted-Password = 1xMKc0GIVUNbE
# Hex encoded NT Hashed password
Encrypted-Password = DCB8E94AC7D0AADC8A81D9C895ACE5F4
```

When Radiator authenticates an MSCHAP or MSCHAP2 request, the default encrypted password format is taken to be an MD4 hashed password, in the standard Windows NT hashed password format (either hex encoded or binary).

7.1.3. Realm

Checks that the realm in the login name matches. The realm is the part following the @ in the user name. You can use either exact match, alternation, or a regular expression.

```
Realm = open.com.au
```

7.1.4. ExistsInRequest

Checks for the presence of an attribute in a request. You can use either exact match or alternation. Attribute value does not matter; empty attributes match too. This type of check item is mostly useful in Handlers

```
<Handler ExistsInRequest = EAP-Message>
    # All EAP messages are handled here
</Handler>
```

```
<Handler ExistsInRequest = OSC-Rate-Limit-Day|OSC-Rate-Limit-Night>
    # Handler is selected when one or the both are present
</Handler>
```

7.1.5. Expiration, ValidTo

Specifies the account expiry date with an optional time component. Passes only if the current local date and time (according to the clock on the host where Radiator is running) is prior to the given date and time. If no time is given, it assumes midnight at the beginning of the date given. The check items Expiration and ValidTo have identical meaning. The following formats are supported:

- Mmm dd yy(yy) (hh:mm:ss)
- Mmm dd, yy(yy) (hh:mm:ss) Note the optional comma.
- dd Mmm yy(yy) (hh:mm:ss)
- yyyy-mm-dd (hh:mm:ss)
- dd/mm/yy(yy) (hh:mm:ss)
- dd.mm.yy (hh:mm:ss)

Some valid examples are:

- Dec 30 1998
- Dec 30, 2000
- 30 Dec 2000
- Dec 30, 2000 11:30:00
- 30 Dec, 2000 09:32:00
- 2002-02-02 23:00:00
- 30/12/2002 01:00:00
- 30.12.2002

All the following Expiration check items are equivalent.

```
Expiration = Jan 02 1999 23:30:00
Expiration = 1999-01-02 23:30:00
Expiration = 02/01/99 23:30:00
# Unix epoch seconds (seconds since midnight, Jan 1 1970):
Expiration = 915280200
```

Tip

You can use Session-Timeout="until ValidTo" or Session-Timeout="until Expiration" as a reply item, which will set the session timeout to be the number of seconds until the Expiration date and time. For more information, see Section 7.2.8. Session-Timeout on page 481.

7.1.6. ValidFrom

Specifies a starting validity date. The same date formats as ValidTo are supported.

```
ValidFrom Jan 02 1999 23:30:00
```

7.1.7. Auth-Type

Auth-Type triggers special behaviour for authenticating the user. The possible values are:

- Reject. Any access request will always be rejected. This is useful for temporarily disabling logins for a given user.
- Accept. Forces acceptance, regardless of any following check items. Use with caution.
- Reject:message. Same as for Reject, except that the message (which can be any string) will be sent back to the user in a Reply-Message (provided the enclosing Realm or Handler has RejectHasReason set). This may be useful for telling your user why their login has been rejected.
- Ignore. Any access request will always be ignored (i.e. no reply will be sent back to the NAS). This is sometimes useful for triggering special behaviour in cascaded AuthBy clauses.
- Anything else. Any other word specifies an Identifier in an AuthBy clause which will used to authenticate the user. The name is matched with the name specified in the Identifier parameter in an AuthBy clause. You can name any other type of AuthBy module, be it SQL, RADIUS, UNIX etc. Specifying Auth-Type for a user causes the authentication to be cascaded to another authentication module. You can cascade authentications like this to any arbitrary depth.

The Auth-Type check item is most useful when you want to have check items and/or reply items, but also want to authenticate with native Unix or NT passwords.

Checks all users using the authentication method that has the identifier System:

```
DEFAULT Auth-Type = System
```

If you want to temporarily disable logins for a single user:

```
username Auth-Type = Reject
```

This one rejects the user and tells them why:

```
username Auth-Type = "Reject:you did not pay your bill"
```

This will first authenticate with the Identifier System, and if they are also in the group "staticip", they will continue to be authenticated with the AuthBy clause that has the Identifier "statics":

```
DEFAULT Auth-Type=System, Group=staticip, Auth-Type=statics
```

7.1.8. Group

The meaning of *Group* depends on the type of module that is doing the authentication. For UNIX and SYSTEM, it means whether the user is a member of the group as defined by the group file (usually `/etc/group/`). *<AuthBy SYSTEM>* supports both numeric and symbolic group names.

For *<AuthBy SQL>*, it runs and uses the result of the *GroupMembershipQuery* if defined. For more information, see Section 3.41.19. GroupMembershipQuery on page 201.

For *<AuthBy LDAP2>*, it requires *GroupSearchFilter* and related options to be defined. For more information, see Section 3.47.15. GroupSearchFilter on page 232.

For other AuthBy modules, it has no meaning, and always causes a rejection.

```
Group = wheel
```

7.1.9. GroupList

This works similarly to *Group*, but succeeds if the user is in any of the space separated group names.

```
GroupList = "wheel dialupusers nocstaff"
```

7.1.10. Group-Authorization

When you need Group-based authorisation for a user, this parameter defines the AuthBy that does the checking.

For usage examples, see goodies/authorize-group1.cfg and goodies/authorize-group2.cfg.

7.1.11. Block-Logon-From

Specifies a time in the format 9:00 am or 15:22. Attempts to authenticate after this time of day will fail. If Block-Logon-To is also specified for a later time of day, access is blocked between those times. The time of day that is used is the local time on the host where Radiator is running. Block-Logon-From is superseded by the Time check item and will not be supported in the future. For more information, see Section 7.1.15. Time on page 470.

7.1.12. Block-Logon-To

Specifies a time in the format 9:00 am or 15:22. Attempts to authenticate before this time of day will fail. If Block-Logon-From is also specified for an earlier time of day, access is blocked between those times. The time of day that is used is the local time on the host where Radiator is running. Block-Logon-From is superseded by the Time check item and will not be supported in the future. For more information, see Section 7.1.15. Time on page 470.

```
Block-Logon-From = 9:00 am,
Block-Logon-To = 5.00 pm
```

7.1.13. Prefix

This check item checks for the presence of a certain prefix in the user name. If it is not present the check item will fail. If it is present, it will be removed from the user name and accepted. This is most useful in DEFAULT items to handle different variations of the same users name, but with different reply attributes.

In this example, there might be a user mikem in the System authentication method. These user entries will allow Pmikem to log in as a PPP user, and Smikem to login as a Slip user:

```
DEFAULT     Prefix = P, Auth-Type = System
            Service-Type = Framed-User,
            Framed-Protocol = PPP,
            Framed-IP-Address = 255.255.255.254,
            Framed-MTU = 1500
DEFAULT     Prefix = S, Auth-Type = System
            Service-Type = Framed-User,
            Framed-Protocol = SLIP,
            Framed-IP-Address = 255.255.255.254,
            Framed-Compression = None
```

Tip

Prefix should appear before any Auth-Type check items.

7.1.14. Suffix

This check item is very similar to Prefix above, but checks for the presence of a certain suffix in the user name. If it is not present the check item will fail. If it is present, it will be removed from the user name and accepted. This is most useful in DEFAULT items to handle different variations of the same users name, but with different reply attributes.

In this example, there might be a user mikem in the System authentication method. These user entries will allow mikem.ppp to log in as a PPP user, and mikem.slip to login as a Slip user:

```
DEFAULT     Suffix = .ppp, Auth-Type = System
            Service-Type = Framed-User,
            Framed-Protocol = PPP,
            Framed-IP-Address = 255.255.255.254,
            Framed-MTU = 1500
DEFAULT     Suffix = .slip, Auth-Type = System
            Service-Type = Framed-User,
            Framed-Protocol = SLIP,
            Framed-IP-Address = 255.255.255.254,
            Framed-Compression = None
```

Tip

Suffix should appear before any Auth-Type check items.

7.1.15. Time

This check item allows you to specify which times of day and which days of the week the user is allowed to log on. The Time check is preferred to the Block-Logon-From and Block-Logon-To check items, which will not be supported in the future.

The format consists of day specifiers followed by hours intervals. Multiple day specifications are permitted with multiple values separated by commas (if you use commas, the entire check item must be enclosed with double quotes ("). Authentication will be permitted if the current local time on the Radiator server is within at least one of the time intervals specified.

Day specifiers are Mo, Tu, We, Th, Fr, Sa, Su for the days of the week. Wk means Mo- Fr and Al means any day. Hours intervals are specified as HHMM-HHMM. Typical examples are:

```
Time = "MoTuWe0800-1400,Wk2200-0400"
Time = "Al1800-0600,Wk1000-1330"
```

Tip

You can use Session-Timeout="until Time" as a reply item, which will set the session timeout to be the number of seconds until the end of the valid time period. For more information, see Section 7.2.8. Session-Timeout on page 481.

7.1.16. Simultaneous-Use

Specifies the maximum number of simultaneous sessions permitted for this user. Radiator keeps track of the number of current sessions for a user by counting the Accounting Start and Stop requests received for that user.

The value of this check item is either an integer or the name of a file that contains an integer. You can use any of the special formatting characters in the file name. The file will be reread for each authentication: it is not cached, so using a file name can have a slight impact on performance.

Tip

You can set the maximum number of simultaneous sessions for all the users in a Realm or Handler by using the Handler MaxSessions parameter. In this case, the smaller of MaxSessions and the users Simultaneous-Use will apply. For more information about setting default limit within an AuthBy, see Section 3.32.14. DefaultSimultaneousUse on page 169.

```
Simultaneous-Use = 1
```

7.1.17. Connect-Rate

Specifies the maximum connection speed that this user is permitted to use. Uses the Connect-Info attribute in the request to determine the speed the user is requesting. If Connect-Info is not present in the request, looks for USR-Connect-Speed. Note: not all NASs send Connect-Info or USR-Connect-Speed in Access Requests. Check with your NAS vendor's documentation.

```
Connect-Rate = 28800
```

7.1.18. NAS-Address-Port-List

Specifies the name of a file that contains a list of permitted NAS address/port combinations. For more information about port list file format, see Section 9.7. Portlist file on page 492. The filename can contain any of the special file name characters. The contents of the file are read at most once, and the port list is cached inside Radiator. If the user is not attempting to log in on one of the permitted address/port combinations, they will be rejected.

```
NAS-Address-Port-List %D/portlist
```

Tip

You can limit users to a particular NAS, irrespective of the port by using the NASIP- Address check item, possibly with a regular expression.

Tip

You can limit users to a particular port or set of ports on all NASs by using the NAS-Port check item, possibly with a regular expression.

7.1.19. Client-Id

Specifies the name of a Client that the request must come from. You can use an exact, alternation or regular expression match. The match is against the Client name (i.e. the word following Client in a <Client xxxxxx> clause). Note that it is legal to use Client- Id=DEFAULT to match requests that arrive through a <Client DEFAULT> clause.

```
<Client nas1.open.com.au>
```

```
      ....
</Client>
<Client nas2.open.com.au>
      ....
</Client>
<Client DEFAULT>
      ....
</Client>

# This user can only log in through nas2.open.com.au:
username1 Password=fred,Client-Id=nas2.open.com.au
# And this one only through NASs other than nas1 and nas2
username2 Password=jim,Client-Id=DEFAULT
```

Tip

When you use the Client-Id check item, it matches against the name or address in the <Client xxxxxx> line, but not against any of the values listed in IdenticalClients.

7.1.20. Client-Identifier

Specifies the Identifier of a Client that the request must come from. You can use an exact, alternation or regular expression match. The match is against the Client Identifier (i.e. the value of the Identifier parameter in a <Client xxxxxx> clause).

```
# Several NASs at pop1
<Client nas1.open.com.au>
      IdenticalClients 1.1.1.1,1.1.1.2,1.1.1.3
      Identifier pop1
      ...
</Client>
# Several NASs at pop2
<Client nas2.open.com.au>
      IdenticalClients 2.1.1.1,2.1.1.2,2.1.1.3
      Identifier pop2
      ....
</Client>

# This user can only log in at NASs in pop1
username1 Password=fred,Client-Identifier=pop1
# And this one only through NASs in pop2
username2 Password=jim,Client-Identifier=pop2
```

7.1.21. NasType

Specifies that the request must come from a Client with the specified NasType. You can use an exact, alternation or regular expression match.

```
<Client xxx>
      NasType Livingston
      ....
</Client>
```

```
<Client yyy>
      NasType Ascend
      ....
</Client>

# This is can only log in through xxx, since it has a NasType
# of Livingston
username Password=fred,NasType=Livingston
```

7.1.22. Request-Type

Specifies that the type of the request must match the name given. You can use an exact, alternation or regular expression match. Typical values of request types are:

- **`Access-Request`**
- **`Accounting-Request`**

`Status-Server` is handled differently; it's not available as a *`Request-Type`* check item. This type of check item is mostly useful in Handlers for selecting special handling for particular request types:

```
<Handler Request-Type=Access-Request>
      # Accounting wont be handled here, only access requests
</Handler>
```

7.1.23. MS-Login-Hours

Specifies a map of permitted login hours per day. The value is an array of bits, one per hour starting at 0000 Sun UTC, 24 bits per day, 7 days. It is not an ASCII array, but an array of binary bytes. The format is exactly compatible with the LoginHours user attribute in Microsoft Active Directory, and can therefore be used when accessing Active Directory via LDAP:

```
<AuthBy LDAP2>
      Host uniform
      AuthDN cn=Administrator,cn=Users,dc=open,dc=com,dc=au
      AuthPasswordadmin
      BaseDN ou=csx users,dc=open,dc=com,dc=au
      ServerChecksPassword
      UsernameAttr sAMAccountName
      AuthAttrDef logonHours,MS-Login-Hours,check
</AuthBy>
```

7.1.24. Tagged Tunnel attributes and other tagged attributes

Radiator supports the IETF RFC 2868 standard tunnel attributes, and also permits you to specify tagged tunnel attributes. Tags are a method of grouping attributes in the same packet which refer to the same tunnel. A tag is an integer from 1 to 31 inclusive. In Radiator you can specify a tag for a tunnel attribute by prefixing it with the tag number and a colon. For example, here are 3 tunnel reply attributes, all with a tag of 1:

```
Tunnel-Type=1:L2F,
Tunnel-Client-Endpoint=1:xyz,
Tunnel-Password=1:1234
```

Note

If a tag and colon are not specified, the tag is assumed to be 0. A tag with value 0 in reply attributes may confuse some clients. We recommend using a non-0 value for reply attributes.

Radiator reply attributes that support tags are:

- Tunnel-Type
- Tunnel-Medium-Type
- Tunnel-Client-Endpoint
- Tunnel-Server-Endpoint
- Acct-Tunnel-Connection (also called Tunnel-ID)
- Tunnel-Password
- Tunnel-Private-Group-ID
- Tunnel-Assignment-ID
- Tunnel-Preference
- Tunnel-Client-Auth-ID
- Tunnel-Server-Auth-ID

7.1.25. EAPType

If the request is an EAP request, this check item matches the EAP type number. Available EAPType numbers include 4 (MD5-Challenge), 13 (TLS) and 26 (MSCHAP-V2). Not useful for tunnelling EAP types, such as TTLS or PEAP.

7.1.26. EAPTypeName

If the request is an EAP request, this check item matches the EAP type name. Available EAPType names include MD5, TLS and MSCHAP-V2. Not useful for tunnelling EAP types, such as TTLS or PEAP.

7.1.27. TunnelledByTTLS

Specifies that the current request was tunnelled by EAP-TTLS.

7.1.28. TunnelledByPEAP

Specifies that the current request was tunnelled by PEAP.

7.1.29. TunnelledByFAST

Specifies that the current request was tunnelled by EAP-FAST.

7.1.30. RecvFromAddress

Specifies the IP address of the RADIUS, Radsec, or TACACS+ client the request was received from. IPv4 and IPv6 addresses are supported. You can use an exact, alternation, or regular expression match.

```
RecvFromAddress=203.63.154.29
RecvFromAddress=::1
```

Tip

This can be useful in Handler clauses to provide special handling of requests received from a specific client:

```
<Handler RecvFromAddress=203.63.154.29>
      ....
<Handler>
```

7.1.31. RecvFromName

Specifies the DNS name of the RADIUS, Radsec, or TACACS+ client the request was received from. IPv4 and IPv6 addresses are supported. You can use an exact, alternation, or regular expression match.

```
# Exact match
RecvFromName=radserver.open.com.au
# Regular expression:
RecvFromName=/.*\.open\.com\.au/
```

Tip

This can be useful in Handler clauses to provide special handling of requests received from a specific client:

```
<Handler RecvFromName=radsec.open.com.au>
      ....
<Handler>
```

CAUTION

This does a reverse name lookup on the address and depending on your environment this may take a number of seconds to resolve.

7.1.32. RecvName

This parameter specifies the DNS name of the RADIUS, Radsec, or TACACS+ client that receives the request. It supports both IPv4 and IPv6 addresses. You can use an exact, alternation, or regular expression match. Here are examples of using different matches:

```
# Exact match
RecvName=radserver.open.com.au

# Regular expression:
RecvName=/.*\.open\.com\.au
```

Tip

In Handler clauses, it can be useful to provide special handling of requests received by a local interface with a certain name:

```
<Handler RecvName=radsec.open.com.au>
      ....
<Handler>
```

CAUTION

This does a reverse name lookup on the address and depending on your environment this may take a number of seconds to resolve.

7.1.33. RecvAddress

This parameter specifies the IP address of the RADIUS, Radsec, or TACACS+ client that receives the request. It supports both IPv4 and IPv6 addresses. You can use an exact, alternation, or regular expression match. Here are examples of using different matches:

```
RecvAddress=203.63.154.29
RecvAddress=::1
```

Tip

In Handler clauses, it can be useful to provide special handling of requests received by a specific local address:

```
<Handler RecvAddress=203.63.154.29>
      ....
<Handler>
```

7.1.34. RecvPort

This parameter specifies the port number of the RADIUS, Radsec, or TACACS+ client that receives the request. It supports both IPv4 and IPv6. You can use an exact, alternation, or regular expression match. Here are examples of using different matches:

```
RecvPort=1812
RecvPort=/164./
```

Tip

In Handler clauses, it can be useful to provide special handling of requests received by a specific port:

```
AuthPort 1645,1812
<Handler RecvPort=1812>
      ....
<Handler>
```

7.1.35. Max-All-Session

Compares the total of all session times for the user against the given number. If it is exceeded, the user will be rejected. Otherwise, the time left will be available for use with Session-Timeout = until ValidTo to limit the remaining time for the user. Supported with AuthBy SQL when AcctTotalQuery is defined.

7.1.36. Max-Hourly-Session

Compares the total of all session times for the user from the beginning of the current hour against the given number. If it is exceeded, the user will be rejected. Otherwise, the time left will be available for use with ‘Session-Timeout = until ValidTo’ to limit the remaining time for the user. Supported with AuthBy SQL when AcctTotalQuery is defined.

7.1.37. Max-Daily-Session

Compares the total of all session times for the user from the beginning of the current day against the given number. If it is exceeded, the user will be rejected. Otherwise, the time left will be available for use with ‘Session-Timeout = until ValidTo’ to limit the remaining time for the user. Supported with AuthBy SQL when AcctTotalQuery is defined.

7.1.38. Max-Monthly-Session

Compares the total of all session times for the user from the beginning of the current month against the given number. If it is exceeded, the user will be rejected. Otherwise, the time left will be available for use with ‘Session-Timeout = until ValidTo’ to limit the remaining time for the user. Supported with AuthBy SQL when AcctTotalQuery is defined.

7.1.39. Max-All-Octets

Compares the total of all traffic octets for the user against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalOctetsQuery is defined.

7.1.40. Max-All-Gigawords

Compares the total of all traffic for the user in GigaWords against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalGigawordsQuery is defined.

7.1.41. Max-Hourly-Octets

Compares the total of all traffic octets for the user from the beginning of the current hour against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalOctetsSinceQuery is defined.

7.1.42. Max-Hourly-Gigawords

Compares the total of all traffic GigaWords for the user from the beginning of the current hour against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalGigawordsSinceQuery is defined.

7.1.43. Max-Daily-Octets

Compares the total of all traffic octets for the user from the beginning of the current day against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalOctetsSinceQuery is defined.

7.1.44. Max-Daily-Gigawords

Compares the total of all traffic GigaWords for the user from the beginning of the current day against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalGigawordsSinceQuery is defined.

7.1.45. Max-Monthly-Octets

Compares the total of all traffic octets for the user from the beginning of the current month against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalOctetsSinceQuery is defined.

7.1.46. Max-Monthly-Gigawords

Compares the total of all traffic GigaWords for the user from the beginning of the current month against the given number. If it is exceeded, the user will be rejected. Supported with AuthBy SQL when AcctTotalGigawordsSinceQuery is defined.

7.1.47. Variable names prefixed with GlobalVar

The attribute to be compared is taken from a GlobalVar. GlobalVar variables can be set from the command line, or with the DefineFormattedGlobalVar parameter. This can be useful for activating different sets of users in multiple instances of Radiator.

```
# In the config file:
DefineFormattedGlobalVar system mysystem

# in a users file:
username Password=fred,GlobalVar:system=mysystem
```

7.1.48. Attributes prefixed with Reply

You can specify a check item that checks an attribute in the currently constructed reply (instead of the incoming request) by prefixing it with “Reply:”. This can be a convenient way to set up user profiles or similar.

