hURL 2.2

The Multi-Protocol Data Transfer Plugin for Hollywood

Andreas Falkenhahn

Table of Contents

1 General information . . . 1

1.1 Introduction . . . 1
1.2 Terms and conditions . . . 1
1.3 Requirements . . . 2
1.4 Installation . . . 2

2 About hURL . . . 5

2.1 Versions . . . 5
2.2 Credits . . . 5
2.3 Frequently asked questions . . . 5
2.4 Future . . . 6
2.5 History . . . 6

3 Using hURL . . . 7

3.1 Overview . . . 7
3.2 Using the high-level interface . . . 7
3.3 Using the low-level interface . . . 7

4 General functions . . . 11

4.1 hurl.Easy . . . 11
4.2 hurl.Form . . . 11
4.3 hurl.Multi . . . 12
4.4 hurl.Share . . . 12
4.5 hurl.URL . . . 13
4.6 hurl.Version . . . 15
4.7 hurl.VersionInfo . . . 16

5 Easy methods . . . 19

5.1 easy:Close . . . 19
5.2 easy:Escape . . . 19
5.3 easy:GetInfo . . . 20
5.4 easy:GetInfo_AppConnect_Time . . . 25
5.5 easy:GetInfo_AppConnect_Time_t . . . 25
5.6 easy:GetInfo_CAInfo . . . 26
5.7 easy:GetInfo_CAPath . . . 26
5.8 easy:GetInfo_CertInfo . . . 27
5.9 easy:GetInfo_Condition_Unmet . . . 27
5.10 easy:GetInfo_Connect_Time . . . 27
5.11 easy:GetInfo_Connect_Time_t . . . 28
5.12 easy:GetInfo_Content_Length_Download . . . 28
5.13 easy:GetInfo_Content_Length_Download_t . . . 29

5.14 easy:GetInfo_Content_Length_Upload 29
5.15 easy:GetInfo_Content_Length_Upload_t 29
5.16 easy:GetInfo_Content_Type 30
5.17 easy:GetInfo_CookieList 30
5.18 easy:GetInfo_Effective_Method 31
5.19 easy:GetInfo_Effective_URL 31
5.20 easy:GetInfo_FileTime 31
5.21 easy:GetInfo_FTP_Entry_Path 32
5.22 easy:GetInfo_Header_Size 32
5.23 easy:GetInfo_HTTPAuth_Avail 33
5.24 easy:GetInfo_HTTP_ConnectCode 33
5.25 easy:GetInfo_HTTP_Version 33
5.26 easy:GetInfo_LastSocket 34
5.27 easy:GetInfo_Local_IP 34
5.28 easy:GetInfo_Local_Port 35
5.29 easy:GetInfo_NameLookup_Time 35
5.30 easy:GetInfo_NameLookup_Time_t 35
5.31 easy:GetInfo_Num_Connects 36
5.32 easy:GetInfo_OS_ErrNo 36
5.33 easy:GetInfo_PreTransfer_Time 36
5.34 easy:GetInfo_PreTransfer_Time_t 37
5.35 easy:GetInfo_Primary_IP 37
5.36 easy:GetInfo_Primary_Port 38
5.37 easy:GetInfo_Protocol 38
5.38 easy:GetInfo_ProxyAuth_Avail 39
5.39 easy:GetInfo_Proxy_Error 39
5.40 easy:GetInfo_Proxy_SSL_VerifyResult 40
5.41 easy:GetInfo_Redirect_Count 40
5.42 easy:GetInfo_Redirect_Time 40
5.43 easy:GetInfo_Redirect_Time_t 41
5.44 easy:GetInfo_Redirect_URL 41
5.45 easy:GetInfo_Referer 42
5.46 easy:GetInfo_Request_Size 42
5.47 easy:GetInfo_Response_Code 42
5.48 easy:GetInfo_Retry_After 43
5.49 easy:GetInfo_RTSP_Client_CSeq 43
5.50 easy:GetInfo_RTSP_CSeq_Recv 44
5.51 easy:GetInfo_RTSP_Server_CSeq 44
5.52 easy:GetInfo_RTSP_Session_ID 44
5.53 easy:GetInfo_Scheme 45
5.54 easy:GetInfo_Size_Download 45
5.55 easy:GetInfo_Size_Download_t 46
5.56 easy:GetInfo_Size_Upload 46
5.57 easy:GetInfo_Size_Upload_t 46
5.58 easy:GetInfo_Speed_Download 47
5.59 easy:GetInfo_Speed_Download_t 47
5.60 easy:GetInfo_Speed_Upload 48