```
# This will set up one of 2 different user profiles, depending
# on the value of the pseudo-attribute Profile, which was set in
# and earlier AuthBy:
DEFAULT     Reply:Profile=premium
            Session-Timeout=1000000
DEFAULT     Reply:Profile=cheap
            Session-Timeout=1000
```

7.1.49. Attributes prefixed with DiaRequest

You can specify a check item that checks a Diameter attribute in an incoming Diameter request. This can be useful for choosing a Handler based on Diameter attributes, for example:

```
<Handler DiaRequest:Auth-Application-Id=NASREQ>
```

or:

```
<Handler DiaRequest:Disconnect-Cause=CREDIT_CONTROL>
```

7.1.50. Any other attribute defined in your dictionary

Checking of all other attributes passes only if the corresponding attribute exists in the request and matches the value specified for the check item.

Radiator allows check items to be specified either as an exact match or as a Perl regular expression (regexp). Radiator regards check items whose value is surrounded with slashes (‘/’) as a regular expression. Anything else is regarded as an exact match.

- Exact match

  The check item will pass only if there is an exact match. The comparison is case sensitive. Radiator will look for an exact match if the value to be matched is not surrounded by slashes.

  ```
  NAS-IP-Address = 203.63.200.5
  Calling-Station-Id = 121284
  ```

- Alternation

  Specify multiple permitted values, separated by vertical bars (‘|’). The check item will pass if at least one of the permitted values is an exact match.

  ```
  Calling-Station-Id = 121284|122882
  ```

- Perl Regular expression

  If the check item is surrounded by slashes (‘/’), it is regarded as a Perl regular expression, and Perl is used to test whether the value of the attribute in the request matches the regexp. The expression modifiers ‘i’ (case insensitive) and ‘x’ (ignore white space in the pattern) are also permitted. Only either 'i' or 'x' modifier is allowed after the ending slash. Use (?...) syntax to enable or disable additional modifiers. Modifier 's' (dot matches also newline) is enabled by default in Radiator 4.24 and later.

  Perl regular expressions give you an enormous amount of power to control the conditions under which a user can log in. The first example below only matches if the user logs in from phone numbers that contain 95980981, 95980982, 95980983 or 95980984. The second example only matches of they log into a port number with one digit (i.e. ports 1-9).

  ```
  Calling-Station-Id = /9598098(1|2|3|4)/
  NAS-Port = /^\d\z/
  ```

  Tip

  Perl regexps are very powerful, but they also take some getting used to. You should use them carefully, and test to make sure they really do what you want. Consult some Perl manuals or a Perl guru for tips on writing regexps.

  Tip

  You can use the ‘i’ and ‘x’ pattern modifiers to get case-insensitive or extended expressions like this, to match a Class attribute that includes "myclass" without regard to case.

  ```
  Class = /myclass/i
  ```

Tip

You can set up “negative” matches (i.e. that only match if the check is not equal to some string) by using Perl negative lookahead assertions in a regexp. For example, this check item will match all Service-Types except for Framed-User:

```
Service-Type = /^(?!Framed-User)/
```

Tip

You can match a string that contains unprintable characters by using a character class negation. For example to match a user name that contains any character not in a-z, A-Z or 0-9:

```
User-Name = /[^a-zA-Z0-9]/
```

7.2. Reply items

Reply items are RADIUS attributes that are used to configure the Terminal Server or NAS when a user is successfully authenticated.

In order to determine what RADIUS attributes you need to configure your NAS, you will need to consult your NAS vendor's documentation. Some RADIUS attributes are common to all NASs, and can be used with any NAS, while some are specific to a particular vendor or model of NAS. All reply items that you use must be defined in your dictionary.

If the user is granted access, all the reply items will be returned in the Access-Accept message to the NAS. This is usually used to configure the NAS in the correct way for that user. Since different brands and models of NAS implement different subsets of the radius specification, it is not possible here to describe all the things you can configure in your NAS with reply items. Refer to your NAS vendor's documentation. Here is a typical example to configure a NAS for a dial-up PPP session:

```
Framed-Protocol = PPP,
Framed-IP-Netmask = 255.255.255.255,
Framed-Routing = None,
Framed-MTU = 1500,
Framed-Compression = Van-Jacobson-TCP-IP
```

Some reply items are given special treatment:

7.2.1. Framed-Group

The reply attribute is used in conjunction with FramedGroupBaseAddress in order to allocate an IP address for the user and return it in a Framed-IP-Address attribute. For more information, see Section 3.14.38. FramedGroupBaseAddress on page 112.

```
Framed-Group = 1
```

7.2.2. Ascend-Send-Secret

Causes the value to be encrypted with the Client's shared secret and returned to the Client.

```
Ascend-Send-Secret = mysecret
```

7.2.3. Tunnel-Password and other tagged attributes

Causes the password to be encrypted with the Client's shared secret according to RFC 2868. This is used by NASs for managing VPN tunnels. For more information, see Section 7.1.24. Tagged Tunnel attributes and other tagged attributes on page 473.

```
Tunnel-Password = 1:yourtunnelpassword
```

7.2.4. MS-CHAP-MPPE-Keys

Causes 2 session keys to be generated from the given string and then encrypted with the Client's shared secret according to RFC 2548. This is used by Microsoft MS-CHAP tunnelling. If the value is exactly 24 octets long, it is assumed to already contain 2 session keys, and will only be encrypted. Requires the Digest::MD4 Perl module to be installed.

```
MS-CHAP-MPPE-Keys = mymppekey
```

Tip

If you are doing MS-CHAP authentication with plaintext passwords and your NAS requires MS-CHAP-MPPE-Keys in the reply, then setting the AutoMPPEKeys parameter in your AuthBy clause will force Radiator to automatically reply with MSCHAP- MPPE-Keys set from the plaintext password.

7.2.5. MS-MPPE-Send-Key

Causes the key to be encrypted with the Client's shared secret according to RFC 2548. This is used by Microsoft MS-CHAP tunnelling.

```
MS-MPPE-Send-Key = mysendkey
```

7.2.6. MS-MPPE-Recv-Key

Causes the key to be encrypted with the Client's shared secret according to RFC 2548. This is used by Microsoft MS-CHAP tunnelling.

```
MS-MPPE-Recv-Key = myrecvkey
```

7.2.7. Fall-Through

This attribute is not actually returned to the NAS. Its presence causes Radiator to continue looking for a match with the next DEFAULT user name.

```
Fall-Through = yes
```

7.2.8. Session-Timeout

If your NAS supports the Session-Timeout RADIUS attribute, this attribute can be used in several ways.

As a simple number, it specifies the maximum time (in seconds) that the session is allowed to run for:

```
Session-Timeout=6300
```

With the special form "until nnnn", it will specify that the maximum time the session will last is until the time of day specified. The time is the local time (according to the host where Radiator is running). If the current time is after the time specified, the cutoff time will be that time on the following day. For example to specify that the maximum time for any session is up until 6am today:

```
Session-Timeout="until 0600"
```

With the special form "until Time", it will set the session timeout to be the number of seconds until the end of the Time check item that permitted the user to log in. This allows you to restrict users so they are only able to connect for the specified time period. In the following example, user fred can log on between 1000 and 1700 each week day, and they will be disconnected (by Session-Timeout) at 1700.

```
fred Password=jim, Time="Wk1000-1700"
      Session-Timeout="until Time"
```

With the special form "until Expiration" or “until ValidTo, it will set the session timeout to be the number of seconds until the Expiration or ValidTo check item that permitted the user to log in. This allows you to restrict users so they are only able to connect until the end of their account validity period. In the following example, user fred can log on up until January 1 2003 and they will be disconnected (by Session-Timeout) at midnight of December 31 2002.

```
fred Password=jim, ValidTo="2003-01-01"
      Session-Timeout="until ValidTo"
```

7.2.9. Exec-Program

This pseudo reply does not actually add any reply items to the reply. It runs the external program given by the argument and waits for it to terminate. It is only run during reply handling, which means that it is only run if the user successfully authenticates. This is a handy way to run an external program when a user successfully logs in. The argument may contain any special characters. For more information, see Section 3.3. Special formatters on page 21.

Tip

There is no guaranteed environment for the external program to run in, so you should specify a program with its full path name.

```
Exec-Program="/usr/local/bin/sendgreeting %u"
```

Tip

Radiator is blocked while the external program runs. On Unix, If you do not want Radiator to wait for the program to terminate, use an ampersand (&) at the end of the line:

```
Exec-Program="/usr/local/bin/myslowprogram %u &"
```

7.2.10. Ascend-Data-Filter, Ascend-Call-Filter

Radiator supports Ascend Binary Filters, which are given the type of ‘abinary’ in the dictionary. The standard dictionary contains the standard abinary attributes Ascend- Data-Filter and Ascend-Call-Filter.

Ascend Binary Filters are binary encoded strings. However Radiator allows you to create filters in a symbolic, textual form. See you NAS documentation about the meaning, construction and use of Ascend Binary Filters with your NAS.

Radiator supports three basic types of filter, each with slightly different syntax.

- IP Filter

  The general syntax of an IP filter is:

  ```
  ip dir action [dstip n.n.n.n/nn] [srcip n.n.n.n/nn] [proto [dstport cmp port]
  [srcport cmp port] [est]]
  ```

  - *dir* is **IN** or **OUT**, case insensitive
  - *action* is **FORWARD** or **DROP**. Case insensitive
  - *proto* is a protocol name, such as **ip**, **icmp**, **tcp**, lower case
  - *cmp* is a port comparison operator like **<**, **=**, **>** or **!=**
  - *port* is a defined port name or integer port number, such as **ftp-data**, **telnet**, **smtp**, lower case

  Examples:

  ```
  Ascend-Data-Filter = "ip in forward icmp"
  Ascend-Data-Filter = "ip in forward dstip 1.2.3.4/24 tcp"
  Ascend-Data-Filter = "ip in forward dstip 195.174.219.30 tcp
  dstport=20",
  ```

- Generic filter

  The general syntax of a generic filter is:

  ```
  generic dir action offset mask value [cmp] [more]
  ```

  - *dir* is **IN** or **OUT**, case insensitive
  - *action* is **FORWARD** or **DROP**, case insensitive
  - *offset* is an integer offset
  - *cmp* is **==** or **!=**

  Examples:

  ```
  Ascend-Data-Filter = "generic in forward 0 0 0"
  Ascend-Data-Filter = "generic in drop 0 ffff 0080 != more"
  ```

- IPX Filter

  The general syntax for an IPX filter is:

  ```
  ipx dir action [srcipxnet nnnn srcipxnode mmmmm [srcipxsoc cmp value]]
  [dstipxnet nnnn dstipxnode mmmmm [dstipxsoc cmp value]]
  ```

  - *dir* is **IN** or **OUT**, case insensitive
  - *action* is **FORWARD** or **DROP**, case insensitive
  - *cmp* is a comparison operator like **<**, **=**, **>** or **!=**

  Examples:

```
Ascend-Call-Filter = "ipx in forward srcipxnet 1 srcipxnode
0x11223344aabb srcipxsoc > abcd dstipxnet 5678 dstipxnode 0xaabbccddee00
dstipxsoc > 1234"
```

CAUTION

Radiator is very strict in its interpretation of filters. You cannot change the order of filter elements, but you can omit the ones shown in square brackets [...]

7.2.11. Any other attribute defined in your dictionary

Any other attribute will be returned to the client in the reply message. All attributes must be defined in your dictionary.

```
Framed-Protocol = PPP,
Framed-IP-Netmask = 255.255.255.0,
Framed-Routing = None,Framed-MTU = 1500,
Framed-Compression = Van-Jacobson-TCP-IP
```

Attributes that are defined as type ipaddr in the dictionary (e.g. Framed-IP-Address, Framed-IP-Netmask etc.) can be specified either in dotted-quad notation (e.g. 1.2.3.4) or as 4 bytes of binary. This is most useful if you want to keep addresses in a database in binary format.

Attributes that are defined as type integer can be specified with either:

- A RADIUS attribute value name
- decimal (i.e. base 10) integer
- A hex (i.e. base 16) integer (with a leading 0x)

Therefore, the following reply items are all equivalent:

```
Framed-Protocol = PPP
Framed-Protocol = 1
Framed-Protocol = 0x01
```

8. Rewriting user names

You can change the User-Name attribute in each request by using *RewriteUsername* configuration parameters globally and in different clauses. This allows you to apply separate rewriting rules to the User-Name. In case there are several clauses with *RewriteUsername* parameters, rewriting rules for a request are executed in the following order:

- Globally when received by Radiator. This occurs prior to other rewrites.
- Within a *<Client>*, *<ServerRADSEC>* or *<ServerTACACSPLUS>* clause depending on how the request was received.
- When handled by a *<Handler>* or *<Realm>* depending on which clause is selected.
- When handled by an *<AuthBy GROUP>* clause. This occurs before any of the *<AuthBy>* clauses in the group are called.

The parameter is a Perl substitution regular expression that is applied to the User-Name attribute in the request. If you do not know how to write Perl substitution regexps, you should consult a Perl programmer. At Trace level 4, you can see the result of each separate rewrite for debugging purposes.

You can have any number of *RewriteUsername* parameters. The rewrites are applied to the username in the same order that they appear in the configuration file.

Here are some example of using this feature in a variety of circumstances:

- Strip the realm name from the username. This is handy if your user database contains only the usernames without the realm extension, for example, *fred* instead of *fred@yourdomain.com*.

```
# Strip realm
RewriteUsername s/^([^@]+).*/$1/
```

- Convert Microsoft or other style usernames from domain\user to the *user@realm* form that Radiator uses. Note that regexps need to quote some characters. For example @ needs to be quoted with backslash \:

```
# Convert a MSN realm/user into user@realm
RewriteUsername s/^(.*)\/(.*)/$2\@$1/
```

- Force all usernames to be lower case or upper case:

```
# Translate all uppercase to lowercase
RewriteUsername tr/A-Z/a-z/
```

- Remove any white spaces from the username:

```
RewriteUsername s/\s+//g
```

- Convert all *<user>@realm1* to *<user>@realm2*:

```
RewriteUsername s/^([^@]+)\@realm1/$1\@realm2/
```

- Change any *mikem* username to *fred*:

```
RewriteUsername s/^mikem\@/fred\@/
```

9. File formats

9.1. Dictionary file

The dictionary file defines easy-to-read names for the attributes and values used in RADIUS messages. It defines how RADIUS attribute numbers map to readable attribute names, and how RADIUS value numbers map to readable value names. The dictionary also defines the type of data that each attribute can hold.

The dictionary file is an ASCII text file. Each definition occupies one line. A hash mark # marks the beginning of a comment. Comment and blank lines are ignored.

9.1.1. ATTRIBUTE `attrname attrnum type [flags]`

This defines the name, RADIUS attribute number, and type for an attribute. Here is an example of attribute definition:

```
ATTRIBUTE Service-Type 6 integer
```

ATTRIBUTE is the keyword that says this is an attribute definition. Service-Type is the name of the attribute: the string that is used as the attribute name when printing the attribute and when setting attributes in the user database. 6 is the standard RADIUS attribute number for this attribute (see RFC 2865), and integer is the data type for this attribute. The supported data types are:

- *string*

  ASCII string of up to 253 bytes. Trailing NULs are stripped.

- *text*

  Similar to *string*

- *integer*

  32-bit unsigned value

- *signed-integer*

  32-bit signed value

- *integer8*

  8-bit unsigned value

- *integer16*

  16-bit unsigned value using network byte order

- *integer64*

  64-bit unsigned value using network byte order

- *date*

  Date as an integer number of seconds since 00:00:00 UTC Jan 1 1970

- *ipaddr*

  IP address in the form aaa.bbb.ccc.ddd, or a 4-byte binary string

- *ipaddrv6*

  IPv6 address in the form 2001:db8:148:100::31

- *ipaddrv4v6*

  4 or 16 octets long IPv4 or IPv6 (respectively) address in network byte order

- *binary*

  Binary data

- *abinary*

  Ascend filter, using the special Ascend filter definition syntax. Radiator is very strict about the syntax. You must follow the filter definition syntax exactly.

- *hexadecimal*

  Binary data formatted as hexadecimal

- *boolean*

  Required only by some Nortel/Aptis CVX vendor-specific attributes. A single byte attribute. Values of 0 or 1 are permitted.

- *tagged-integer*

See Section 7.1.24. Tagged Tunnel attributes and other tagged attributes on page 473.

- *tagged-string*

  See Section 7.1.24. Tagged Tunnel attributes and other tagged attributes on page 473.

- *ipv4prefix*

  IPv4 prefix in the form 192.168.1.0/24

- *ipv6prefix*

  IPv6 prefix in the form 2001:db8:148:100::/64

- *ifid*

  IPv6 interface identifier in the form aaaa:bbbb:cccc:dddd

- *tlv*

  Encapsulation attribute that contains one or several attributes

- *custom*

  See Section 9.1.10. Using VSA framework for customised attributes on page 490 for more about custom attribute VSA framework.

If you redefine an ATTRIBUTE by defining a new name for an previously defined attribute number, the new definition replaces the old one. The first is a synonym for the second when used in a reply.

attrnum may be in decimal, hex (prefixed by '0x') or octal (prefixed by 0).

ATTRIBUTE also supports optional flags to control whether the attribute is tagged or requires encryption like this:

```
ATTRIBUTE Tunnel-Password 69 string has_tag,encrypt=2
```

The permitted flags are:

- *has_tag*

  Specifies that the encoded attribute is prefixed a tag octet. The value of the tag can be specified in an attribute value with a leading tag number and a colon.

- *encrypt=n* (n = 1, 2 or 3)

  Specified that the attribute is to encrypted with the specified algorithm. The following algorithms are supported:

  1. RADIUS User-Password encryption
  2. The SALT algorithm as described by RFC 2548
  3. Symmetric encoding and decoding as required for Ascend-Send-Secret

9.1.2. VALUE `attrname valuename value`

This defines a symbolic name for an integer type attribute. When setting the value of an integer attribute in a check or reply item, you can use the symbolic name instead of the raw integer.

```
VALUE Service-Type Login-User 1
```

VALUE is a keyword that says this is a value definition. Service-Type means that this is a definition of a value for the Service-Type attribute (which should be of type integer). Login-User is the symbolic name for the value, and 1 is the value that Login-User translates to. *value* may be in decimal or hex (prefixed by '0x').

If you redefine a VALUE by defining a new name for a previously defined value number, the new definition will silently replace the old one. The first will be a synonym for the second when used in a reply.

9.1.3. VENDORATTR `vendornum attrname attrnum type [flags]`

This defines a vendor-specific attribute. RADIUS defines a special attribute number 26 that can be used to hold any vendor defined data type. This allows NAS vendors to add their own NAS-specific codes.

```
VENDORATTR 9 cisco-avpair 1 string
```

VENDORATTR is a keyword that says this is a vendor-specific attribute definition. 9 is the SMI Network Management Private Enterprise Code of the vendor (Cisco in this example). cisco-avpair is the symbolic name for the attribute, and string is the data type. The same data types as for ATTRIBUTE are supported. Radiator supports both hex and decimal attribute numbers in VENDORATTR.

After an integer VENDORATTR is defined, you can have VALUE definitions in the same format as for other ATTRIBUTES.

If you redefine a VENDORATTR by defining a new name for an previously defined vendor attribute number, the new definition will silently replace the old one.

`attrnum` may be in decimal, hex (prefixed by ‘0x’) or octal (prefixed by 0).

VENDORATTR also supports the same optional flags. For more information, see Section 9.1.1. ATTRIBUTE `attrname attrnum type [flags]` on page 485.

9.1.4. VENDOR `vendorname vendornumber`

Specifies that the decimal number vendornumber can be used to refer to the Vendor given by vendorname.

9.1.5. BEGIN-VENDOR `vendorname [parent=attributename]`

Specifies that all subsequent ATTRIBUTE keywords, until the next `END-VENDOR` are to be interpreted as `VENDORATTR` definitions for the named vendor.

The optional `parent=attributename` syntax is needed when Vendor-Specific attributes from extended space, as defined by RFC 6929, are used. The following example shows how to define a Vendor-Specific Attribute named `OSC-Uid` and an Extended-Vendor-Specific Attribute named `OSC-Extended-Example`. These both use *`Vendor-Id`* `9048` and *`Vendor-Type`* `1`. The full "dotted number" notation that uniquely identifies `OSC-Extended-Example` is `241.26.9048.1`. The unique identifier of `OSC-Uid` Vendor-Specific Attribute is `26.9048.1`.

```
VENDOR OSC 9048

VENDORATTR 9048 OSC-Uid 1 string

BEGIN-VENDOR OSC parent=Extended-Vendor-Specific-1

ATTRIBUTE OSC-Extended-Example 1 string

END-VENDOR
```

Tip

For backwards compatibility, `format=attributename` is also allowed. The exact syntax used by different implementations is still evolving.

9.1.6. END-VENDOR

Terminates the previous BEGIN-VENDOR.

9.1.7. include `filename`

Includes the dictionary file specified by filename. Special characters are supported.

9.1.8. $INCLUDE `filename`

FreeRRADIUS style include command, includes the dictionary file specified by filename. Relative filenames are relative to the directory of the current dictionary. Special characters are not supported.

9.1.9. Available dictionaries

Radiator comes with a single main dictionary, plus an optional add-on dictionaries.

- `dictionary`

  The is the normal and default one. It merges attribute definitions from several sources, and should be satisfactory for most configurations.

- `dictionary.ascend`

  This is an add-on dictionary that includes definitions for some old-fashioned Ascend non-vendor-specific attributes that would normally conflict with newer attributes. If you need these old attributes, you can specify it as a second dictionary.

- `dictionary.*` and `goodies/dictionary.*`

  These dictionary files have more attributes for special cases. See the individual files for more details.

Under very unusual circumstances, you may wish to modify your dictionary. In any case, your dictionary must have entries for at least the following attributes, which are referred to by name in the *`radiusd`* code.

- User-Name
- User-Password
- Acct-Delay-Time
- Any other attributes that are required by your check and reply item configuration.

Wherever possible, use the default dictionary: it will work with the vast majority of cases. If you are operating with NASs from only one vendor, choose the default dictionary, or dictionary for that vendor. If you are operating in a mixed environment, use the default dictionary. If that does not work for you, try concatenating the dictionaries for the vendors you are using into one big dictionary or use *`DictionaryFile`* parameter to load the dictionaries as separate files.

If you need to add or modify attributes or other dictionary entries, we recommend adding them to your own site-specific dictionary, and then add that dictionary to the list of dictionaries given by the *`DictionaryFile`* parameter in your configuration file. See Section 3.7.16. DictionaryFile on page 35.

Whichever dictionary you choose to use, you should place it in the directory where *radiusd* expects to find it before starting *radiusd*. You should also be careful to specify the same dictionary to *radpwtst* with the -dictionary argument if you use it for testing.

9.1.10. Using VSA framework for customised attributes

Radiator supports using customised attributes. Currently there are some ready, customised modules available in Radiator Service Provider Module. For more information, see Radiator Service Provider Module [https://files.radiatorsoftware.com/carrier/radiator-carrier-ref.pdf].

9.2. Flat file user database

Flat file user databases are used to list all the legitimate users for the <AuthBy FILE> module. You can also use flat file user databases as input to the *builddbm* and *buildsql* utility to create a DBM user database. See the `users` file in the Radiator distribution for an example.

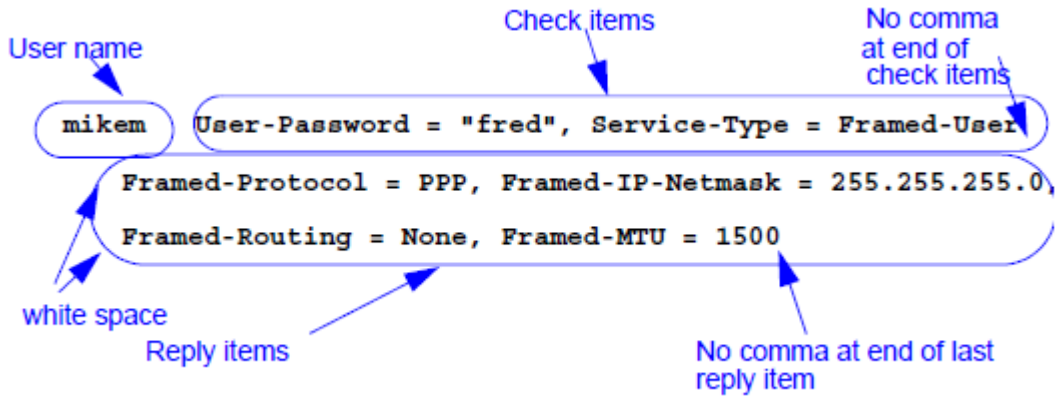
Flat file user databases are ASCII text files containing zero or more user definitions. Lines beginning with '#' are ignored. Each user definition is one or more lines. The first line must start with the user name in the first column. Subsequent lines for the user must begin with white space.

User names are quoted with double quotes (") if they contain white space. The user name must be followed by white space, followed by zero or more check items. Each check item is in the form *Attribute = Value*, and it defines a RADIUS attribute that is checked in Access-Requests before the user is authenticated. Multiple check items are separated by commas. There must be no comma after the last check item in the line. Values may optionally be surrounded by double quotes, which are ignored. For more information about check items, see Section 7. Check and reply items on page 462.

The second and subsequent lines are 'reply' items. Each line must commence with white space. Each reply item is in the form "Attribute = Value", and it defines a RADIUS attribute that are returned to the NAS if the authentication succeeds. Such reply items are generally used to configure the NAS for the user. Each reply item must have a trailing comma (',') except the last item on the last line. Values may optionally be surrounded by double quotes, which are ignored. Reply item values support special formatting characters, see Section 3.3. Special formatters on page 21.

The following figure shows an example, if the user *mikem* is granted access, the modem is configured for Framed-Protocol of PPP, and IP netmask of 255.255.255.0, Framed-Routing of None and a Framed-MTU of 1500.

Figure 5. Typical user entry in a flat file user database



9.3. DBM user database

DBM user database file is in similar format to the DBM files supported by Merit and other RADIUS servers. Entries are hashed by user name, which is unique. Each user entry consists of two strings separated by newline characters. The first line is a list of comma separated Check items in the form “Attribute = Value”. The second line is a list of comma separated Reply items in the form “Attribute = Value”. During authentication, the check and reply items are used in exactly the same way as for a Flat file user database. For more information, see Section 7. Check and reply items on page 462.

Radiator will choose the best format of DBM file available to you, depending on which DBM modules are installed on your machine. (Hint: You can force it to choose a particular format by using the DBType parameter, and by using the -t flag with *builddbm*).

You can convert a flat file user database directly into a DBM user database with *builddbm* utility, and you can also use *builddbm* to print the attributes for a particular user. You can convert a DBM user database into an SQL database with the *buildsql* utility.

A DBM database can have multiple DEFAULT users. During authentication, if no user name is found in the DBM database, the DEFAULT users will be tried in the order that they appeared in the input file, until a match is found. Technical Note: when the DBM file is built from a flat file, the first DEFAULT user encountered is added to the DBM file with the name DEFAULT. The seconds and subsequent DEFAULT entries are added as DEFAULT1, DEFAULT2 etc. Therefore the uniqueness and order of DEFAULT users in the database is guaranteed.

9.4. Unix password file

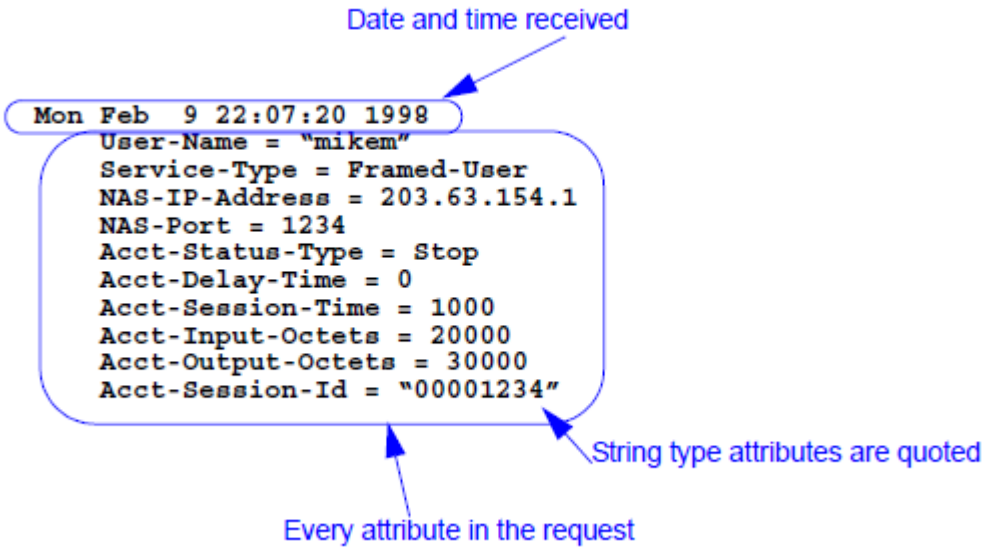
A Unix password file as understood by the AuthBy UNIX module is an ASCII file consisting of one line per user. There are at least 2 colon (:) separated fields per line. The first field is the user name, and the second is the crypt(3) encrypted password. The third and subsequent fields (if they are present) are ignored.

It will be recognised that this description fits the standard Unix password file format, and Radiator will work with `/etc/password` on Unix implementations that do not use a password shadow file. It will work with `/etc/shadow` on Unix implementations that do use a password shadow file (Radiator will need to run as root to get read access to `/etc/shadow`).

9.5. Accounting log file

The accounting log file is used to store the details of every Accounting-Request received for a Realm if the AcctLogFileName parameter is defined for the Realm. It is an ASCII text file with each entry occupying one or more lines and followed by a blank line. The log file format is identical to the format used by Livingston and other RADIUS servers. The first line gives the date and time the request was received by the server. The subsequent lines give the values for every attribute in the request. Every Accounting-Request, regardless of Acct-Status-Type is stored in the log file.

Figure 6. Typical accounting log file entry



9.6. Password log file

The Password log file is useful for your help desk to diagnose user's login problems. It is only written if you define PasswordLogFileName for a Realm. It is an ASCII text file containing the results of password checks, one check per line. Each line contains 5 colon (:) separated fields:

```
time:seconds:user:submitted_password:correct_password:result
```

The time is the full date time string, seconds is the number of seconds since January 1 1970 (Unix epoch). Result is the word "PASS" or "FAIL". For example:

```
Mon Jun 29 12:24:21 1999:899087061:mikem:fredd:fred:FAIL
Mon Jun 29 12:24:38 1999:899087078:mikem:fred:fred:PASS
Mon Jun 29 12:38:53 1999:899087933:mikem:fred:fred:PASS
```

The file is opened, written and closed for each entry, so it can be safely rotated at any time.

9.7. Portlist file

A port list file contains a list of permitted NAS address/port combinations that are permitted for a user or group of users. It is used by the NAS-Address-Port-List check item. It is an ASCII file, with two white space separated fields. The first field is the NAS address (either as a DNS name or IP address). The second field contains the lower and upper permitted port numbers in the format “lowport-highport”. Blank lines and lines starting with hash (‘#’) are ignored. You can have any number of entries for each NAS, and any number of NASs.

Note

iPASS roaming users do not get their real NAS Address sent to Radiator. Radiator considers these to be address 0.0.0.0 for the purposes of NAS-Address- Port-List.

```
# Users can log into ports 1-10 and 21-30 inclusive
# on 10.1.1.1 or into ports 100 to 115 inclusive on
# 10.1.1.2, or into ports 16 to 20 inclusive on
```

```
# mynas.domain.com
10.1.1.1 1-10
10.1.1.1 21-30
10.1.1.2 100-115
mynas.domain.com 16-20
# For iPASS roaming:
0.0.0.0 0-1000
```

9.8. Group membership file

This file contains a list of groups and their members. *<AuthBy FILE>* uses these groups and the group file is defined with *GroupFilename* parameter. For more information, see Section 3.35.2. GroupFilename on page 182.

The format of group membership file has first the name of the group and then a comma-separated list of users that belong to that group. One user can belong to one or several groups.

Here is an example of a group membership file:

```
Admin-group username1

WiFiClient-group username1, username2, username3

Readonly-group username4, username5
```

10. Configuring Radiator with GUI

If your Radiator configuration file has a <ServerHTTP> clause configured, you can connect to Radiator with any standard web browser and use it to inspect, monitor, configure, restart, reconfigure, save and check statistics of your Radiator server from anywhere in your network. For more information, see Section 3.122. <ServerHTTP> on page 424.

This makes configuration and maintenance of your Radiator easier for staff that are not familiar with more traditional command-line tools and configuration file editing.

Tip

A simple configuration file that enables <ServerHTTP>, allowing you to get started with using the web-based configuration GUI is in `goodies/serverhttp.cfg` in your distribution.

Tip

The default port number that <ServerHTTP> uses is 9048. Use the following URL:

```
http://localhost:9048
```

If TLS or SSL is enabled in your <ServerHTTP> clause, use the following URL:

```
https://localhost:9048
```

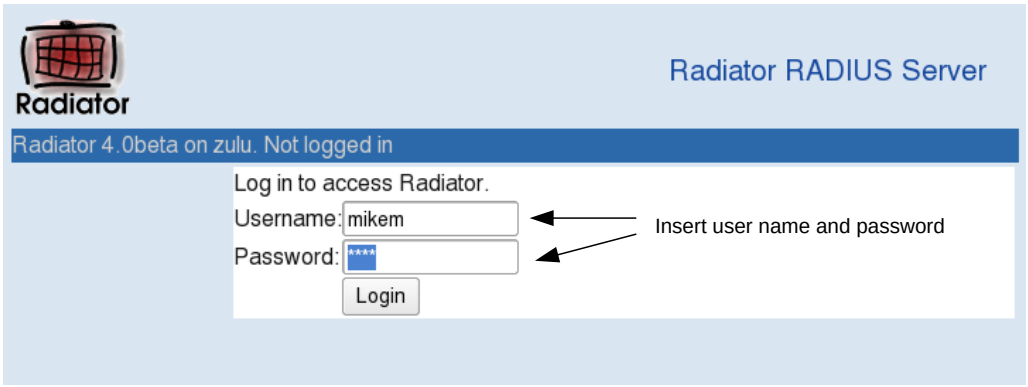
10.1. Login page

When you first connect to the HTTP server port, you will be presented with a login page. Log in with the user name and password configured into the Radiator configuration file (the default shipped in the sample configuration file is user name ‘mikem’ and password is ‘fred’).

Tip

If the HTTP server has been configured not to require a user name and password, you will be logged in immediately without seeing the Login page.

Figure 7. Login page



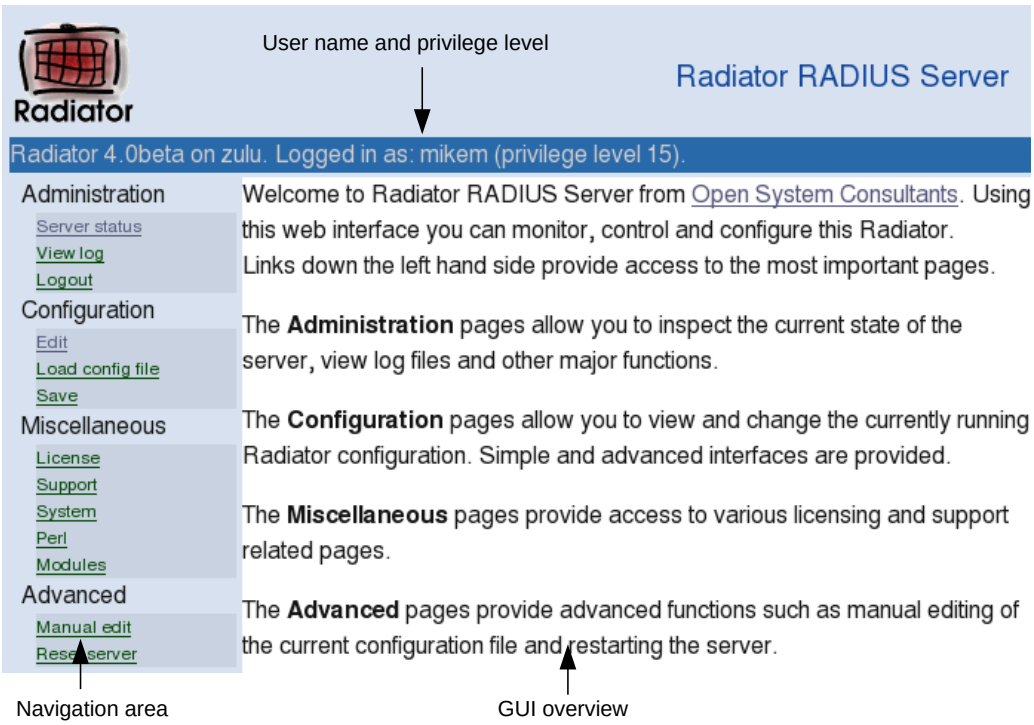
After a successful login, Home Page is opened. For more information, see Section 10.2. Home Page on page 494.

10.2. Home Page

The Home page displays basic information about the GUI and provides a Navigation Area, from which you can access all the functions available to you.

The Navigation area contains links to each type of page you are authorised to see. The list of links depends on your Privilege Level. For more information, see Section 3.122. <ServerHTTP> on page 424.

Figure 8. Home Page

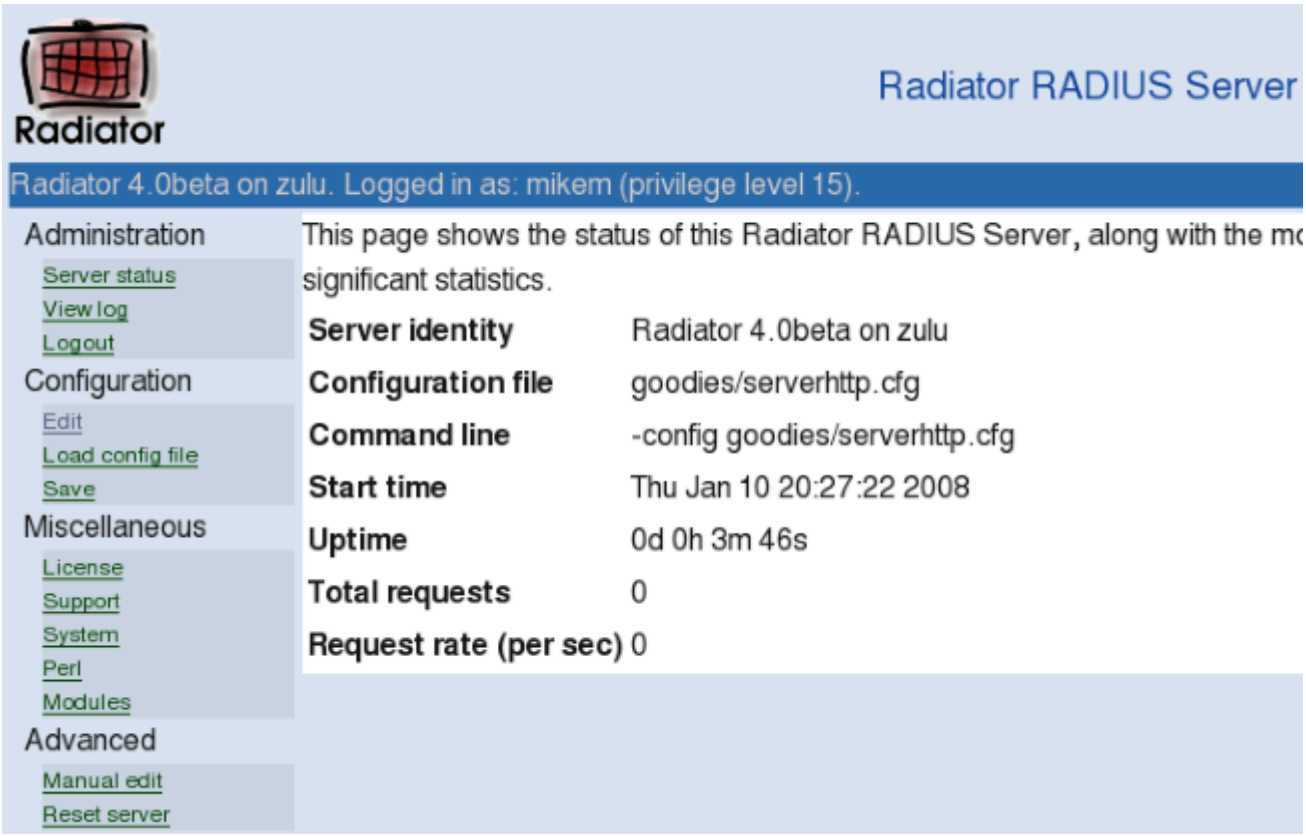


10.3. Administration

10.3.1. Server status

The Server Status page shows details about this instance of Radiator, including the command line used to start it, the name of the configuration file, how long the server has been running for and how many requests it has processed.

Figure 9. Server status page

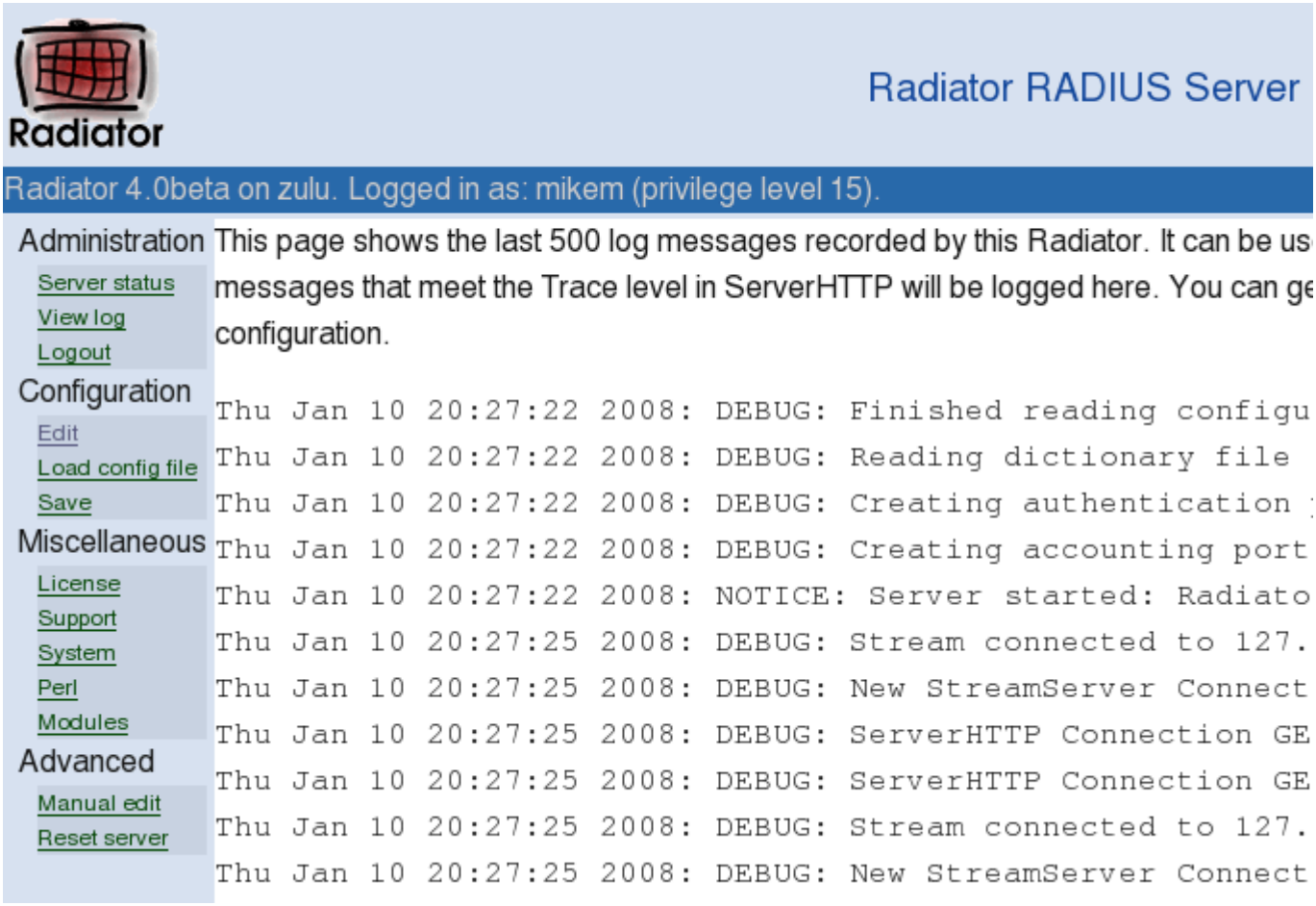


10.3.2. View log

The GUI log collects and displays the last LogMaxLines of log messages at or above the current Trace level. To renew the displayed list, click Refresh on your browser.

To change the number or severity of collected log messages, click on Edit and see the ServerConfig page. Click on Show Advanced Options. Click on Server ServerHTTP and see the ServerHTTP configuration page. Adjust Trace on this page and LogMax-Lines on the Advanced Options page.

Figure 10. View log Page



10.3.3. Logout

The logout link logs you out of the GUI. A new Login page will be displayed.

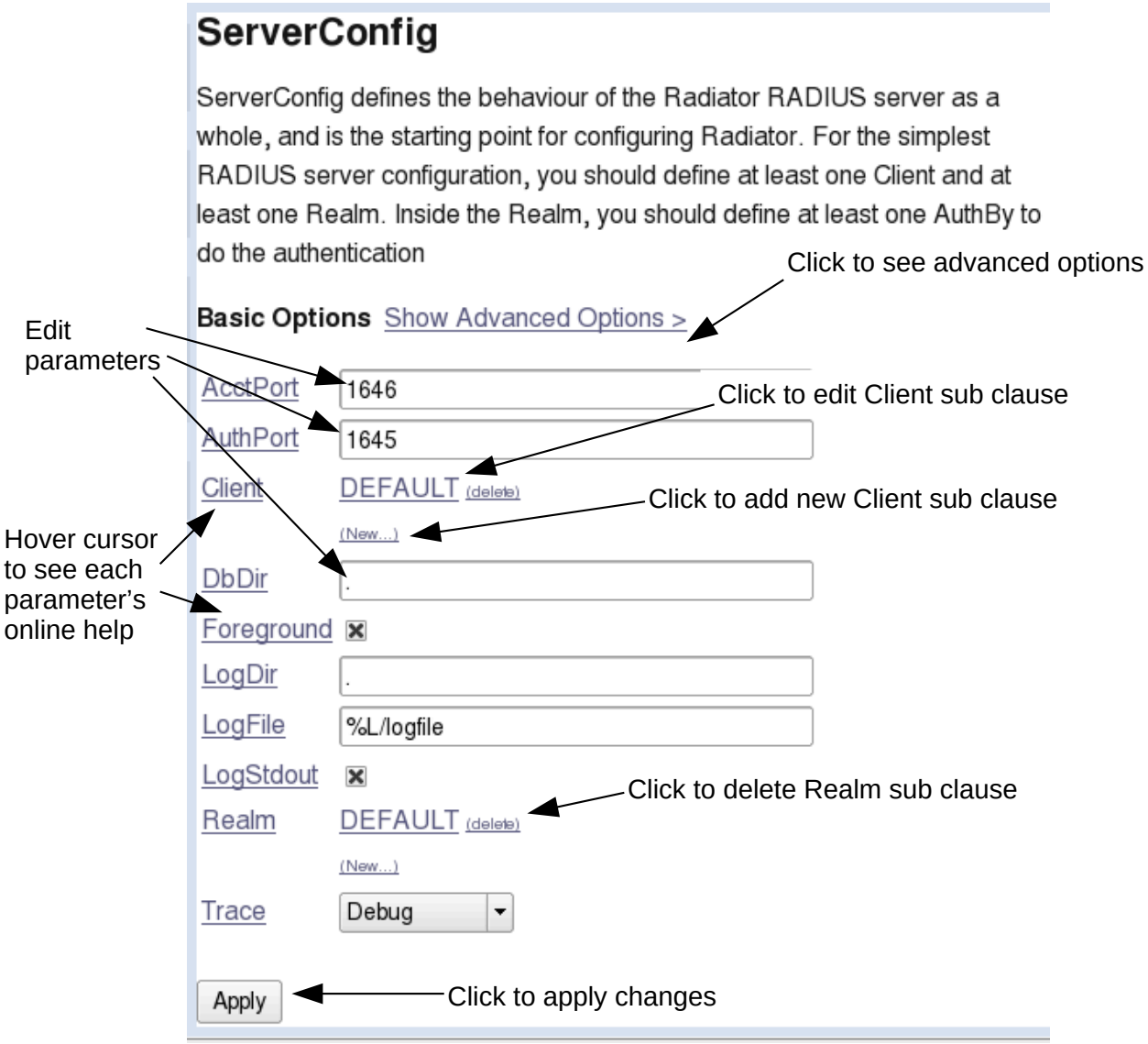
10.4. Configuration

10.4.1. Edit

The Edit page displays configuration details for a single configurable Radiator object. Each Edit page corresponds to a Radiator configuration clause. By clicking on appropriate links you can drill down into the hierarchy of configuration clauses and examine and change the configuration of each clause.

The first page you see when you click on the Edit link is the ServerConfig page, which is the main (top-level) configuration page. All the Radiator configuration clauses and their sub-clauses are accessed starting at the ServerConfig page and drilling down into subclauses.

Figure 11. Edit Page



For a given configuration object, the GUI displays up to four different pages, one for each level of configuration detail:

- Basic Options

  Displays the most basic configuration options that must be configured in order for the object to work at all.

- Advanced Options

  Shows more advanced options that can fine tune the basic operation of the object.

- Very Advanced Options

  Shows options that are rarely used and which are only required under unusual circumstances.

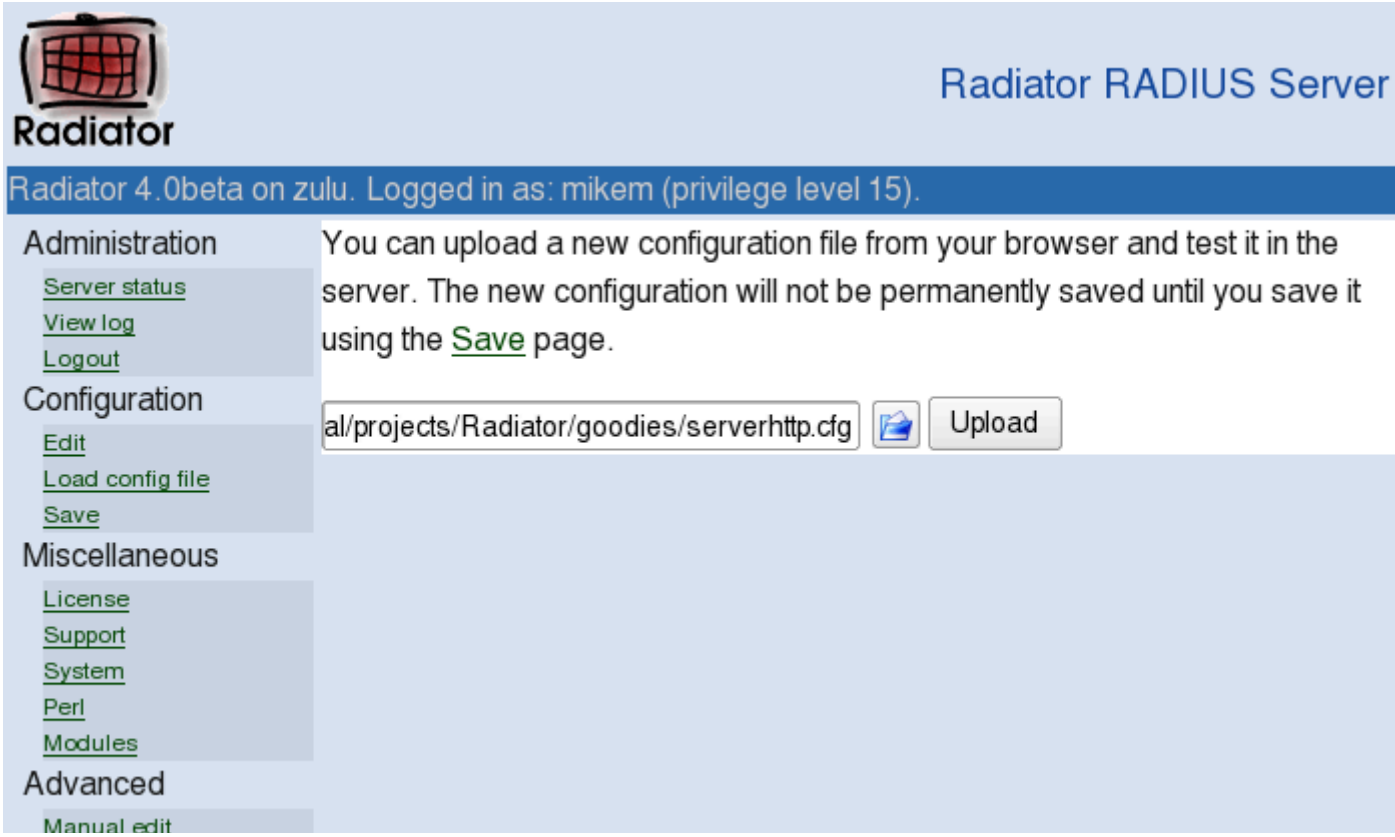
- Deprecated Options

Shows options that should never be used in a new installation. They are only provided for compatibility with old deprecated (discouraged) options.

10.4.2. Load config file

This page allows you to upload an entire new configuration file into Radiator.

Figure 12. Load config Page



After uploading, the new configuration will take effect. However the new configuration will not be saved until you click on Save.

Tip

If the new configuration does not include a ServerHTTP clause, you will not be able to reconnect to the server with the browser.

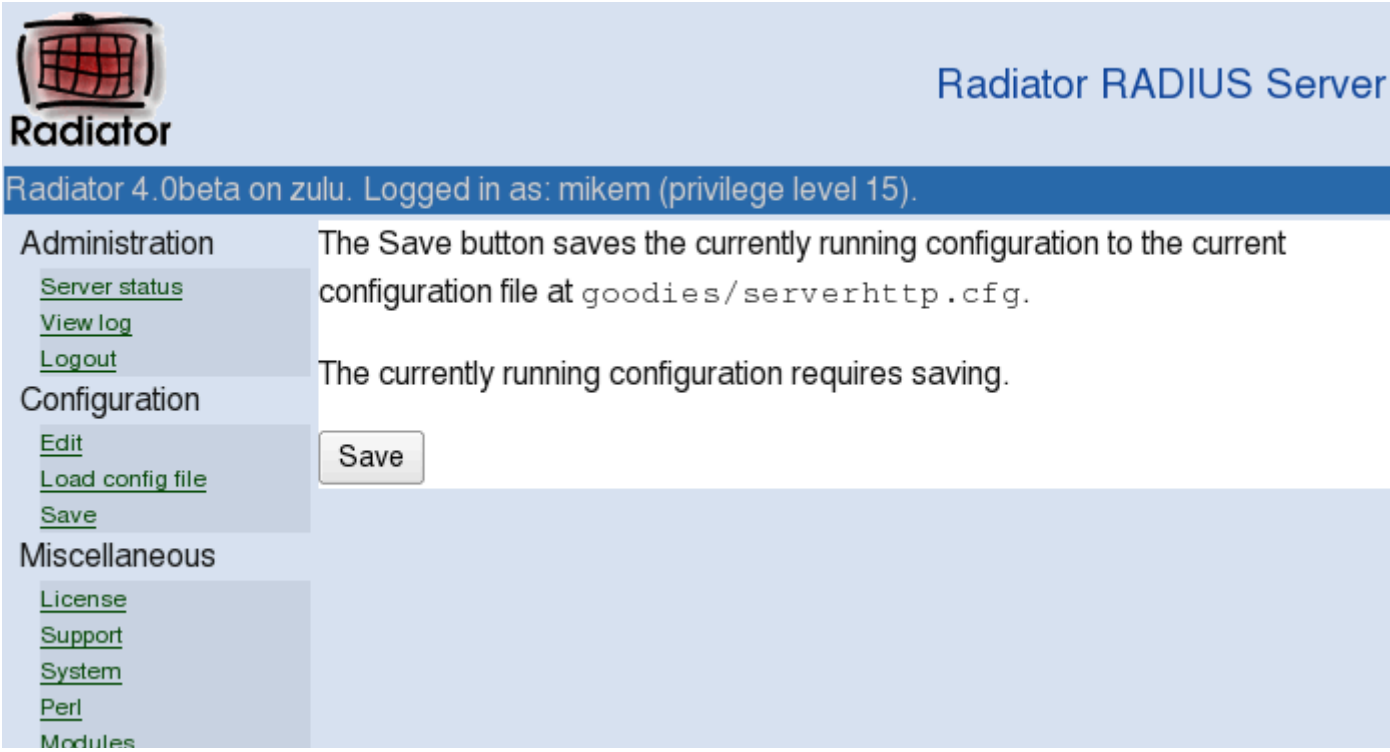
Tip

There are many sample configuration files included in the `goodies/` directory of your Radiator distribution.

10.4.3. Save

This page allows you to save the currently running Radiator configuration to the configuration file specified on the Radiator command line. The original configuration file will be saved and renamed with a '`.bak`' extension.

Figure 13. Save Page



CAUTION

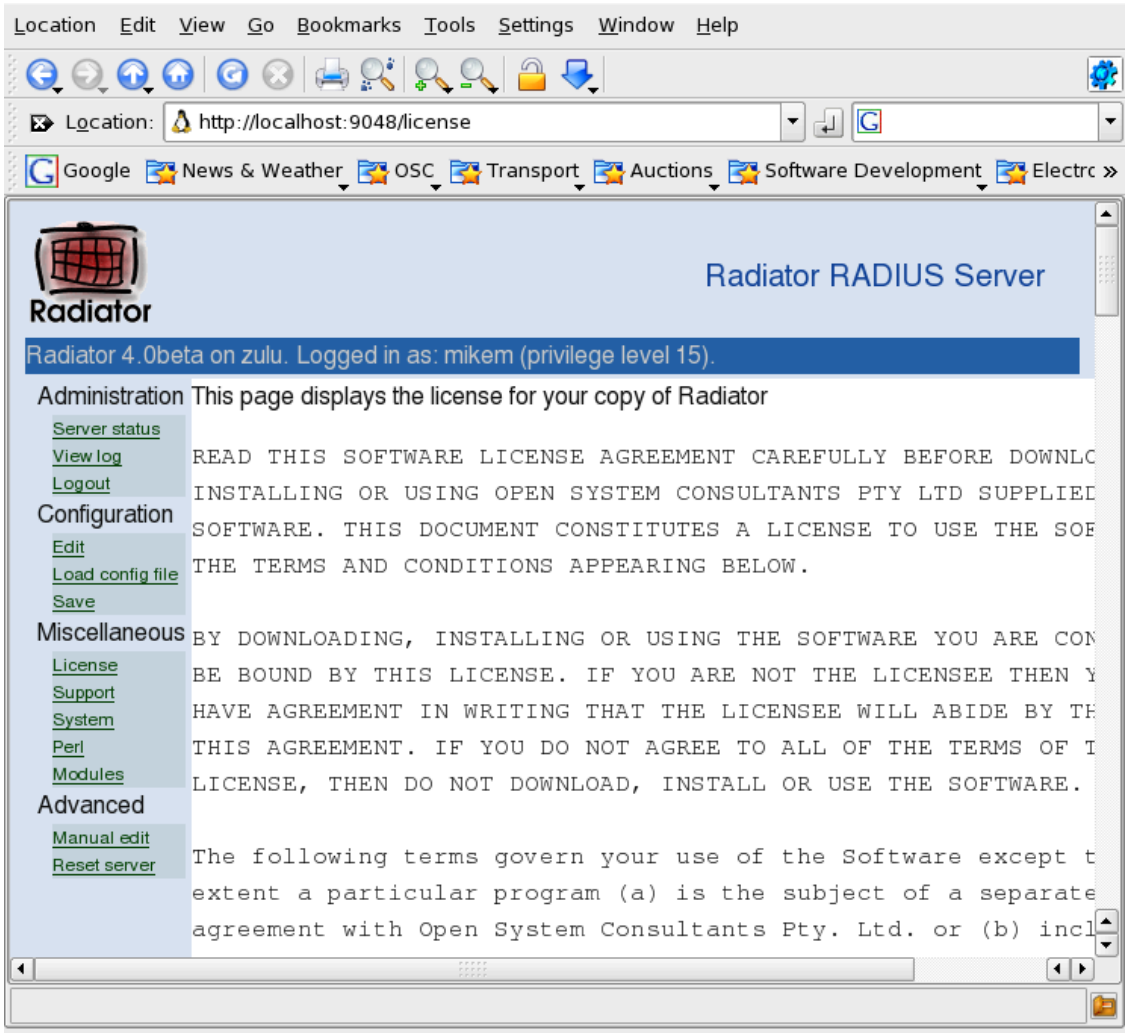
The newly saved configuration file will not include any formatting or comments that may have been present in the original configuration file.

10.5. Miscellaneous

10.5.1. Licence

This page displays the software license that applies to your copy of Radiator.

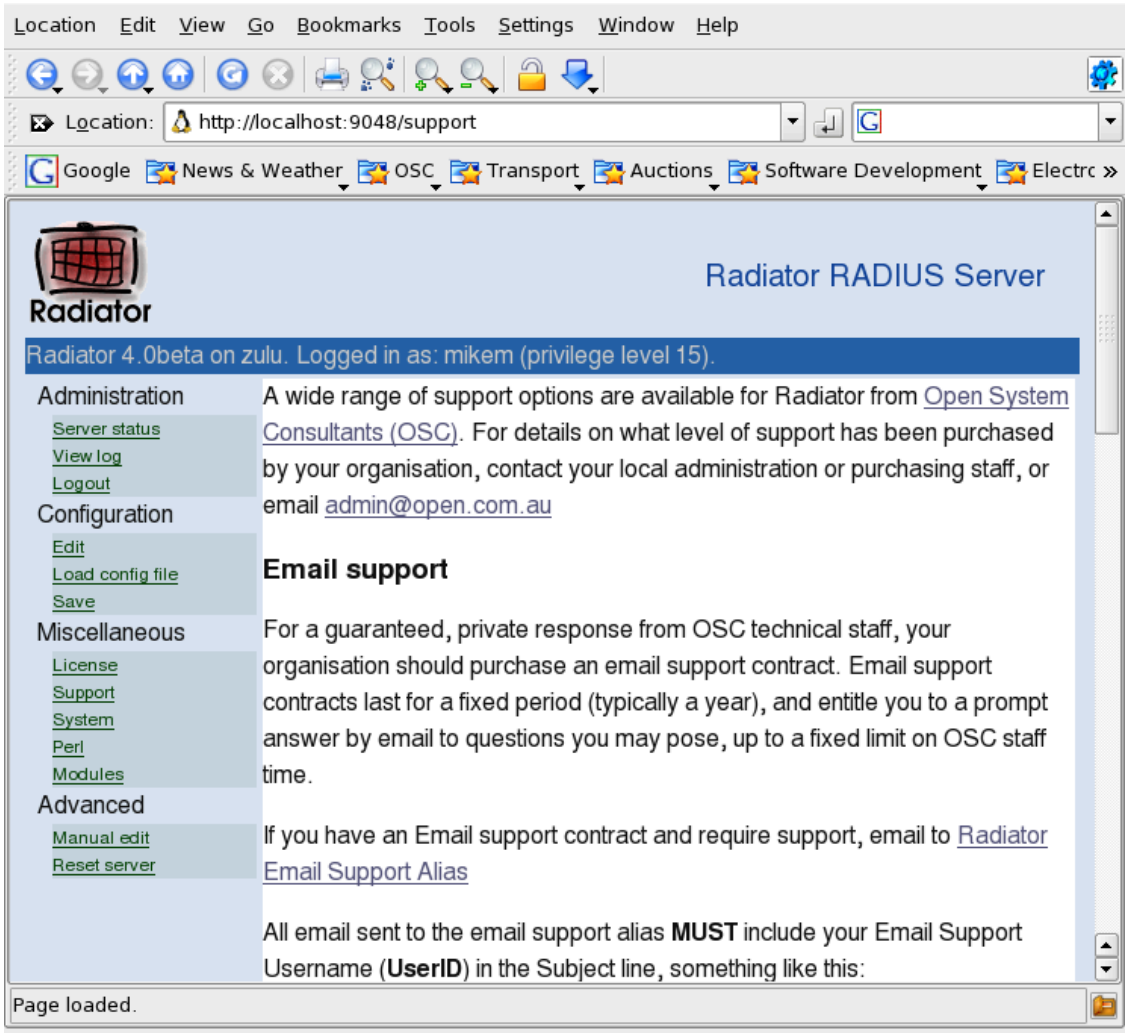
Figure 14. License Page



10.5.2. Support

This page displays useful information about what levels of support are available for Radiator and how to access them.

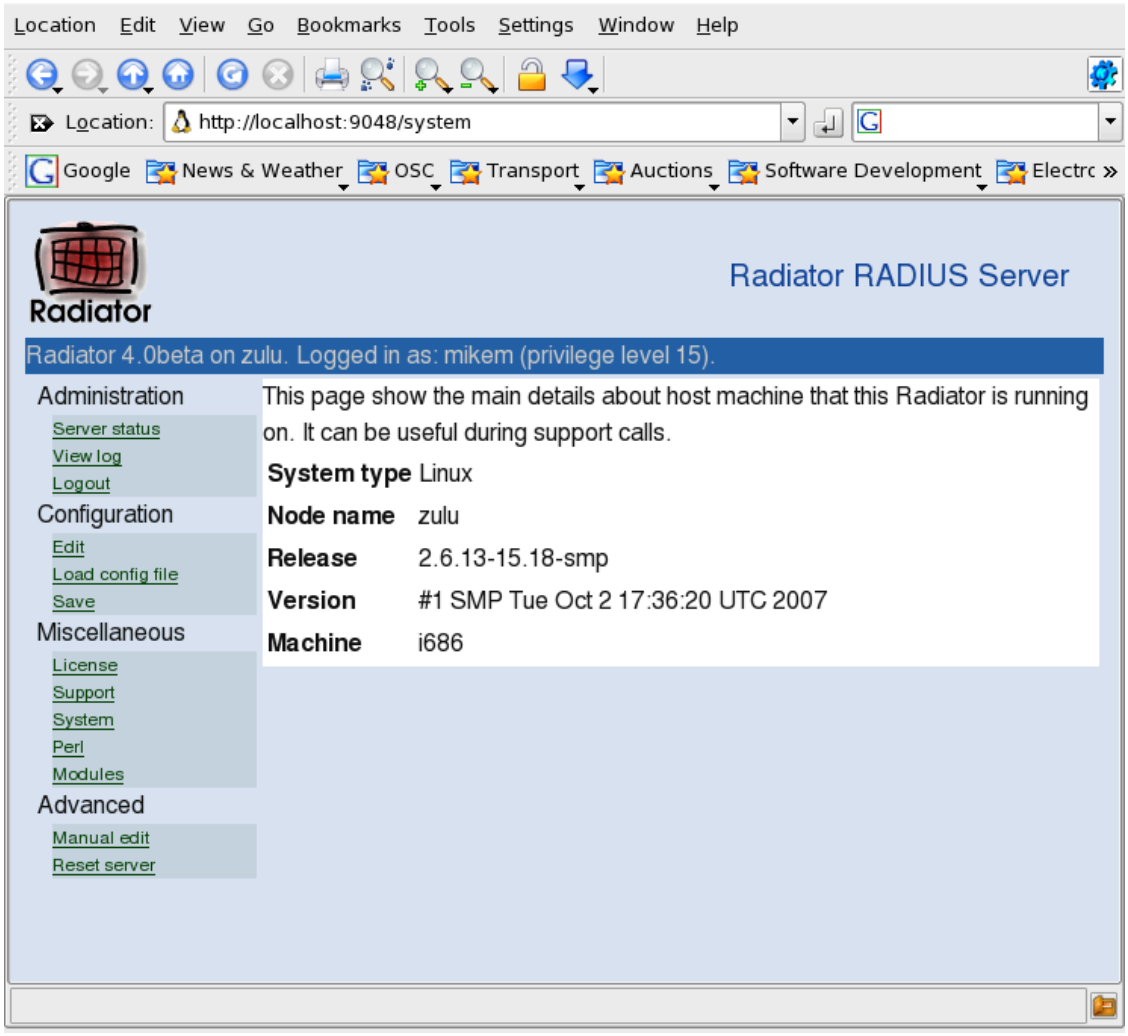
Figure 15. Support Page



10.5.3. System

This page shows information about the machine that this instance of Radiator is running on.

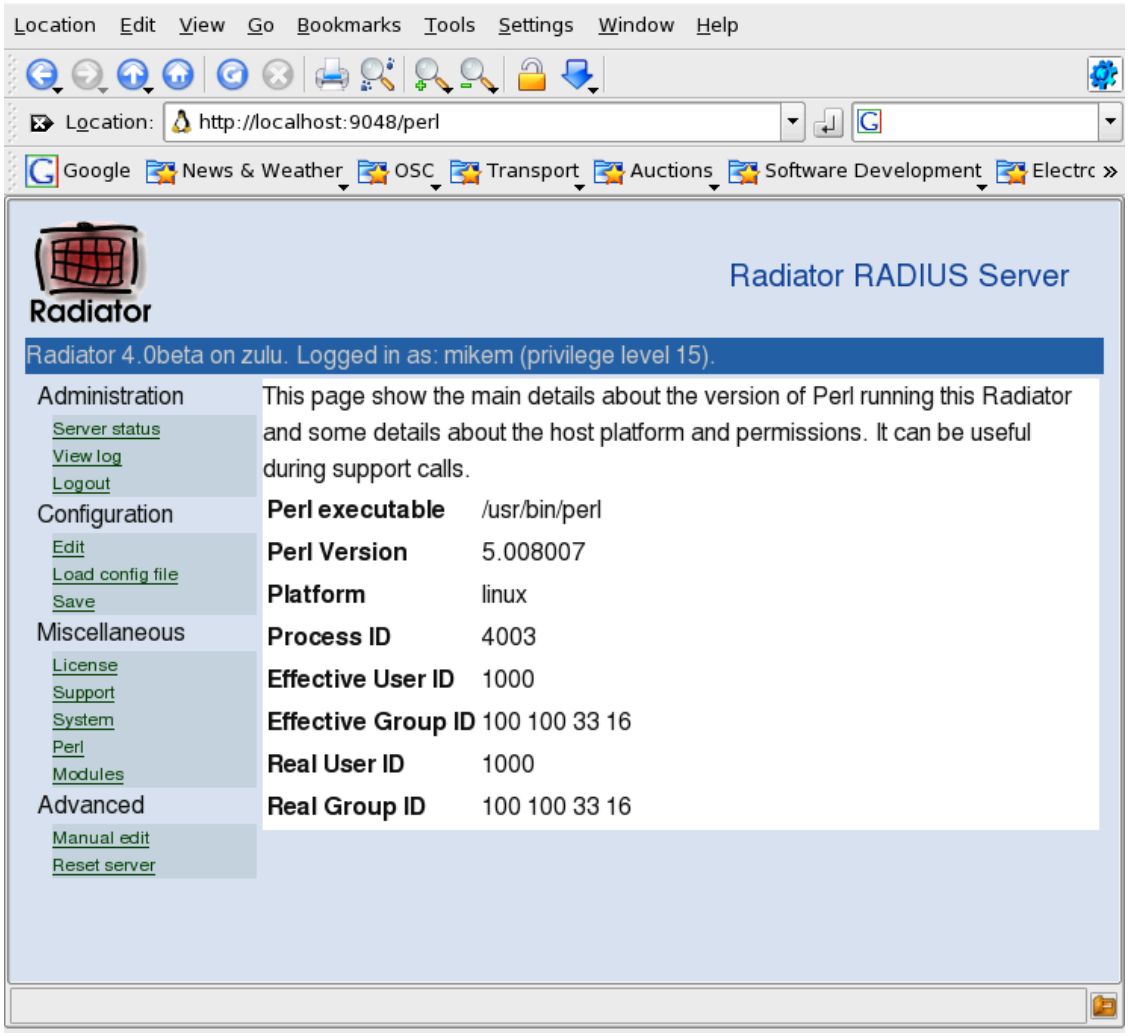
Figure 16. System Page



10.5.4. Perl

This page shows information about the version of the Perl interpreter that Radiator is using on this machine.

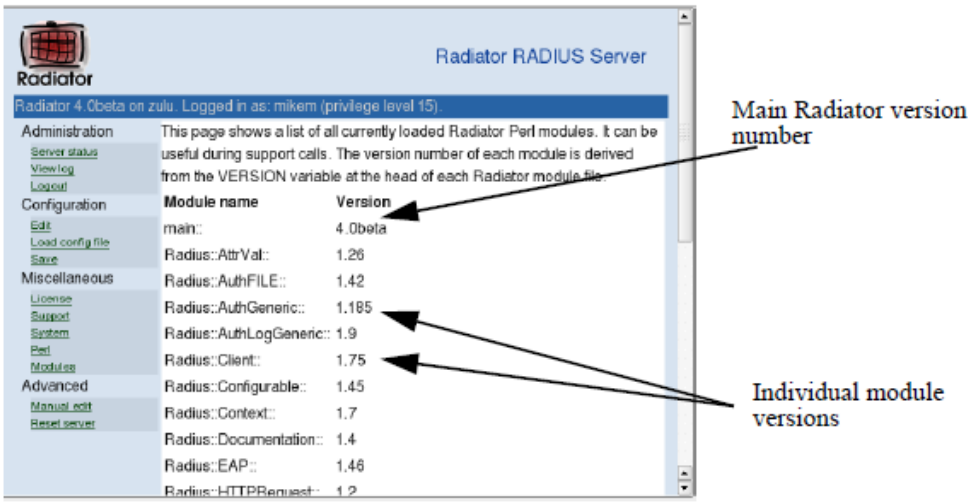
Figure 17. Perl Page



10.5.5. Modules

This page lists all the Radiator Perl modules that are currently loaded into Radiator, and the version number of each one. Also shown is the main Radiator version number.

Figure 18. Modules Page



Tip

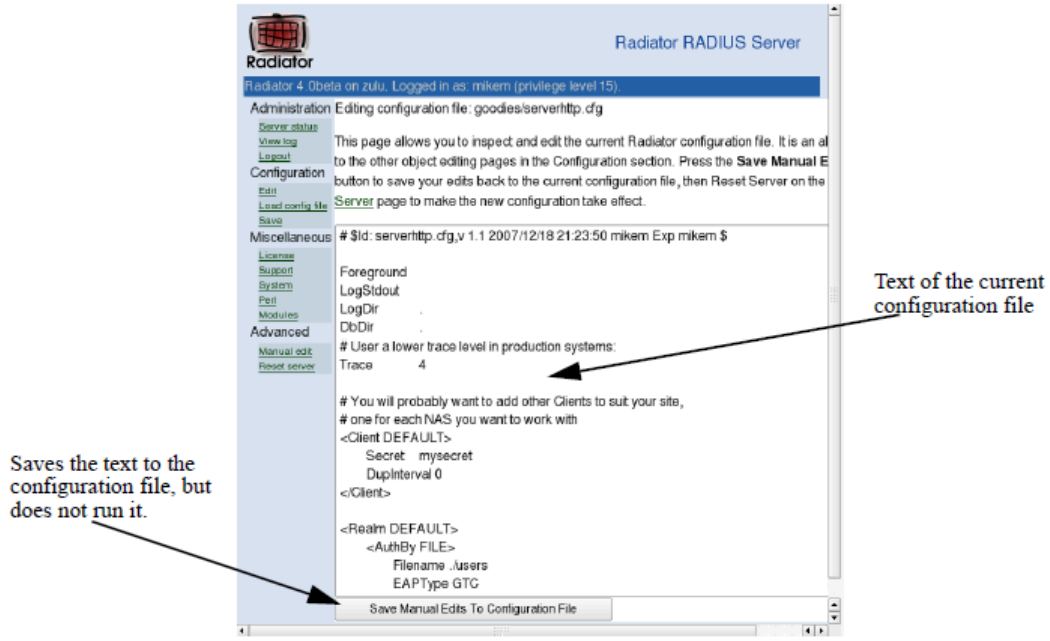
The list of modules you see will depend on the precise configuration your Radiator is running, since many Radiator modules are only loaded when required.

10.6. Advanced

10.6.1. Manual edit

This page allows direct manual editing of the configuration file without having to start a text editor on the host machine. In order to use this interface, you need to be familiar with Radiator configuration file format and syntax. For more information, see Section 3. Configuring Radiator on page 17. The original configuration file will be saved and renamed with a '`.bak`' extension.

Figure 19. Manual edit Page

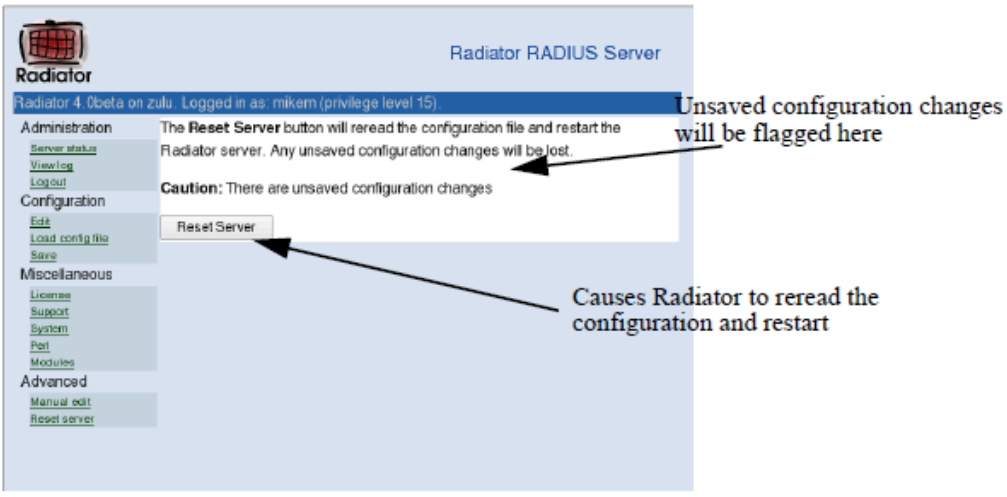


Saving a new configuration with the Save button saves the edited text to the configuration file (after making a backup of the original), but Radiator will not run with the new configuration until it is reset with the Reset server page. For more information, see Section 10.6.2. Reset server on page 506.

10.6.2. Reset server

This page allows you to Reset Radiator. Resetting causes Radiator to reread the configuration file and restart. Any unsaved configuration changes in the currently running configuration will be lost. If there are any unsaved configuration changes you will be informed of this.

Figure 20. Reset server Page



11. Using Gossip framework

Radiator's Gossip framework allows Radiator instances to share information and event notifications. The instances can be part of server farm, completely separate processes running on the same or different hosts, or any combination of thereof.

Radiator's Gossip framework supports the following backends:

- GossipRedis clause is used to configure a Redis-based Gossip framework instance. For more information, see Section 3.134. <GossipRedis> on page 439.
- GossipUDP clause is used to configure a UDP-based Gossip framework instance. For more information, see Section 3.135. <GossipUDP> on page 441.

Currently the following Radiator modules support Gossip:

- *<AuthBy RADIUS>* and its subclasses can use GossipRedis for communicating next hop host unreachability and reachability information with Gossip messages. Multiple Radiator instances, in both stand-alone and FarmSize configuration, can share proxy status information. This allows, for example, just one member to run Status-Server queries when FarmSize configuration parameter is enabled.
- *<AuthBy RADSEC>* support is similar to *<AuthBy RADIUS>*.
- Radius Clients can use GossipRedis for duplicate cache. When the global configuration parameter *DupCache* is set to **global**, GossipRedis is used for RADIUS duplicate cache.
- StatsLog REDIS uses GossipRedis for storing statistics in Redis.

Gossip message format supports optional header for TTL, payload encryption, and future extensions.

More Radiator modules will be added and upgraded to use the Gossip framework in the future. You can also create customised modules for your requirements. *Radius::Gossip* module provides the common API for all Gossip implementations such as GossipRedis and GossipUDP.

12. Adding custom AuthBy modules

Radiator provides an easy way of plugging in and integrating additional custom AuthBy modules. This allows you to use Radiator to authenticate from and store accounting information to other databases and storage types not currently supported by Radiator. You will need to be a competent Perl programmer in order to implement custom AuthBy modules.

Custom modules are usually written in Perl. They are automatically loaded when a matching <AuthBy ...> clause is read in the configuration file. Details on how to load, configure and implement a custom AuthBy module are described in this section.

There is a simple AuthBy module called AuthTEST.pm included in the RADIUS directory. It implements the <AuthBy TEST> clause, and would be a good starting point if you want to write your own custom AuthBy module. You should copy it to a new name and modify the copy to suit your needs.

12.1. Loading and configuring

When *radiusd* sees an <AuthBy XYZ> clause while parsing a Realm or Handler, it will load the file `Radius/AuthXYZ.pm` with a Perl require. `AuthXYZ.pm` is expected to be a Perl package that implements the class *Radius::AuthXYZ. Realm.pm* will then instantiate a new *Radius::AuthXYZ* by calling the constructor with *Radius::AuthXYZ->new($file)*. The constructor is expected to create an instance of Radius::AuthXYZ that can handle all the requests for that type of authentication.

The constructor is passed a filehandle *$file*, which is the configuration file being currently read. The constructor is expected to configure its instance from lines read from the configuration file up until it sees a

</AuthBy> line. This is made very easy by the *Radius::Configurable* class, which your AuthBy module should inherit from.

If the *Radius::AuthXYZ* constructor fails, it is expected to return **undef**.

12.2. Handling Requests

After construction and initialisation, your instance will be called upon to handle requests that are sent to it. For each request received, the *Handler.pm* module will call *($result, $reason) = Radius::AuthXYZ->handle_request($p, $rp)*, where *$p* is a reference to the *Radius::Radius* packet received from the client, and *$rp* is an empty reply packet, ready for you to fill. *Client.pm* and *Handler.pm* will filter out duplicate requests, requests from unknown clients and requests with bad authenticators, so your *handle_request* will only be called for new, good requests from known clients. The contents of the request will have been unpacked into the *Radius::Radius* instance passed in as *$p*, so you can immediately start examining attributes and doing things.

handle_request returns an array. The first element is a *result* code, and the second is an optional *reason* message.

The result code from *handle_request* will indicate whether *Handler.pm* should automatically send the reply to the original requester:

- If *handle_request* returns *$main::ACCEPT*, *Handler.pm* will send back *$rp* as an accept message appropriate to the type of request received (i.e. it will turn *$rp* into an Access-Accept if the original request was an Access-Request). In the case of Accounting-Request, this is the only result code that will cause a reply to be sent back.
- If *handle_request* returns *$main::REJECT*, *Handler.pm* will send back *$rp* as a reject message appropriate to the type of request received (i.e. it will turn *$rp* into an Access-Reject if the original request was an Access-Request). In this case the reason message should be supplied.
- If *handle_request* returns *$main::CHALLENGE*, *Handler.pm* will send back *$rp* as a Access-Challenge message.
- If *handle_request* returns *$main::IGNORE*, *Handler.pm* will not send any reply back to the originating client. You should only use this if the request is to be completely ignored, or if your module undertakes to send its own reply some time in the future. If the Handler or Realm has more than one AuthBy handler module specified, it will continue calling handlers in the order in which they were specified until one returns something other than *$main::IGNORE*. You can change this behaviour with AuthByPolicy, for more information, see Section 3.31.12. AuthByPolicy on page 154.

Your *handle_request* function may want to use utility functions in *Radius::Radius* (see *Radius.pm*) to examine attributes in the incoming request, and to construct replies. There are some convenience routines in *Client.pm* for packing and sending replies to the original requester, such as