5.61 easy:GetInfo_Speed_Upload_t 48
5.62 easy:GetInfo_SSL_Engines 48
5.63 easy:GetInfo_SSL_VerifyResult 49
5.64 easy:GetInfo_StartTransfer_Time 49
5.65 easy:GetInfo_StartTransfer_Time_t 50
5.66 easy:GetInfo_Total_Time 50
5.67 easy:GetInfo_Total_Time_t 50
5.68 easy:Mime 51
5.69 easy:Pause 51
5.70 easy:Perform 52
5.71 easy:Recv 53
5.72 easy:Reset 54
5.73 easy:Send 54
5.74 easy:SetOpt 55
5.75 easy:SetOpt_Abstract_Unix_Socket 74
5.76 easy:SetOpt_Accept_Encoding 75
5.77 easy:SetOpt_AcceptTimeout_MS 76
5.78 easy:SetOpt_Address_Scope 76
5.79 easy:SetOpt_AltSvc 77
5.80 easy:SetOpt_AltSvc_Ctrl 77
5.81 easy:SetOpt_Append 78
5.82 easy:SetOpt_AutoReferer 78
5.83 easy:SetOpt_AWS_SigV4 78
5.84 easy:SetOpt_BufferSize 79
5.85 easy:SetOpt_CA_Cache_Timeout 80
5.86 easy:SetOpt_CAInfo 80
5.87 easy:SetOpt_CAInfo_Blob 81
5.88 easy:SetOpt_CAPath 81
5.89 easy:SetOpt_CertInfo 82
5.90 easy:SetOpt_Chunk_BGN_Function 82
5.91 easy:SetOpt_Chunk_End_Function 83
5.92 easy:SetOpt_Connect_Only 84
5.93 easy:SetOpt_ConnectTimeout 84
5.94 easy:SetOpt_ConnectTimeout_MS 84
5.95 easy:SetOpt_Connect_To 85
5.96 easy:SetOpt_Cookie 86
5.97 easy:SetOpt_CookieFile 87
5.98 easy:SetOpt_CookieJar 87
5.99 easy:SetOpt_CookieList 88
5.100 easy:SetOpt_CookieSession 88
5.101 easy:SetOpt_CRLF 89
5.102 easy:SetOpt_CRLFile 89
5.103 easy:SetOpt_CURLU 90
5.104 easy:SetOpt_CustomRequest 90
5.105 easy:SetOpt_DebugFunction 91
5.106 easy:SetOpt_Default_Protocol 92
5.107 easy:SetOpt_DirListOnly 93

5.108 easy:SetOpt_Disallow_Username_In_URL 94
5.109 easy:SetOpt_DNS_Cache_Timeout 94
5.110 easy:SetOpt_DNS_Interface 95
5.111 easy:SetOpt_DNS_Local_IP4 95
5.112 easy:SetOpt_DNS_Local_IP6 95
5.113 easy:SetOpt_DNS_Servers 96
5.114 easy:SetOpt_DNS_Shuffle_Addresses 96
5.115 easy:SetOpt_DNS_Use_Global_Cache 96
5.116 easy:SetOpt_DoH_SSL_VerifyHost 97
5.117 easy:SetOpt_DoH_SSL_VerifyPeer 97
5.118 easy:SetOpt_DoH_SSL_VerifyStatus 98
5.119 easy:SetOpt_DoH_URL 99
5.120 easy:SetOpt_EGDSocket 99
5.121 easy:SetOpt_Expect_100_Timeout_MS 99
5.122 easy:SetOpt_FailOnError 100
5.123 easy:SetOpt_FileTime 100
5.124 easy:SetOpt_FNMatch_Function 101
5.125 easy:SetOpt_FollowLocation 101
5.126 easy:SetOpt_Forbid_Reuse 102
5.127 easy:SetOpt_Fresh_Connect 102
5.128 easy:SetOpt_FTP_Account 103
5.129 easy:SetOpt_FTP_Alternative_To_User 103
5.130 easy:SetOpt_FTP_Create_Missing_Dirs 103
5.131 easy:SetOpt_FTP_FileMethod 104
5.132 easy:SetOpt_FTPPort 105
5.133 easy:SetOpt_FTP_Response_Timeout 105
5.134 easy:SetOpt_FTP_Skip_PASV_IP 106
5.135 easy:SetOpt_FTPSSLAuth 106
5.136 easy:SetOpt_FTP_SSL_CCC 107
5.137 easy:SetOpt_FTP_Use_Eprt 107
5.138 easy:SetOpt_FTP_Use_Epsv 107
5.139 easy:SetOpt_FTP_Use_Pret 108
5.140 easy:SetOpt_GSSAPI_Delegation 108
5.141 easy:SetOpt_Happy_Eyeballs_Timeout_MS 109
5.142 easy:SetOpt_HAProxyProtocol 109
5.143 easy:SetOpt_Header 109
5.144 easy:SetOpt_HeaderFunction 110
5.145 easy:SetOpt_HeaderOpt 111
5.146 easy:SetOpt_HSTS 111
5.147 easy:SetOpt_HSTS_Ctrl 112
5.148 easy:SetOpt_HSTSReadFunction 113
5.149 easy:SetOpt_HSTSWriteFunction 113
5.150 easy:SetOpt_HTTP09_Allowed 114
5.151 easy:SetOpt_HTTP200Aliases 115
5.152 easy:SetOpt_HTTPAuth 115
5.153 easy:SetOpt_HTTP_Content_Decoding 117
5.154 easy:SetOpt_HTTPGet 117

5.155 easy:SetOpt_HTTPHeader 118
5.156 easy:SetOpt_HTTPPost 118
5.157 easy:SetOpt_HTTPProxyTunnel 119
5.158 easy:SetOpt_HTTP_Transfer_Decoding 119
5.159 easy:SetOpt_HTTP_Version 120
5.160 easy:SetOpt_Ignore_Content_Length 121
5.161 easy:SetOpt_InFileSize 121
5.162 easy:SetOpt_InFileSize_Large 122
5.163 easy:SetOpt_Interface 122
5.164 easy:SetOpt_IPResolve 123
5.165 easy:SetOpt_IssuerCert 123
5.166 easy:SetOpt_IssuerCert_Blob 124
5.167 easy:SetOpt_Keep_Sending_On_Error 124
5.168 easy:SetOpt_KeyPasswd 125
5.169 easy:SetOpt_KRBLevel 125
5.170 easy:SetOpt_LocalPort 125
5.171 easy:SetOpt_LocalPortRange 126
5.172 easy:SetOpt_Login_Options 126
5.173 easy:SetOpt_Low_Speed_Limit 126
5.174 easy:SetOpt_Low_Speed_Time 127
5.175 easy:SetOpt_Mail_Auth 127
5.176 easy:SetOpt_Mail_From 128
5.177 easy:SetOpt_Mail_RCPT 128
5.178 easy:SetOpt_Mail_RCPT_AllowFails 128
5.179 easy:SetOpt_MaxAge_Conn 129
5.180 easy:SetOpt_MaxConnects 129
5.181 easy:SetOpt_MaxFileSize 130
5.182 easy:SetOpt_MaxFileSize_Large 130
5.183 easy:SetOpt_MaxLifeTime_Conn 131
5.184 easy:SetOpt_Max_Recv_Speed_Large 131
5.185 easy:SetOpt_MaxRedirs 132
5.186 easy:SetOpt_Max_Send_Speed_Large 132
5.187 easy:SetOpt_MIME_Options 132
5.188 easy:SetOpt_MIMEPost 133
5.189 easy:SetOpt_Netrc 133
5.190 easy:SetOpt_Netrc_File 134
5.191 easy:SetOpt_New_Directory_Perms 134
5.192 easy:SetOpt_New_File_Perms 135
5.193 easy:SetOpt_Nobody 135
5.194 easy:SetOpt_NoProgress 136
5.195 easy:SetOpt_NoProxy 136
5.196 easy:SetOpt_NoSignal 136
5.197 easy:SetOpt_Password 137
5.198 easy:SetOpt_Path_As_Is 137
5.199 easy:SetOpt_PinnedPublicKey 138
5.200 easy:SetOpt_PipeWait 138
5.201 easy:SetOpt_Port 139