```
$p->{Client}->replyTo($rp, $p);
```

If your handler cannot successfully handle the request, perhaps due to some unforeseen event, software failure, system unavailability etc., it is common to not reply at all to the original request. This will usually force the original NAS to retransmit to another server as a fallback. You can do this by returning *$main::IGNORE* from *handle_request*.

12.3. AuthGeneric

This is a generic authentication module superclass. It implements behaviour that will be required in most authentication modules, and therefore most modules should inherit from it. Most simple authentication can be handled merely by overriding the *findUser()* method, which is expected to find and return a User object

given a user name. The *AuthGeneric::handle_request* handles all the cascading of DEFAULT users, checking of check items, assembling replies etc. All you have to do is find the user in your database and return it in *findUser()*.

AuthGeneric only responds to Access-Request messages. Accounting-Requests are accepted but ignored (i.e. it does nothing with them). If you want to do something with accounting messages (other than what *Realm.pm* does, such as logging to the accounting log file or wtmp file), you will probably want to override *handle_request*, pass Access-Request messages to *$self->SUPER::handle_request()*, and process Accounting- Request messages yourself.

If your handler needs to fork so it can do a “slow” authentication or accounting task, you can call *AuthGeneric::handlerFork*, which will arrange for the handler to *fork(2)*, and also arrange for the child to exit after handling is complete.

AuthGeneric has a number of other methods that you can override for specific functions like:

- *log($p, $s)* Logs a message that originated in that class. *$p* is a message priority (see `Log.pm`). *$s* is a message. The base class behaviour is to log to the all the logging systems configured into Radiator by calling *main::log($p, $s)*.

12.4. Step-by-step

Assuming you want to create a custom AuthBy module called XYZ, these are the basic steps:

- Figure out what parameters your new module needs from the Radiator configuration file to configure its behaviour.
- Copy AuthTEST.pm to AuthXYZ.pm.
- Replace all instances of TEST in AuthXYZ.pm with XYZ.
- Implement the *keyword* function so it parses the parameters that your new module needs.
- If your module needs any sub-objects, implement the object function to parse out any sub-objects (see example code in *Radius::Realm->object*).
- Implement the *handle_request* function. You may wish to handle Access-Requests and Accounting-Requests differently.
- Add a test realm to the configuration file with something like:

```
<Realm xxxxxx>
      RealmParameter1 ....
      <AuthBy XYZ>
            AuthByParameter1 ....
            .....
      </AuthBy>
</Realm>
```

- Restart or SIGHUP Radiator.
- Test your new module by sending requests to the server with the *radpwtst* utility.
- Install your new module with *make install*.

If your authentication method is simple and synchronous, your class only has to override the following methods in AuthGeneric:

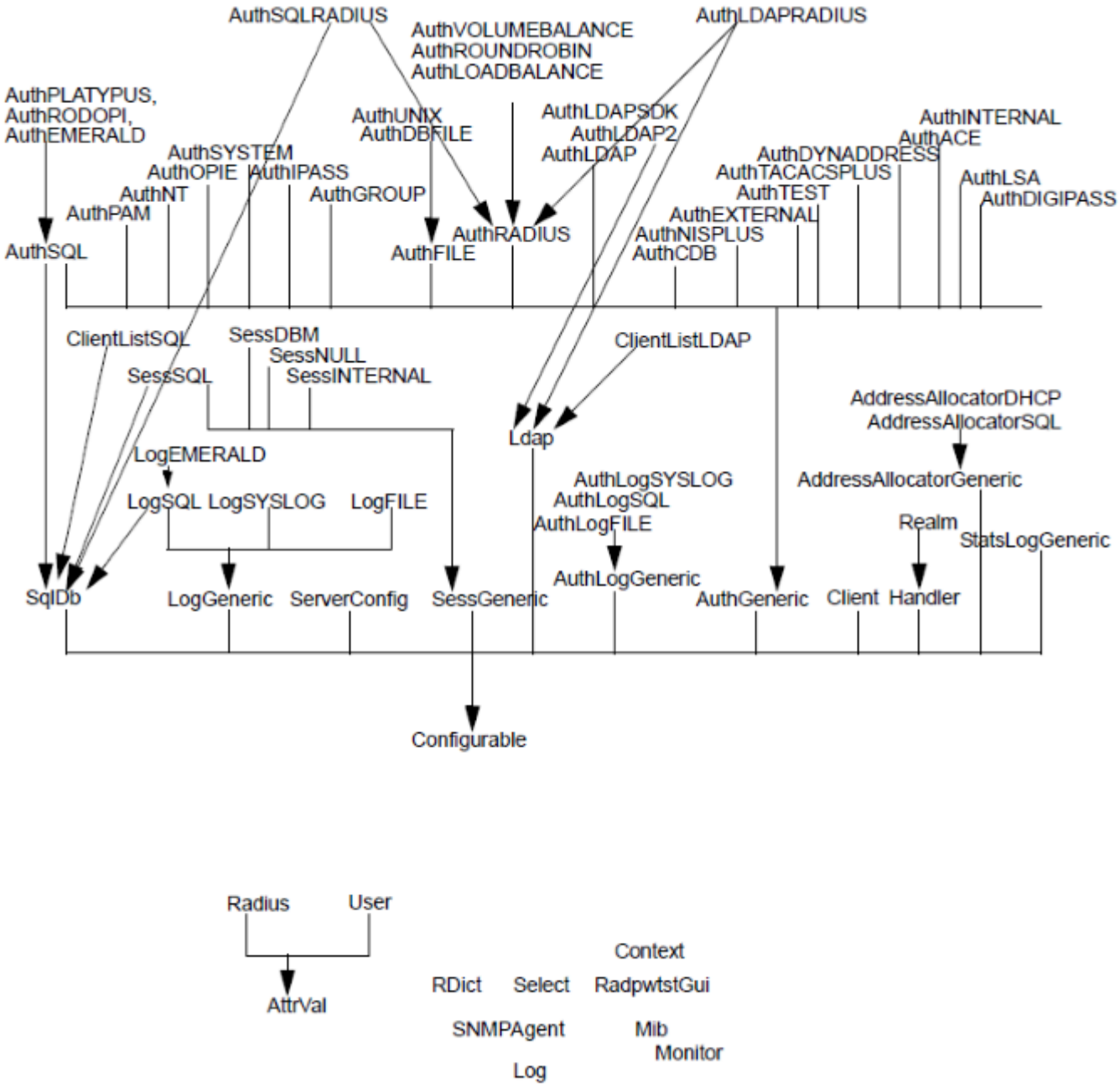
- *new* Create a new instance by calling *$class->SUPER::new($file);*
- *keyword* Recognise any class specific parameter keywords from the configuration file and remember them.

- *findUser* Construct and return a User object if the named user can be found in your database.

12.5. Class Hierarchy

This section is only of interest to developers who plan to build new Radiator classes. It shows the class inheritance hierarchy of all the classes provided with Radiator.

Figure 21. Radiator Class inheritance hierarchy



13. Compatibility with other servers

The flexibility of Radiator allows you to easily emulate the behaviour of Livingston and other similar radius servers. These radius servers use special entries in the user database file to control the behaviour of the server.

Radiator, in contrast, uses the configuration file for controlling the server. Nevertheless, Radiator allows you to use the Group and Auth-Type check items that are sometimes used with Livingston compatible servers.

There is an example configuration file in goodies/livingCompat.cfg in the Radiator distribution. If you use Livingston style “Auth-Type = System” in a user entry in the user file, Radiator will consult the UNIX password file /etc/password to authenticate the user. Radiator will also behave in the same way with respect to DEFAULT users. You can have multiple DEFAULT users in a user database. During authentication, if there is no user name entry found for the user, the DEFAULT entries are checked in the order in which they appear in the file (this is the case with both AuthBy FILE and AuthBy DBM). The DEFAULT entries will continue to be checked in order until one is found where all the check items match and where the Fall-Through reply item is not set to “Yes”.

Radiator Software can assist with converting database dumps from other commercial RADIUS servers to standard Livingston format users file, including:

- Cisco ACS database dump files
- Funk RIF database export files

Contact info@radiatorsoftware.com for more details.

14. Execution sequence and hook processing

This section describes when and how hooks are processed.

Radiator supports Perl hooks in many places in the configuration file. A hook is a piece of Perl code (in fact a Perl subroutine) that is run by Radiator at specified times. This allows the administrator to add custom code to handle unique or special processing requirements at run-time.

Hooks are specified verbatim or by file name in the Radiator configuration file.

Hooks are parsed by Perl at startup time, when the Radiator configuration file is read or re-read. Any syntax or compile-time errors in the hook are reported at that time.

Hooks are run at their specified times inside a Perl eval. This means that run-time errors, or a **`perl die()`** command results in a ERROR level message logged by Radiator.

Each hook is passed a set of run-time arguments, which depend on the type of hook being called. Arguments might include, for example, a reference to the request currently being processed, allowing the custom code to inspect and depend on the contents of the request. Hook arguments are documented separately for each hook.

The code for a hook can be placed verbatim in the configuration file:

```
UserPasswordHook sub {return $_[1]->{'GN'}}
```

Extend it over several lines using backslash line ending:

```
ConnectionHook sub {$_[1]->func(-access_mode => 'read_write',\
      -isolation_level => 'read_committed',\
      -lock_resolution => 'wait',\
      'ib_set_tx_param')}
```

You can also place the hook code in a separate external file that is named in the configuration file:

```
AcctHook file:"%d/hooks/sqlradacct.pl"
```

The code in the external file looks like this:

```
# Contents of file sqlradacct.pl:
sub
```

```
{
      use DBI;
      my ($p, $rp, $handled, $reason) = @_;
      ........
}
```

Tip

An external file can define several Perl functions that can be called from other functions in that file, or by another hook.

The standard hooks are:

- *StartupHook($restarted)* on page 38
- *PreClientHook(\$request)* on page 39
- *ClientHook(\$request)* on page 107
- *PreHandlerHook(\$request)* on page 78
- *PreProcessingHook(\$request, \$reply)* on page 153
- *PreAuthHook(\$request, \$reply)* on page 154
- *PostAuthHook(\$request, \$reply, \$handledflag, \$reason)* on page 176 (For AuthBy)
- *PostAuthHook(\$request, \$reply, \$handledflag, \$reason)* on page 154 (For Handler)
- *PostProcessingHook(\$request, \$reply)* on page 153
- *MainLoopHook()* on page 41
- *ReplyHook(\$replyfromproxy, \$replytonas, \$originalrequest, \$senttoproxy, $host)* on page 212 (<AuthBy RADIUS> only, when a reply is received)
- *CacheReplyHook($authobject, $user_name, $request, $reply)* on page 174 (Some authenticators, such as <AuthBy RADIUS>, when no reply is received, and *CachePasswords* is set)
- *NoReplyHook(\$originalrequest, \$senttoproxy \$replytonas)* on page 213 (<AuthBy RADIUS> only, when no reply is received)
- *PostSearchHook($self, $name, $p, $user, $entry, $rp)* on page 230 (<AuthBy LDAP2> and <ClientListLDAP> only, after the LDAP search has completed)
- *PostAuthSelectHook($self, $name, $p, $user, \@row)* on page 200 (<AuthBy SQL> only, after the SQL query has completed. <AuthBy RADMIN> has different set of arguments.)

Note that arguments preceded by '\' are passed by reference. You need to dereference them to get at the actual parameter. It is done this way to enable you to switch the object it points to. In any case, to reference the request in, such as *PreClientHook*, you would need to refer to it as *${$_[0]}*, and the second argument to a hook would be accessed as *${$_[1]}*, etc. For example, to get an attribute from the request in *PreClientHook*, you would use something like:

```
my $modtype = ${$_[0]}->get_attr('USR-Modulation-Type');
```

Some example hooks are found in `goodies/hooks.txt` in your distribution.

Hooks are executed at fixed times during request processing:

1. Server started

2. *StartupHook* called
3. Radiator waits for requests
4. Request received from NAS
5. Global RewriteUsernames applied
6. *PreClientHook* called
7. Client clause selected
8. Global *ClientHook* called
9. Client-specific *ClientHook* called
10. Client *RewriteUsernames* applied
11. Duplicate detection done
12. *PreHandlerHook* called
13. Handler selected
14. *PreProcessingHook* called
15. Handler's *RewriteUsername* and *RewriteFunction* applied
16. Session database updated (accounting requests only)
17. Accounting log files (AcctLogFileName and WtmpFileName) written
18. *PreAuthHook* called
19. AuthBy clauses invoked. AuthBy SQL may call *PostAuthSelectHook*, AuthBy LDAP2 may call *PostSearchHook* and other AuthBys may call their specific hooks. AuthBy-specific *PostAuthHook* called after each AuthBy
20. Handler-specific *PostAuthHook* called
21. Statistics updated
22. *PostProcessingHook* called (if there is a reply to be sent)
23. *AuthLog* and *AcctLog* called
24. Reply sent to NAS (unless request was proxied)
25. Reply received from proxy server (if the request was proxied to another RADIUS server...)
26. *ReplyHook* called
27. *PostProcessingHook* called
28. Reply sent to NAS
29. If no reply was received from a proxy server by <AuthBy RADIUS>, even after multiple retransmissions and timeouts, *CacheReplyHook* is called (if CachePasswords is specified), then *NoReplyHook* is called.
30. After all requests have been satisfied and timers run at the end of the main loop, *MainLoopHook* is run. Typically this happens once per second.
31. *ShutdownHook* called after receiving a SIGTERM and before exiting.

15. Interoperation with iPASS Roaming

iPASS (TM) operate a Global Roaming system that can allow your users to log in at any cooperating ISP around the world. For more information, see iPASS website [http://www.ipass.com/]. In order to interoperate you must

enter into a commercial arrangement with iPASS. iPASS will then be able to provide the software that Radiator needs to communicate with the iPASS network.

iPASS regard roaming as two distinct products:

- NetServer, where you originate authentication and accounting requests for users who dial in to your POP. The requests are sent to the iPASS network, where they are authenticated, possibly by someone else's Roam Server. This happens when someone else's customers dial in to your POP. We call this “outbound”.
- RoamServer, where authentication and accounting requests arrive from the iPASS network. This happens when your customers dial in to someone else's POP. We call this “inbound”.

Radiator uses different methods for handling inbound and outbound iPASS requests, and each must be set up separately with Radiator and with iPASS.

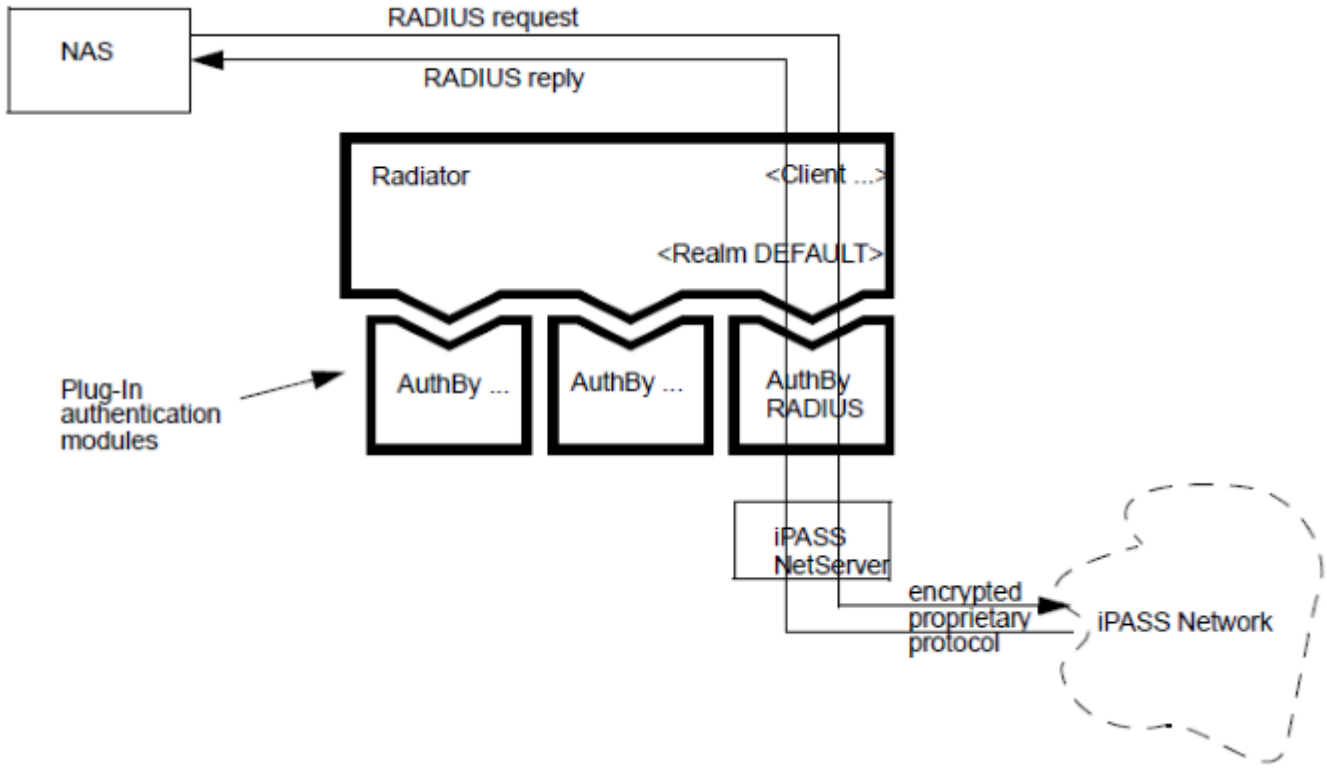
15.1. iPASS Outbound

Outbound requests must be proxied to an NetServer, configured to work with the iPASS system. The NetServer is provided by iPASS. The NetServer may be run either on the same host as Radiator, or on a different host. If the NetServer is run on the same host as Radiator, it must be configured to use different ports to Radiator.

As an example, here is part of a typical configuration that will handle requests for local users from a file, and proxy all other realms to an NetServer running on another host:

```
# Local realm is handled locally
<Realm my.local.realm>
      <AuthBy FILE>
            Filename xxxxxx
      </AuthBy>
</Realm>
# Al other realms are proxied to NetServer on fred
<Realm DEFAULT>
      <AuthBy RADIUS>
            Host fred
            Secret mysecret
      </AuthBy>
</Realm>
```

Figure 22. Schematic diagram of how iPASS outbound requests are handled



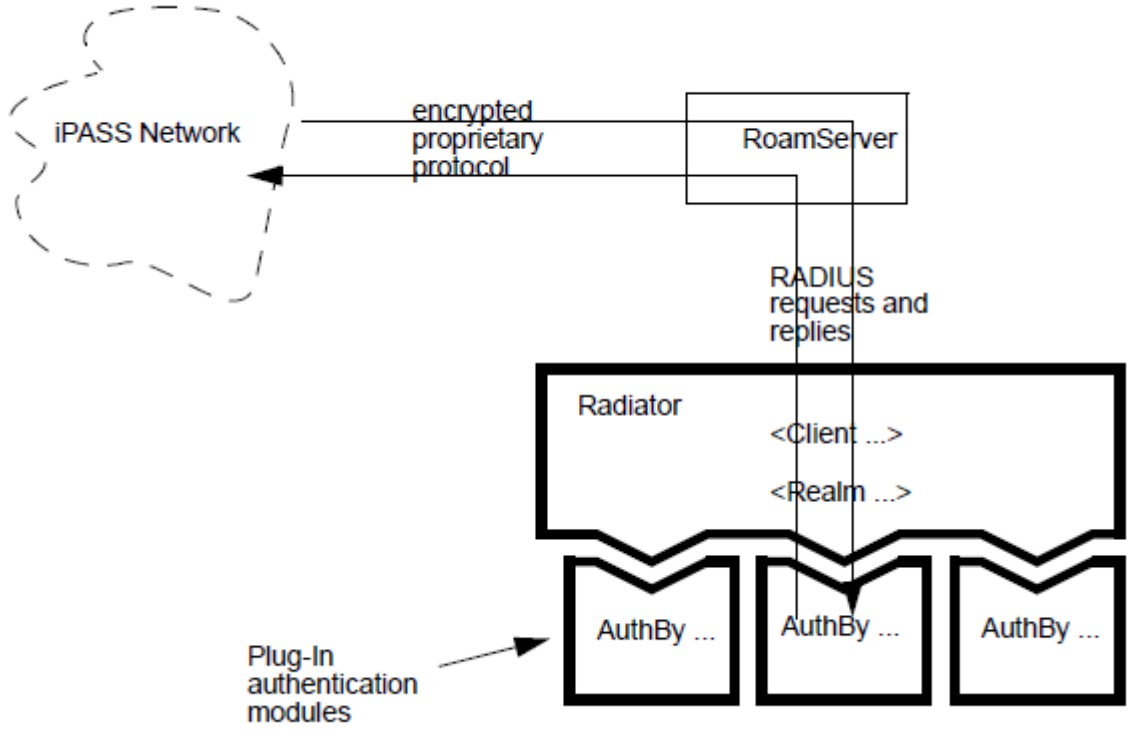
In order to configure Radiator to handle outbound iPASS requests, you need to do the following things:

1. Enter into a commercial arrangement for iPASS to provide Net Server access to you. iPASS will provide you with an ISP partner number.
2. Download, install and configure the iPASS software. You will need to configure both the RoamServer and RADIUS server. This will involve configuring the package, requesting and receiving an encryption certificate, and submitting details of your server and realm to iPASS. Install the package in the normal place (`/usr/ipass`).
3. Test the installed iPASS package by using the test programs provided with it. Make sure it is really working properly before you go on to the next step.
4. Configure Radiator so that all realms that are not handled locally are forwarded to the NetServer.
5. Test Radiator with the radpwtst program to make sure that requests for non-local realms are forwarded to iPASS.

15.2. iPASS Inbound

Inbound requests are received by a special server that you also must get from iPASS called RoamServer. RoamServer receives requests from the iPASS network and then sends them to Radiator as ordinary RADIUS requests. Seethe following figure. RoamServer will usually run on the same host as Radiator, or possibly on a different host in your network.

Figure 23. Schematic diagram of how iPASS inbound requests are handled



In order to configure Radiator to handle inbound iPASS requests, you need to do the following things:

1. Enter into a commercial arrangement for iPASS to provide Roam Server access to you. iPASS will provide you with an ISP partner number.
2. Request the iPASS RoamServer software for your platform from iPASS.
3. Install and configure the RoamServer software according to the instructions included with it. This will involve configuring the package, requesting and receiving an encryption certificate, and submitting details of your server and realm to iPASS. Install the package in the normal place (`/usr/ipass`). If you have already done this for outbound requests above, you do not need to do it again.
4. Configure Radiator in the usual way for your local realms. Add a Client clause specifying the host where the RoamServer software is running, and the shared secret you configured into RoamServer:

```
<Client localhost>
      Secret secret
</Client>
<Realm ...>
      ....
```

5. Test that RoamServer sends requests to Radiator by using the test software provided with RoamServer.

16. RadSec (RFC 6614)

It is often the case that RADIUS requests need to be sent from one RADIUS server to another. This is called proxying. It is commonly used to send requests to another RADIUS server that is better able to handle the request. In Radiator proxying is usually implemented with the <AuthBy RADIUS> clause.

Conventional UDP based RADIUS proxying is inherently unreliable and insecure. It is unreliable because the UDP protocol does not guarantee delivery, and the RADIUS protocol only requires a limited number of retransmits. Therefore, in an unreliable or congested network, RADIUS packets may be irretrievably lost. Further, conventional RADIUS requests are not encrypted, and most of the attributes (other than the password) in a RADIUS request are in plaintext. This means that eavesdroppers can obtain valuable information from unprotected RADIUS requests.

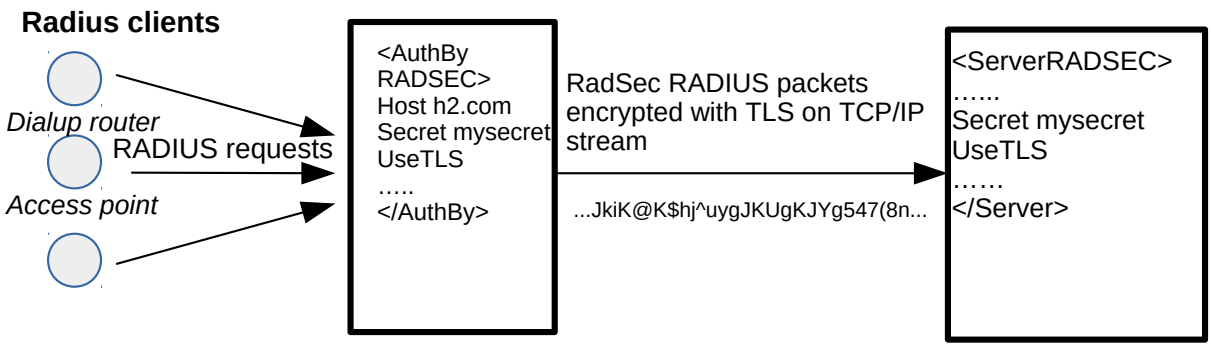
As a result, RFC 6614 ‘Transport Layer Security (TLS) Encryption for RADIUS’ was created to specify a secure, reliable RADIUS transport. RFC 6614 is based on the original RadSec protocol in Radiator, and the Radiator RadSec implementation is by default compliant with RFC6614.

RFC 6614 RadSec is a communication protocol that provides secure, reliable proxying of RADIUS requests from one Radiator to another. It can be used in place of AuthBy RADIUS when proxying across insecure or unreliable networks.

When using RadSec, one Radiator is designated the RadSec client, and the other is the RadSec server. The RadSec client establishes the connection to the RadSec server, and sends RADIUS requests to the RadSec server over a reliable stream connection. The RadSec server sends any replies to each RADIUS request back to the RadSec client. A RadSec server can accept connections from any number of RadSec clients.

RadSec uses the TCP/IP (or optionally SCTP) stream protocol to transport requests from the RadSec client to the RadSec server and replies from the server to the client. By default, TLS is used to encrypt the requests, and to enforce server authentication with a server PKI certificate (i.e. the RadSec client confirms that it is connected to the RadSec server it expects by checking a server certificate). By default, the Radiator RadSec server requires that clients confirm their identity by requiring a PKI client certificate.

Figure 24. Using RadSec to proxy request from one Radiator to another



For more information about RadSec, including a description of the RadSec protocol, RadSec white paper [https://files.radiatorsoftware.com/radiator/whitepapers/radsec-whitepaper.pdf]. See also RFC 6614.

16.1. RadSec Certificate Validation

During the establishment of a TLS RadSec connection between a RadSec client and RadSec server, certificate validation is performed in order to confirm that they are connected to the peer they expect to be connected to. This is a brief description of how validation is performed.

If a peer presents a certificate (*TLS_RequireClientCert*), then it is always validated using the peer certificate issuer's Root Certificate (see Section 3.11.2. TLS_CAFile on page 81 or Section 3.11.3. TLS_CAPath on page 81) and any certificate revocation list (CRL) for the certificate's issuer (see Section 3.11.17. TLS_CRLCheck on page 85 and Section 3.11.20. TLS_CRLFile on page 85). If *TLS_PolicyOID* is defined, the OIDs must be present in the certificate path. If that is successful, the contents of the peer certificate are checked:

In the RadSec server the client certificate is examined. It is accepted if:

- The certificate contains a *subjectAltName* extension of type IPADDR that matches the client's IP address.
- There were no *subjectAltName* extensions, but the certificate Subject contains a Common Name (CN) that matches the client's IP address.
- The certificate Subject matches the *TLS_ExpectedPeerName* pattern.

and:

- If *TLS_SubjectAltNameURI* is defined in the *<ServerRADSEC>* clause, the certificate must contain a *subjectAltName* of type URI that matches the *TLS_SubjectAltNameURI* regular expression.
- If *TLS_CertificateFingerprint* is defined in the *<ServerRADSEC>* clause, the certificate's fingerprint must match at least one of the *TLS_CertificateFingerprint* options.

In the RadSec client, the server certificate is examined. It is accepted if:

- The certificate contains a subjectAltName extension of type IPADDR or DNS that matches the Host name used to connect to the server. If *TLS_SubjectAltNameDNS* is set, use its value to match type DNS.
- There were no *subjectAltName* extensions, but the certificate Subject contains a Common Name (CN) that matches the Host name used to connect to the server.
- The certificate Subject matches the *TLS_ExpectedPeerName* pattern.

and:

- If *TLS_SubjectAltNameURI* is defined in the *<AuthBy RADSEC>* clause, the certificate must contain a *subjectAltName* of type URI that matches the *TLS_SubjectAltNameURI* regular expression.
- If *TLS_CertificateFingerprint* is defined in the *<AuthBy RADSEC>* clause, the certificate's fingerprint must match at least one of the *TLS_CertificateFingerprint* options.
- *TLS_SRVName* check passes, usually applicable to *<AuthBy RADSEC>* utilised by *<AuthBy DNSROAM>*.

17. Extensible Authentication Protocol (EAP)

Extensible Authentication Protocol (EAP) is a standard for defining and extending authentication protocols. EAP is defined by RFC 3748, and RFC 2869 defines how EAP authentication messages are carried in RADIUS packets. Radiator complies with these standards, and can be extended to handle any EAP-compliant protocol. Because EAP is designed to be easily extensible, a number of EAP protocols have been defined for various special requirements This includes mutual authentication between client and RADIUS server, and encryption of the authentication conversation. EAP over RADIUS is commonly used as the authentication protocol for 802.1X wired and wireless networks. For more information, see IETF website [https://www.ietf.org/]

Conventional RADIUS requests send the User-Name in one RADIUS attribute and the User-Password in another attribute. All EAP-over-RADIUS requests send their authentication information in the EAP-Message attribute. Usually the client sends some EAP protocol information in the EAP-Message attribute in a RADIUS Access-Request message, and the RADIUS server asks for some more information from the client by sending back another EAP-Message in a RADIUS Access-Challenge. When the server has enough information from the client, the RADIUS server replies with an Access-Accept or an Access-Reject message. Some EAP protocols,

such as TLS, TTLS and PEAP, often require a number of messages (10 or more) to be exchanged between client and RADIUS server during authentication. Such a group of EAP-over-RADIUS messages while authenticating a user is called a conversation.

Note that many EAP protocols hide or encrypt the identity of the real user name of the user being authenticated, and send a generic name (typically anonymous) as the visible User-Name in the RADIUS requests.

Radiator supports several EAP protocols. This section describes their characteristics and how Radiator can be configured to support them. In order to configure Radiator to support EAP, the relevant AuthBy clauses must have the *EAPType* parameter set to the names of the EAP types that you need to support. In addition, a number of other special configuration parameters may have to be set in order to support those EAP types. An AuthBy clause can be configured to accept more than one EAP type. When that is the case, the first one in the list will be the default EAP method offered to the client. The client can accept the offered method or request another EAP method. If the requested method is one of the methods named in the *EAPType* parameter, then EAP authentication continues with that method.

There are many example configuration files covering various EAP authentication requirements in the `goodies/` directory of the Radiator distribution. See `goodies/README` for details.

17.1. EAP MD5-Challenge

In EAP MD5-Challenge, the RADIUS server sends a random challenge to the client. The client forms an MD5 hash of the user's password and the challenge and sends the result back to the server. The server then validates the MD5 hash using the known correct plaintext password from the user database. EAP MD5-Challenge does not support dynamic WEP keys.

EAP MD5-Challenge can work with most Radiator AuthBy clauses that support the retrieval of a plaintext password, such as FILE, DBFILE, SQL, LDAP etc.

17.2. EAP One-Time-Password

In EAP One-Time-Password, the client sends a plaintext one-time password to the server, which then checks the password and either accepts or rejects the request. The one-time password is sent in the clear, so it is subject to eavesdropping, and should not be used with static password. EAP One-Time-Password does not support dynamic WEP keys.

EAP One-Time-Password can work with Radiator AuthBy clauses that support one-time passwords, such as AuthBy OTP, LDAPDIGIPASS and SQLDIGIPASS.

17.3. EAP Generic-Token

EAP Generic-Token is intended to support a range of authentication tokens from various vendors. In this context, a ‘token’ is a small device that the end user carries and which displays a login passcode that is used to authenticate the user. Examples are RSA SecurID, Encotone TeleID and Vasco tokens. Most such systems have back end servers that do the actual authentication. EAP Generic-Token does not support dynamic WEP keys.

EAP Generic-Token can be used with Radiator AuthBy clauses ACE, LDAPDIGIPASS, SQLDIGIPASS, OTP and RSAAM to support one-time passwords. It can also be used with all AuthBy clauses that support static passwords, such as FILE, SQL, LDAP, UNIX etc. In this case, it prompts the user ‘Enter your static password’.

17.4. EAP TLS

EAP TLS uses Public Key Infrastructure (PKI) digital certificates to provide mutual authentication between the EAP client and the RADIUS server. A PKI certificate is a file created by a program called a Certificate Authority. The certificate contains the name of the server or user that has been issued to. The EAP client and RADIUS server use the certificates to verify that the other party is indeed who it claims to be. In EAP TLS, a

PKI certificate is required for the Radiator RADIUS server and for each and every EAP TLS client. EAP TLS does support dynamic WEP keys.

You can obtain certificates from a Public Certificate authority such as Thawte [https://www.thawte.com/]. The advantage of Public Certificates is that they will generally be recognised by any client or server without taking any special steps. A disadvantage of Public certificates is that you usually have to pay an annual fee for each one. With a Private Certificate Authority, you can generate your own server and client certificates for free, but you will generally have to install the 'Root Certificate' from your Certificate Authority on each client before it will recognise a private server certificate. Private Certificates are considered by many to be more secure that Public Certificates.

The basic steps of EAP TLS authentication are:

1. The EAP TLS client and RADIUS server establish a communications channel via the RADIUS protocol.
2. The RADIUS server sends its Server PKI Certificate to the client.
3. The client verifies that the server certificate is valid and is the correct certificate for the RADIUS server it is communicating with. It uses the Root Certificate of the Certificate Authority that issued the Server Certificate to validate the Server Certificate. (Root Certificates for most Public Certificate Authorities are built in to most clients. If the Server Certificate was issued by a Private Certificate Authority, the client requires a copy of the Root Certificate to be installed in order to validate the Server Certificate.)
4. If the client validates the server certificate, it then sends the user's PKI certificate to the RADIUS server.
5. The RADIUS server verifies that the client certificate is valid and is the correct certificate for the user name that is being authenticated. The RADIUS server can be configured to validate Private Client Certificates using a locally installed copy of the Root Certificate of the Certificate Authority that issued the client certificate.
6. If the RADIUS server validates the client certificate then the authentication is successful, and the client is permitted to be connected to the network.

EAP TLS does not use or exchange any passwords, but you can use AuthBy methods in Radiator to enable or disable EAP TLS users based on their user name, time of day etc.

17.5. EAP LEAP

EAP LEAP (often called just LEAP) is a proprietary protocol from Cisco [https://www.cisco.com/]. It provides password based authentication that can be used with any Radiator authentication method that stores plaintext passwords (such as FILE, SQL, LDAP) etc., as well as direct Windows authentication using AuthBy LSA. LEAP authentication requires a special LEAP client that is only available from Cisco.

17.6. EAP TTLS

Like EAP TLS (see Section 17.4. EAP TLS on page 519), EAP TTLS uses Public Key Infrastructure (PKI) digital certificates. Unlike TLS, it only uses a Server Certificate so the client can validate the server, and then establish a secure, encrypted communications channel with the RADIUS server. When this channel is established, it is used to tunnel conventional RADIUS attributes, such as User-Name, User-Password etc. to the RADIUS server. Radiator converts each of these so-called 'inner requests' into a new RADIUS request which can be authenticated by any supported AuthBy method. So EAP TTLS authentication happens in 2 phases following these basic steps:

1. The EAP TTLS client and RADIUS server establish a communications channel via the RADIUS protocol.
2. The RADIUS server sends its Server PKI Certificate to the client.
3. The client verifies that the server certificate is valid and is the correct certificate for the RADIUS server it is communicating with. It uses the Root Certificate of the Certificate Authority that issued the Server

Certificate to validate the Server Certificate. (Root Certificates for most Public Certificate Authorities are built in to most clients. If the Server Certificate was issued by a Private Certificate Authority, the client requires a copy of the Root Certificate to be installed in order to validate the Server Certificate.)

4. If the client validates the server certificate, it then sends the real user name and password in a RADIUS request through the encrypted TLS tunnel. Any conventional RADIUS authentication system may be used depending on the client configuration, such as PAP, CHAP, MSCHAP, MSCHAPV2 etc.
5. Radiator converts this 'inner' request into a new RADIUS request and dispatches it to the first matching Realm or Handler clause, where it can be handled by one or more AuthBy clauses. To assist in discriminating TTLS inner requests, each inner request is tagged with the pseudo-attribute TunnelledByTTLS set to 1.
6. The result of the inner authentication is sent back to the client through the TLS tunnel.

In order to use EAP TTLS, you must install a unique Server Certificate on your RADIUS server host, and configure Radiator to use it. For more information about Public and Private certificates and how to obtain them, see Section 17.4. EAP TLS on page 519. EAP TTLS does support dynamic WEP keys.

You can configure Radiator to handle the inner and outer requests in separate Handler or Realm clauses. You can also configure Radiator to proxy the inner RADIUS requests to another RADIUS server, which means that Radiator can server as a gateway between EAP TTLS clients and a non-EAP enabled RADIUS server.

17.7. EAP SIM

This protocol is used to authenticate using a GSM SIM card. EAP-SIM support is available as an add-on package for Radiator. See https://radiatorsoftware.com/products/radiator-sim-pack/ [https://radiatorsoftware.com/products/radiator-sim-pack/] or contact info@radiatorsoftware.com for more details.

17.8. EAP AKA

This protocol is used to authenticate using a UMTS SIM card. EAP-AKA support is available as an add-on package for Radiator. See https://radiatorsoftware.com/products/radiator-sim-pack/ [https://radiatorsoftware.com/products/radiator-sim-pack/] or contact info@radiatorsoftware.com for more details.

17.9. EAP AKA'

This protocol is a revision of EAP-AKA and is used to authenticate using a UMTS SIM card. EAP-AKA' support is available as an add-on package for Radiator. See https://radiatorsoftware.com/products/radiator-sim-pack/ [https://radiatorsoftware.com/products/radiator-sim-pack/] or contact info@radiatorsoftware.com for more details.

17.10. EAP PEAP

Like EAP TLS (See Section 17.4. EAP TLS on page 519), EAP PEAP (often called just PEAP) uses Public Key Infrastructure (PKI) digital certificates. Unlike TLS, it only uses a Server Certificate so the client can validate the server, and then establish a secure, encrypted communications channel with the RADIUS server. When this channel is established, it is used to tunnel encrypted EAP messages to the RADIUS server. Radiator converts each of these so-called 'inner requests' into a new RADIUS request which can be authenticated by any supported AuthBy method. So EAP PEAP authentication happens in 2 phases following these basic steps:

1. The EAP PEAP client and RADIUS server establish a communications channel via the RADIUS protocol.
2. The RADIUS server sends its Server PKI Certificate to the client.
3. The client verifies that the server certificate is valid and is the correct certificate for the RADIUS server it is communicating with. It uses the Root Certificate of the Certificate Authority that issued the Server Certificate to validate the Server Certificate. (Root Certificates for most Public Certificate Authorities are

built in to most clients. If the Server Certificate was issued by a Private Certificate Authority, the client requires a copy of the Root Certificate to be installed in order to validate the Server Certificate.)

4. If the client validates the server certificate, it then sends one or more EAP requests through the encrypted TLS tunnel. The type of inner EAP request depends on the PEAP client configuration, but the most common types of inner EAP requests are EAP MSCHAPV2 and EAP TLS.
5. Radiator converts this 'inner' request into a new RADIUS request and dispatches it to the first matching Realm or Handler clause, where it can be handled by one or more AuthBy clauses. To assist in discriminating PEAP inner requests, each inner request is tagged with the pseudo-attribute TunnelledByPEAP set to 1.
6. The result of the inner authentication is sent back to the client through the TLS tunnel.

In order to use EAP PEAP, you must install a unique Server Certificate on your RADIUS server host, and configure Radiator to use it. For more information about Public and Private certificates and how to obtain them, see Section 17.4. EAP TLS on page 519. EAP PEAP does support dynamic WEP keys.

You can configure Radiator to handle the inner and outer requests in separate Handler or Realm clauses. You can also configure Radiator to convert an inner EAP-MSCHAPV2 request into a conventional RADIUS-MSCHAPV2 request, which means that Radiator can act as a gateway between EAP PEAP clients and a non-EAP enabled RADIUS server.

17.11. EAP MSCHAPV2

EAP MSCHAPV2 is an EAP version of the common MSCHAPV2 authentication mechanism. It provides mutual authentication between client and server. It is most commonly used as the inner authentication protocol with EAP PEAP on Microsoft Windows clients. EAP MSCHAPV2 does support dynamic WEP keys.

EAP MSCHAPV2 can be used with any Radiator AuthBy that has access to plaintext passwords, such as FILE, SQL, LDAP2, DBM etc. It can also be used with AuthBy LSA to authenticate with a Windows Local Security Authority, Windows Domain Controller etc. It can also be used with LDAPDIGIPASS and SQLDIGIPASS.

17.12. EAP PAX

EAP PAX provides strong encryption and mutual authentication between supplicant and server based on a per-user Authentication Key (AK). It is described in RFC 4746. Based on the per-user AK, the server and supplicant derive strong cryptographic keys and authenticate each others knowledge of the AK. The derived keys can be used for dynamic WEP and WPA keys.

The AK is required to be configured into the per-user data in the Radiator user database, and also into each user's EAP-PAX supplicant configuration. The AK is required to be 16 bytes. It can be specified in a Radiator user database as 32 hex digits:

```
pskuser     User-Password=12345678901234567890123456789012
```

EAP PAX can be used with any Radiator user database that supports a plaintext *User-Password*.

17.13. EAP PSK

EAP PSK provides strong encryption and mutual authentication between supplicant and server based on a per-user Pre-Shared-Key (PSK). It is described in RFC 4764. Based on the per-user PSK, the server and supplicant derive strong cryptographic keys and authenticate each others knowledge of the PSK. The derived keys can be used for dynamic WEP and WPA keys.

The PSK is required to be configured into the per-user data in the Radiator user database, and also into each user's EAP-PSK supplicant configuration. The PSK is required to be 16 bytes. It can be specified in a Radiator user database as 32 hex digits:

```
pskuser     User-Password=12345678901234567890123456789012
```

If the User-Password does not appear to be 32 hex digits, it will be regarded as a plaintext password, and will be converted into a PSK using the algorithm described in RFC 4764. The conversion to a PSK depends on the plaintext password and the server and supplicant IDs. Use of such plaintext passwords is discouraged by RFC 4764 (because the PSK then becomes vulnerable to dictionary attacks) and is not supported by all EAP PSK supplicants. We also discourage use of such plaintext passwords.

EAP PSK can be used with any Radiator user database that supports a plaintext *`User-Password`*. Requires `Crypt::Rijndael`. For more information, see Section 2.1.2. CPAN on page 3.

17.14. EAP PWD

EAP PWD provides strong encryption and mutual authentication between supplicant and server based on a shared password. It is described in RFC 5931. Based on the per-user password, the server and supplicant derive strong cryptographic keys and authenticate each others knowledge of the password. The derived keys can be used for dynamic WEP and WPA keys.

EAP PWD is highly secure (the password is never transmitted, even in encrypted form), and does not require PKI certificates, and also requires only 3 authentication roundtrips. Further, it is not encumbered by intellectual property issues. So it is considered efficient to roll out in eduroam and other environments.

Authentication of EAP PWD by Radiator depends in having access to the user's plain text password. EAP PWD can be used with any Radiator user database that supports a User-Password in format like below. Some EAP PWD clients may also support additional password formats. For more information, see Section 3.10.58. EAP_PWD_PrepMethod on page 77. :