5.202 easy:SetOpt_Post 139
5.203 easy:SetOpt_PostFields 140
5.204 easy:SetOpt_PostQuote 141
5.205 easy:SetOpt_PostRedir 141
5.206 easy:SetOpt_Pre_Proxy 142
5.207 easy:SetOpt_Prequote 142
5.208 easy:SetOpt_PreReqFunction 143
5.209 easy:SetOpt_ProgressFunction 144
5.210 easy:SetOpt_Protocols 144
5.211 easy:SetOpt_Protocols_Str 145
5.212 easy:SetOpt_Proxy 147
5.213 easy:SetOpt_ProxyAuth 148
5.214 easy:SetOpt_Proxy_CAInfo 148
5.215 easy:SetOpt_Proxy_CAInfo_Blob 149
5.216 easy:SetOpt_Proxy_CAPath 149
5.217 easy:SetOpt_Proxy_CRLFile 150
5.218 easy:SetOpt_ProxyHeader 150
5.219 easy:SetOpt_Proxy_IssuerCert 151
5.220 easy:SetOpt_Proxy_IssuerCert_Blob 151
5.221 easy:SetOpt_Proxy_KeyPasswd 152
5.222 easy:SetOpt_ProxyPassword 152
5.223 easy:SetOpt_Proxy_PinnedPublicKey 152
5.224 easy:SetOpt_ProxyPort 153
5.225 easy:SetOpt_Proxy_Service_Name 153
5.226 easy:SetOpt_Proxy_SSLCert 154
5.227 easy:SetOpt_Proxy_SSLCert_Blob 154
5.228 easy:SetOpt_Proxy_SSLCertType 154
5.229 easy:SetOpt_Proxy_SSL_Cipher_List 155
5.230 easy:SetOpt_Proxy_SSLKey 155
5.231 easy:SetOpt_Proxy_SSLKey_Blob 156
5.232 easy:SetOpt_Proxy_SSLKeyType 156
5.233 easy:SetOpt_Proxy_SSL_Options 156
5.234 easy:SetOpt_Proxy_SSL_VerifyHost 157
5.235 easy:SetOpt_Proxy_SSL_VerifyPeer 158
5.236 easy:SetOpt_Proxy_SSLVersion 158
5.237 easy:SetOpt_Proxy_TLSAuth_Password 160
5.238 easy:SetOpt_Proxy_TLSAuth_Type 160
5.239 easy:SetOpt_Proxy_TLSAuth_UserName 160
5.240 easy:SetOpt_Proxy_Transfer_Mode 161
5.241 easy:SetOpt_ProxyType 161
5.242 easy:SetOpt_ProxyUserName 162
5.243 easy:SetOpt_ProxyUserPwd 162
5.244 easy:SetOpt_Put 163
5.245 easy:SetOpt_Quick_Exit 163
5.246 easy:SetOpt_Quote 163
5.247 easy:SetOpt_Random_File 165
5.248 easy:SetOpt_Range 165

5.249 easy:SetOpt_ReadFunction 165
5.250 easy:SetOpt_Redir_Protocols 167
5.251 easy:SetOpt_Redir_Protocols_Str 168
5.252 easy:SetOpt_Referer 169
5.253 easy:SetOpt_Request_Target 169
5.254 easy:SetOpt_Resolve 170
5.255 easy:SetOpt_Resolver_Start_Function 170
5.256 easy:SetOpt_Resume_From 171
5.257 easy:SetOpt_Resume_From_Large 171
5.258 easy:SetOpt_RTSP_Client_CSeq 172
5.259 easy:SetOpt_RTSP_Request 172
5.260 easy:SetOpt_RTSP_Server_CSeq 174
5.261 easy:SetOpt_RTSP_Session_ID 174
5.262 easy:SetOpt_RTSP_Stream_URI 174
5.263 easy:SetOpt_RTSP_Transport 175
5.264 easy:SetOpt_SASL_AuthZID 175
5.265 easy:SetOpt_SASL_IR 176
5.266 easy:SetOpt_SeekFunction 176
5.267 easy:SetOpt_Service_Name 177
5.268 easy:SetOpt_Share 177
5.269 easy:SetOpt_Socks5_Auth 178
5.270 easy:SetOpt_Socks5_GSSAPI_NEC 178
5.271 easy:SetOpt_Socks5_GSSAPI_Service 179
5.272 easy:SetOpt_SSH_Auth_Types 179
5.273 easy:SetOpt_SSH_Compression 179
5.274 easy:SetOpt_SSH_HostKeyFunction 180
5.275 easy:SetOpt_SSH_Host_Public_Key_MD5 181
5.276 easy:SetOpt_SSH_KnownHosts 181
5.277 easy:SetOpt_SSH_Private_KeyFile 181
5.278 easy:SetOpt_SSH_Public_KeyFile 182
5.279 easy:SetOpt_SSLCert 182
5.280 easy:SetOpt_SSLCert_Blob 183
5.281 easy:SetOpt_SSLCertType 183
5.282 easy:SetOpt_SSL_Cipher_List 183
5.283 easy:SetOpt_SSL_EC_Curves 184
5.284 easy:SetOpt_SSL_Enable_Alpn 184
5.285 easy:SetOpt_SSL_Enable_Npn 185
5.286 easy:SetOpt_SSLEngine 185
5.287 easy:SetOpt_SSLEngine_Default 185
5.288 easy:SetOpt_SSL_FalseStart 186
5.289 easy:SetOpt_SSLKey 186
5.290 easy:SetOpt_SSLKey_Blob 186
5.291 easy:SetOpt_SSLKeyType 187
5.292 easy:SetOpt_SSL_Options 187
5.293 easy:SetOpt_SSL_SessionID_Cache 188
5.294 easy:SetOpt_SSL_VerifyHost 188
5.295 easy:SetOpt_SSL_VerifyPeer 189

5.296 easy:SetOpt_SSL_VerifyStatus . . . 190
5.297 easy:SetOpt_SSLVersion . . . 190
5.298 easy:SetOpt_Stream_Depends . . . 191
5.299 easy:SetOpt_Stream_Depends_e . . . 192
5.300 easy:SetOpt_Stream_Weight . . . 192
5.301 easy:SetOpt_Suppress_Connect_Headers . . . 193
5.302 easy:SetOpt_TCP_FastOpen . . . 194
5.303 easy:SetOpt_TCP_KeepAlive . . . 194
5.304 easy:SetOpt_TCP_KeepIdle . . . 194
5.305 easy:SetOpt_TCP_KeepIntvl . . . 195
5.306 easy:SetOpt_TCP_NoDelay . . . 195
5.307 easy:SetOpt_TelnetOptions . . . 196
5.308 easy:SetOpt_TFTP_BlkSize . . . 196
5.309 easy:SetOpt_TFTP_No_Options . . . 196
5.310 easy:SetOpt_TimeCondition . . . 197
5.311 easy:SetOpt_Timeout . . . 197
5.312 easy:SetOpt_Timeout_MS . . . 198
5.313 easy:SetOpt_TimeValue . . . 198
5.314 easy:SetOpt_TimeValue_Large . . . 199
5.315 easy:SetOpt_TLS13_Ciphers . . . 199
5.316 easy:SetOpt_TLSAuth_Password . . . 200
5.317 easy:SetOpt_TLSAuth_Type . . . 200
5.318 easy:SetOpt_TLSAuth_UserName . . . 200
5.319 easy:SetOpt_TrailerFunction . . . 201
5.320 easy:SetOpt_Transfer_Encoding . . . 201
5.321 easy:SetOpt_TransferText . . . 202
5.322 easy:SetOpt_Unix_Socket_Path . . . 202
5.323 easy:SetOpt_Unrestricted_Auth . . . 203
5.324 easy:SetOpt_Upkeep_Interval_MS . . . 203
5.325 easy:SetOpt_Upload . . . 204
5.326 easy:SetOpt_Upload_Buffersize . . . 204
5.327 easy:SetOpt_URL . . . 205
5.328 easy:SetOpt_UserAgent . . . 209
5.329 easy:SetOpt_UserName . . . 209
5.330 easy:SetOpt_UserPwd . . . 210
5.331 easy:SetOpt_Use_SSL . . . 211
5.332 easy:SetOpt_Verbose . . . 211
5.333 easy:SetOpt_WildcardMatch . . . 212
5.334 easy:SetOpt_WriteFunction . . . 213
5.335 easy:SetOpt_WS_Options . . . 214
5.336 easy:SetOpt_XOAuth2_Bearer . . . 214
5.337 easy:Unescape . . . 214
5.338 easy:UnsetOpt . . . 215
5.339 easy:UnsetOpt_Abstract_Unix_Socket . . . 234
5.340 easy:UnsetOpt_Accept_Encoding . . . 235
5.341 easy:UnsetOpt_AcceptTimeout_MS . . . 235
5.342 easy:UnsetOpt_Address_Scope . . . 235