```
username    User-Passsword=fred
```

EAP PWD requires OpenSSL 0.9.8i libraries or later, `Crypt::OpenSSL::EC` and `Crypt::OpenSSL::Bignum` 0.06 or later.

Tip

`Crypt::OpenSSL::EC` and `Crypt::OpenSSL::Bignum` may not be readily available for Windows. We recommend Linux or Unix hosts for deployment of EAP PWD.

---

18. Monitor command language

The Monitor clause client programs to make an (authenticated) TCP connection to Radiator, and use that connection to monitor, probe, modify and collect statistics from Radiator. For more information, see Section 3.133. <Monitor> on page 436.

Monitor implements a simple command language, which is described in this section.

All Monitor connections must be authenticated with the LOGIN command before any significant command can be executed. If a connection has not been authenticated, the only commands that can be executed are LOGIN, HELP and QUIT.

Monitor supports 2 basic connection types:

- ASCII text with line-feed terminators, which permits use by Telnet or other interactive TCP programs. New connections always start in this mode.

- BINARY mode, which permits specialised external client programs (such as Radar from Radiator Software) to exchange information more efficiently. BINARY mode is entered with the BINARY command.

Monitor supports 2 types of authentication in the LOGIN command:

- Plaintext. The password is supplied as plaintext. Since the password will be transmitted as plaintext on a TCP connection, it is possible for eavesdroppers to sniff the password. Therefore you should only use this on interactive Telnet connections over trusted connections.
- CHAP. The password is supplied as CHAP response to a CHAP challenge. This is suitable for client programs to authenticate Monitor connections. CHAP passwords start with {chap}.

18.1. Object naming

Some Monitor commands accept Radiator object names as arguments. This allows you to identify specific internal Radiator objects to operate on or to monitor. A Radiator object is an internal Radiator structure which generally corresponds to a clause in the Radiator configuration file. There always exists a main object (ServerConfig) that represents the server as a whole, and which is the ultimate ‘parent’ of all the other objects. The Radiator objects form a tree structure with ServerConfig as the root.

You can identify specific Radiator objects with an object name. An object name consists of a number of dot (‘.’) separated words. A single dot is taken to mean the root, the main ServerConfig object. In object names, case is significant.

Consider this skeleton Radiator configuration file:

```
DbDir ....
Trace 4
...
<Client 1.2.3.4>
      ...
</Client>
<Realm xyz.com>
      <AuthBy FILE>
            ...
      </AuthBy>
</Realm>
```

Then the following object names could be used:

- .

  A single dot refers to the server as a whole, or ServerConfig. If you got STATS from . then you would receive the statistics for the whole server.

- .Client.0

  This refers to the first (and only) Client 1.2.3.4 clause.

- .Realm.0

  This refers to the first (and only) Realm xyz.com clause.

- .Realm.0.AuthBy.0

  This refers to the AuthBy FILE clause inside the Realm clause.

18.2. Commands

All command names are case insensitive, but arguments to commands may be case sensitive.

18.2.1. BINARY

Makes the connection run in BINARY mode. In BINARY mode, commands and their responses are separated by NULL characters. BINARY mode is only suitable for Monitor client programs. Do not use BINARY for interactive Telnet connections.

18.2.2. CHALLENGE

Requests a CHAP challenge from Monitor. The reply may be used to generate a CHAP response in a subsequent LOGIN command. This allows client programs to avoid sending plaintext passwords on the Monitor TCP connection.

18.2.3. DESCRIBE objectname

Describes the named object by returning a list of its attributes and their types.

18.2.4. GET objectname

Not implemented.

18.2.5. HELP

Prints a list of the supported commands.

18.2.6. ID

Prints the Radiator server identification string.

18.2.7. LIST objectname

If objectname names an object array parameter (such as a Client, Realm or Handler array), prints a list of the objects in the array and their indexes.

18.2.8. LOGIN username password

Authenticate the connection, so that privileged commands can be executed. Username and password must be authenticated by one of the Monitor's AuthBy clauses, or by the hardwired Username and Password (if present).

Password may be either a plaintext password or a CHAP response to a previously issued CHALLENGE command.

```
LOGIN mikem fred
LOGGEDIN
```

or

```
CHALLENGE
CHALLENGE 5b0e00b1a379ae225634946268cc0144
LOGIN mikem {chap}4bda435ca72249c0e9fe05d32775eb6e98
LOGGEDIN
```

18.2.9. RESTART

Causes Radiator to restart and reread its configuration file, with the same effect as a SIGHUP.

18.2.10. TRACE n

Causes Radiator to start printing log message at or below the given log level. The number n can be 0 to 5 inclusive. Each new Monitor connection starts out at level 0 (ERR level). Level 4 is DEBUG level. After setting a new Trace level, Radiator will send all log messages at or below that level on the Monitor connection.

If a TRACE_USERNAME is in force, only messages that result from requests from that specific User-Name will be printed.

18.2.11. TRACE_USERNAME user name

Causes TRACE to only print log messages that result from a RADIUS request from the given User-Name. This is handy for seeing only log messages for a given user when debugging connection problems.

The user name is compared to the User-Name in the incoming request after any RewriteUsername rules have been applied. Setting the TRACE_USERNAME to an empty string (the default) turns off this special behaviour, and all messages that are at or below the current TRACE level will be output.

In the following example, the TRACE level is set to 4, meaning all DEBUG and lower level log messages will be printed. Then the TRACE_USERNAME is set to mikem@open.com.au, which means that only DEBUG and lower log messages due to requests from mikem@open.com.au will be printed. Finally the TRACE_USERNAME is set to an empty string, which means that all DEBUG and lower level log messages will be printed again.

```
TRACE 4
TRACE_USERNAME mikem@open.com.au
      <<< only messages that are at or below the current TRACE level
      for requests from mikem@open.com.au
      are output here.>>>
TRACE_USERNAME
      <<< no more user-specific logging now. All trace level 4
      messages will be output >>>
```

18.2.12. SET objectname parameter value

Sets a new value for the named parameter in the named object. The new value will persist in the Radiator until it is restarted. This is useful for experimenting with new configuration values without having to restart the server. It is probably most useful for setting the PacketTrace parameter, permitting DEBUG level tracing of all packets that pass through a given object.

For example, this command will set PacketTrace on the first Client clause. Subsequently, all packets arriving from that Client will by logged to this Monitor connection (and any other logger) at DEBUG level:

```
SET .Client.0 PacketTrace 1
```

18.2.13. STATS objectname

Requests all the statistics available from the named object.

18.2.14. QUIT

Forces disconnection from Monitor.

19. Using SQL with various database vendors

Radiator's SQL modules can be used with any Relational Database that is supported by a database specific DBD module or ODBC DBD module. Some examples are: Oracle, Microsoft SQL Server, PostgreSQL, MySQL, MariaDB, SQLite, Firebird, Sybase and Informix. Many of these databases have both: you can use a database specific DBD module or DBD::ODBC to connect to them.

We have not directly tested every one of these. We have tested some of them, and this section contains some tips about using Radiator with them. We supply some simple schemas with Radiator for a few databases. See the `goodies/*Create.sql` files in the distribution for more information about them. You will probably want to create a more elaborate schema to handle the tasks you need, and the schemas we supply should be regarded as a starting point only. You should probably consult with a Database Analyst to maximise the performance of your SQL database.

19.1. General

Whenever Perl's DBI module is used to work with a database, you need to supply up to 3 pieces of information in order to specify the database to which you want to connect:

- DBSource

  This is the data source name. It has the special format: “dbi:drivername:options”, where driver name is the name of the DBI driver to use, and options is an option string whose exact format depends on the DBI driver you are using.

- DBUsername

  This is usually the SQL user name to use to connect to the SQL database, but for some database types, it has a different meaning.

- DBAuth

  This is usually the password for DBUsername, but for some database types, it has a different meaning or is not required.

19.2. MySQL and MariaDB

In DBSource, driver name is “mysql”, and options is the database name. DBUsername is the MySQL user name, and DBAuth is the password for the MySQL user.

To create a new database, see `goodies/mysqlCreate.sql` for more information.

Configure your SQL clause like this:

```
DBSource    dbi:mysql:radius
DBUsername  radius
DBAuth      password
```

Tip

To specify server name and port, use:

```
dbi:mysql[:database[;host=hostname[;port=port]]]
```

19.3. PostgreSQL

In DBSource, driver name is “Pg”, and options is the database name to use. DBUsername is the PostgreSQL user name, and DBAuth is the password for the PostgreSQL user.

To create a new database, see `goodies/postgres-sqliteCreate.sql` for more information.

Configure your SQL clause like this:

```
DBSource    dbi:Pg:dbname=radius
DBUsername  user
DBAuth      password
```

Tip

To specify server name and port, use:

```
dbi:Pg:dbname=radius[;host=hostname[;port=port]]
```

19.4. Oracle

In DBSource, driver name is “Oracle”, and options is the SID of the Oracle database instance you want to use. DBUsername is the Oracle user name, and DBAuth is the password for the Oracle user.

To create a new database, see `goodies/oracleCreate.sql` for more information.

Configure your SQL clause like this:

```
DBSource    dbi:Oracle:sid
DBUsername  user
DBAuth      password
```

Tip

To specify server name and port directly without $ORACLE_HOME/network/admin/tnsnames.ora, use:

```
dbi:Oracle:service_name=xe;host=10.20.30.40;port=1521
```

19.5. Microsoft SQL Server

Multiple methods are available to connect to Microsoft SQL Server.

If you run Radiator on Windows, use the ODBC administration tools and add a System DSN (data source name). Microsoft provides ODBC support for Linux and macOS with *Microsoft ODBC Driver for SQL Server on Linux and macOS*. The driver and installation instructions are available from the Microsoft web. For more about ODBC configuration on all operating systems, see Section 19.8. ODBC on page 529

To create a new database, see `goodies/sqlserver-sybaseCreate.sql` for more information.

Configure your SQL clause like this:

```
DBSource    dbi:ODBC:radius
DBUsername  radius
```

```
DBAuth      password
```

If you can not use Microsoft ODBC driver, consider DBD::Sybase with FreeTDS. Sybase and MS SQL Server have common history and for this reason DBD::Sybase is known to work with Microsoft SQL Server. You need to compile and link DBD::Sybase with FreeTDS libraries. For more about DBD::Sybase, see Section 19.9. Sybase on page 530

Yet another option is to use the ODBC driver that comes with FreeTDS and configure Radiator to use DBD::ODBC. For more about DBD::ODBC, see Section 19.8. ODBC on page 529

19.6. SQLite

DBD::SQLite is a lightweight integrated SQL library that keeps its database in a single flat file. It does not use a separate SQL server: all the SQL functions are embedded in the SQLite library. DBD::SQLite is available from CPAN. For more information, see Section 2.1.2. CPAN on page 3.

In DBSource, driver name is “SQLite”, and a required option is *`dbname`*, which specifies the path name of the single file where the database is to be stored. It can be an absolute file path or a relative path. DBUsername and DBAuth are not required and are ignored.

To create a new database, see `goodies/postgres-sqliteCreate.sql` for more information.

Configure your SQL clause like this. DBUsername and DBAuth are not required:

```
DBSource    dbi:SQLite:dbname=/path/to/your/dbfile
```

19.7. Firebird

In DBSource, driver name is “Firebird”, and a required option is *`database`*, which specifies the path name of the file where the database is to be stored. DBUsername is the Firebird user name, and DBAuth is the password for the Firebird user.

To create a new database, see `goodies/ansiCreate.sql` for more information..

Configure your SQL clause like this:

```
DBSource    dbi:Firebird:database=/path/to/database.fdb
DBUsername  username
DBAuth      password
```

Tip

DBSource can also contain other hints and directives for connecting to the Firebird server, for example:

```
DBSource dbi:Firebird:port=3050;host=10.20.30.40;db=/path/to/database.fdb;
```

19.8. ODBC

To use ODBC, you must first create the database and tables in a way that depends on the type of database to which you are going to connect. See your vendor's documentation. You need to install and configure your ODBC manager. The way to do this also depends on your ODBC data manager.

For example, on Linux, macOS and other Unix systems, ODBC configuration typically is done with `odbcinst.ini` and `odbc.ini` files in `/etc`, `/usr/local/etc`, `$HOME` or other directory that depends on the system. For Windows, use the ODBC administration tools and add a System DSN (data source name).

Configure your SQL clause like this:

```
DBSource    dbi:ODBC:datasourcename
DBUsername  user
DBAuth      password
```

19.9. Sybase

In DBSource, driver name is “Sybase”, and the options can be empty. DBUsername is the Sybase user name, and DBAuth is the password for the Sybase user.

To create a new database, see `goodies/sqlserver-sybaseCreate.sql` for more information.

Configure your SQL clause like this:

```
DBSource    dbi:Sybase:
DBUsername  user
DBAuth      password
```

Tip

DBSource can also contain other hints and directives for connecting to the Sybase server, for example:

```
DBSource dbi:Sybase:server=SERVERNAME;database=DBNAME
```

You may need to set SYBASE environment variable so that the Sybase libraries can find their interfaces library. Otherwise you may need to use directives. You may also need to set LD_LIBRARY_PATH, or similar, to $SYBASE/lib.

19.10. InterBase

In DBSource, driver name is “InterBase”, and a required option is *`database`*, which specifies the path name of the file where the database is to be stored. DBUsername is the InterBase user name, and DBAuth is the password for the InterBase user.

To create a new database, see `goodies/ansiCreate.sql` for more information.

Be careful to review the InterBase specific notes in `goodies/ansiCreate.sql`. Some SQL clauses require a couple of little tweaks. Note that, for example, the name of the password column is PASS_WORD. PASSWORD is a reserved word in InterBase.

Configure your SQL clause like this:

```
DBSource    dbi:InterBase:database=/path/to/your/database.gdb
DBUsername  username
DBAuth      password
AuthSelect  select PASS_WORD from SUBSCRIBERS where USERNAME=%0
```

19.11. Informix

In DBSource, driver name is “Informix”. DBUsername is the Informix user name, and DBAuth is the password for the Informix user.

The Perl DBD::Informix module is very mature and was developed in conjunction with Informix development staff. Building DBD::Informix requires that you have a working Informix installation and DBA access to a test

database. We recommend that you carefully follow the procedures in the DBD::Informix README file, then build, test and install DBD::Informix before attempting to create and configure a Radiator database on Informix.

The create a new database, see `goodies/ansiCreate.sql` for more information.

You must ensure you have your INFORMIXDIR and INFORMIXSERVER environment variable set up specifying an operating Informix server. Also check that your `$INFORMIXDIR/etc/sqlhosts` file has correct information.

Configure your SQL clause like this:

```
DBSource    dbi:Informix:radius
DBUsername  user
DBAuth      password
```

19.12. CSV

DBD::CSV is a Perl database driver that uses flat text files as the database. The default supports comma separated files, but that can be customised in many ways, to support for example Unix password format and Excel spreadsheet formats. DBD::CSV is available from CPAN. For more information, see Section 2.1.2. CPAN on page 3.

In DBSource, driver name is "CSV", and a required option is f_dir, which specifies the directory where the database files exist. DBUsername and DBAuth are not required and are ignored.

DBD::CSV databases are flat text files. The file name is the table name, for example, `subscribers` for the default AuthSelect. The first line of a database file contains the names of the columns in the file. To create a new database file, you need to create the text file with an editor. A simple database file for subscribers that will work the default AuthSelect in AuthBy SQL would be named `subscribers` and it would look like this:

```
USERNAME,PASSWORD
mikem,fred
jim,jim
......
<more lines, one per user>
......
```

For DBD::CSV on Unix, configure your SQL clause like this. DBUsername and DBAuth are not required. f_dir specifies the directory where the database files are located. csv_eol in this example specifies that the line separators are Unix newlines. You must have this for a Unix style text file. You can leave it off to get Windows standard text files.

```
DBSource    dbi:CSV:f_dir=/your/data/dir;csv_eol=\012
```

19.13. DB2 and other database servers

IBM provides ODBC driver for DB2 which works with Linux and macOS. Follow the installation instructions from IBM and then configure Radiator to use ODBC to access DB2. For more information, see Section 19.8. ODBC on page 529. There is also DBD::DB2 in CPAN which works directly without ODBC.

Other databases should work with Radiator provided that there is a native DBD driver or ODBC driver available for them.

20. Performance and tuning

Radiator has been tuned for maximum performance. You can find some examples of server performance on the Radiator web site. The performance you achieve will depend on many factors including the hardware and the

authentication type. If you find you need to get better performance than you are achieving, you might try the following ideas and suggestions:

- Operate multiple radius servers, and share the load between them. You would normally make each radius server the primary radius server for some of your NASs, and the secondary for a different group of NASs. Consider doing this anyway in order to make your network more robust in the face of a network or server failure. Consider using LOADBALANCE, VOLUMEBALANCE or ROUNDROBIN to distribute the incoming request load among multiple servers).
- Use DBM, SQL, cached FILE or UNIX authentication in preference to any other method.
- If you are using SQL, make sure that your User and Accounting tables have been designed for performance. This will usually mean adding indexes to the tables. Without indexes, selects on large tables can be very slow. A properly designed index will usually speed them up enormously.
- Deploy Radiator on a different (faster) machine. Radiator is highly portable and will run on most Unix hosts, Windows 7/8/8.1/10 and Server 2008/2012/2016/2019.
- Use exact Realm names instead of regexps.
- If you are authenticating multiple realms, consider creating sub-servers, one for each realm, and a main server that uses AuthBy RADIUS to retransmit to the sub-servers.
- Deploy any sub-servers on other machines on the same network as the main server.
- If you are using SQL, deploy the SQL server and Radiator on different hosts.
- If you are using LDAP, deploy the LDAP server and Radiator on different hosts.
- Do not specify RewriteUsername unless you need it.
- If you do not need accounting log files (perhaps you are already getting accounting logged by SQL) turn off *`AcctLogFileName`*.
- If you do not need wtmp log files (perhaps you are already getting accounting logged by SQL or *`AcctLogFileName`*) turn off WtmpFileName.
- Use the lowest Trace level you really need. Higher levels slow radiusd down.
- Use as few check items as possible.
- Do not use Simultaneous-Use check items that specify a filename.
- Do not use check items that are regular expressions.
- Use the Fork parameter if your authentication method is “slow”.
- Do not specify MaxSessions or Simultaneous-Use. If you do, do not specify the Nas-Type in the <Client> clause (interrogating the NAS to confirm when logins are exceeded slows Radiator down).
- Do not use *`PasswordLogFileName`* unless you really need it.
- Do not specify an external Session Database with the *`<SessionDatabase ...>`* clause unless you need Simultaneous-Use limits and you are running multiple instances of Radiator.
- Consider running separate servers, one for accounting and one for authentication.
- If you have large number of concurrent users and require an external session database, consider using SessionDatabase SQL instead of SessionDatabase DBM.

21. Getting help

21.1. Support contract holders

Your Radiator license package may include email support and telephone support. It's also possible to purchase additional support packages, including consulting, training and custom coding. For more information about paid support, please see https://radiatorsoftware.com/support/.

If you have a support contract, you may send email to support@radiatorsoftware.com. Include your support contract identifier in the Subject line. This email address is reserved for support contract holders only. For the detailed information about contacting support, please see https://radiatorsoftware.com/support/

Latest Radiator reference manual and other information is available at https://radiatorsoftware.com/products/radiator/

21.2. No support contract

Latest Radiator reference manual and other information is available at Radiator product pages [https://radiatorsoftware.com/products/radiator/].

All Radiator users are welcome to join the public Radiator mailing list. This means you can work with other Radiator users in the user community. In order to participate with others in this effort, please see Radiator mailing list [https://lists.open.com.au/mailman/listinfo/radiator]. This list is archived at https://lists.open.com.au/pipermail/radiator/

The staff of Radiator Software monitors the Radiator mailing list and frequently answers questions.

21.3. What to do if you need help

When you need help, we suggest you go through the following check list:

1. Consult this reference manual.
2. Consult the public Radiator mailing list archive [https://lists.open.com.au/mailman/listinfo/radiator] for more hints. You can search this archive for items related to your problem.
3. Check that you are using the latest version of Radiator. See Radiator downloads [https://radiatorsoftware.com/downloads/], use the user name and password we have issued to you. See the revision history and upgrade if you need to.
4. Check whether there are any patches that address your problem. See the index file in the patches directory for your release at Radiator downloads [https://radiatorsoftware.com/downloads/]. Apply the patch set if you think you might need it.
5. If you still have the problem and have a support contract, please see Radiator Support pages [https://radiatorsoftware.com/support/] for the next steps. In any case you may also consider joining the public Radiator mailing list. See step 2 above.
6. Professional services for design, installation, configuration and training are available with hourly and daily rates. Send requests to info@radiatorsoftware.com.

This information helps people to understand your problem and help find a solution more quickly.

21.4. Announcements

There is a separate Radiator mailing list that just carries product announcements and upgrades. If you want to know about upgrades available, but do not want all the technical volume from the normal mailing list, this may be the one for you. Radiator product announcements will be posted to both lists.

You can join the Radiator Announcements mailing list by following the instructions on the list information page [https://lists.open.com.au/mailman/listinfo/radiator-announce]. The list archive is also available via the same page.

21.5. Bug reports

We are interested in your feedback, both positive and negative, and bug reports. Please send them to info@radiatorsoftware.com. Radiator download access holders are entitled to free upgrades, and we do fix bugs that are reported to us, so if you report a bug, you can expect to get an upgrade with a fix one day.

21.6. RFCs

A number of relevant RFCs can be found in the doc directory of the Radiator distribution. A full set of RFCs including the ones relevant to RADIUS, RFC 2865 and RFC 2866, can be found at IETF RFC Documents [https://www.rfc-editor.org/].