5.343 easy:UnsetOpt_AltSvc 236
5.344 easy:UnsetOpt_AltSvc_Ctrl 236
5.345 easy:UnsetOpt_Append 236
5.346 easy:UnsetOpt_AutoReferer 236
5.347 easy:UnsetOpt_AWS_SigV4 237
5.348 easy:UnsetOpt_BufferSize 237
5.349 easy:UnsetOpt_CA_Cache_Timeout 237
5.350 easy:UnsetOpt_CAInfo 238
5.351 easy:UnsetOpt_CAInfo_Blob 238
5.352 easy:UnsetOpt_CAPath 238
5.353 easy:UnsetOpt_CertInfo 238
5.354 easy:UnsetOpt_Chunk_BGN_Function 239
5.355 easy:UnsetOpt_Chunk_End_Function 239
5.356 easy:UnsetOpt_Connect_Only 239
5.357 easy:UnsetOpt_ConnectTimeout 240
5.358 easy:UnsetOpt_ConnectTimeout_MS 240
5.359 easy:UnsetOpt_Connect_To 240
5.360 easy:UnsetOpt_Cookie 240
5.361 easy:UnsetOpt_CookieFile 241
5.362 easy:UnsetOpt_CookieJar 241
5.363 easy:UnsetOpt_CookieList 241
5.364 easy:UnsetOpt_CookieSession 242
5.365 easy:UnsetOpt_CRLF 242
5.366 easy:UnsetOpt_CRLFile 242
5.367 easy:UnsetOpt_CURLU 242
5.368 easy:UnsetOpt_CustomRequest 243
5.369 easy:UnsetOpt_DebugFunction 243
5.370 easy:UnsetOpt_Default_Protocol 243
5.371 easy:UnsetOpt_DirListOnly 244
5.372 easy:UnsetOpt_Disallow_Username_In_URL 244
5.373 easy:UnsetOpt_DNS_Cache_Timeout 244
5.374 easy:UnsetOpt_DNS_Interface 244
5.375 easy:UnsetOpt_DNS_Local_IP4 245
5.376 easy:UnsetOpt_DNS_Local_IP6 245
5.377 easy:UnsetOpt_DNS_Servers 245
5.378 easy:UnsetOpt_DNS_Shuffle_Addresses 246
5.379 easy:UnsetOpt_DNS_Use_Global_Cache 246
5.380 easy:UnsetOpt_DoH_SSL_VerifyHost 246
5.381 easy:UnsetOpt_DoH_SSL_VerifyPeer 246
5.382 easy:UnsetOpt_DoH_SSL_VerifyStatus 247
5.383 easy:UnsetOpt_DoH_URL 247
5.384 easy:UnsetOpt_EGDSocket 247
5.385 easy:UnsetOpt_Expect_100_Timeout_MS 248
5.386 easy:UnsetOpt_FailOnError 248
5.387 easy:UnsetOpt_FileTime 248
5.388 easy:UnsetOpt_FNMatch_Function 248
5.389 easy:UnsetOpt_FollowLocation 249

5.390 easy:UnsetOpt_Forbid_Reuse 249
5.391 easy:UnsetOpt_Fresh_Connect 249
5.392 easy:UnsetOpt_FTP_Account 250
5.393 easy:UnsetOpt_FTP_Alternative_To_User 250
5.394 easy:UnsetOpt_FTP_Create_Missing_Dirs 250
5.395 easy:UnsetOpt_FTP_FileMethod 250
5.396 easy:UnsetOpt_FTPPort 251
5.397 easy:UnsetOpt_FTP_Response_Timeout 251
5.398 easy:UnsetOpt_FTP_Skip_PASV_IP 251
5.399 easy:UnsetOpt_FTPSSLAuth 252
5.400 easy:UnsetOpt_FTP_SSL_CCC 252
5.401 easy:UnsetOpt_FTP_Use_Eprt 252
5.402 easy:UnsetOpt_FTP_Use_Epsv 252
5.403 easy:UnsetOpt_FTP_Use_Pret 253
5.404 easy:UnsetOpt_GSSAPI_Delegation 253
5.405 easy:UnsetOpt_Happy_Eyeballs_Timeout_MS 253
5.406 easy:UnsetOpt_HAProxyProtocol 254
5.407 easy:UnsetOpt_Header 254
5.408 easy:UnsetOpt_HeaderFunction 254
5.409 easy:UnsetOpt_HeaderOpt 254
5.410 easy:UnsetOpt_HSTS 255
5.411 easy:UnsetOpt_HSTS_Ctrl 255
5.412 easy:UnsetOpt_HSTSReadFunction 255
5.413 easy:UnsetOpt_HSTSWriteFunction 256
5.414 easy:UnsetOpt_HTTP09_Allowed 256
5.415 easy:UnsetOpt_HTTP200Aliases 256
5.416 easy:UnsetOpt_HTTPAuth 256
5.417 easy:UnsetOpt_HTTP_Content_Decoding 257
5.418 easy:UnsetOpt_HTTPGet 257
5.419 easy:UnsetOpt_HTTPHeader 257
5.420 easy:UnsetOpt_HTTPPost 258
5.421 easy:UnsetOpt_HTTPProxyTunnel 258
5.422 easy:UnsetOpt_HTTP_Transfer_Decoding 258
5.423 easy:UnsetOpt_HTTP_Version 258
5.424 easy:UnsetOpt_Ignore_Content_Length 259
5.425 easy:UnsetOpt_InFileSize 259
5.426 easy:UnsetOpt_InFileSize_Large 259
5.427 easy:UnsetOpt_Interface 260
5.428 easy:UnsetOpt_IPResolve 260
5.429 easy:UnsetOpt_IssuerCert 260
5.430 easy:UnsetOpt_IssuerCert_Blob 260
5.431 easy:UnsetOpt_Keep_Sending_On_Error 261
5.432 easy:UnsetOpt_KeyPasswd 261
5.433 easy:UnsetOpt_KRBLevel 261
5.434 easy:UnsetOpt_LocalPort 262
5.435 easy:UnsetOpt_LocalPortRange 262
5.436 easy:UnsetOpt_Login_Options 262

5.437 easy:UnsetOpt_Low_Speed_Limit 262
5.438 easy:UnsetOpt_Low_Speed_Time 263
5.439 easy:UnsetOpt_Mail_Auth 263
5.440 easy:UnsetOpt_Mail_From 263
5.441 easy:UnsetOpt_Mail_RCPT 264
5.442 easy:UnsetOpt_Mail_RCPT_AllowFails 264
5.443 easy:UnsetOpt_MaxAge_Conn 264
5.444 easy:UnsetOpt_MaxConnects 264
5.445 easy:UnsetOpt_MaxFileSize 265
5.446 easy:UnsetOpt_MaxFileSize_Large 265
5.447 easy:UnsetOpt_MaxLifeTime_Conn 265
5.448 easy:UnsetOpt_Max_Recv_Speed_Large 266
5.449 easy:UnsetOpt_MaxRedirs 266
5.450 easy:UnsetOpt_Max_Send_Speed_Large 266
5.451 easy:UnsetOpt_MIME_Options 266
5.452 easy:UnsetOpt_MIMEPost 267
5.453 easy:UnsetOpt_Netrc 267
5.454 easy:UnsetOpt_Netrc_File 267
5.455 easy:UnsetOpt_New_Directory_Perms 268
5.456 easy:UnsetOpt_New_File_Perms 268
5.457 easy:UnsetOpt_Nobody 268
5.458 easy:UnsetOpt_NoProgress 268
5.459 easy:UnsetOpt_NoProxy 269
5.460 easy:UnsetOpt_NoSignal 269
5.461 easy:UnsetOpt_Password 269
5.462 easy:UnsetOpt_Path_As_Is 270
5.463 easy:UnsetOpt_PinnedPublicKey 270
5.464 easy:UnsetOpt_PipeWait 270
5.465 easy:UnsetOpt_Port 270
5.466 easy:UnsetOpt_Post 271
5.467 easy:UnsetOpt_PostFields 271
5.468 easy:UnsetOpt_PostQuote 271
5.469 easy:UnsetOpt_PostRedir 272
5.470 easy:UnsetOpt_Pre_Proxy 272
5.471 easy:UnsetOpt_Prequote 272
5.472 easy:UnsetOpt_PreReqFunction 272
5.473 easy:UnsetOpt_ProgressFunction 273
5.474 easy:UnsetOpt_Protocols 273
5.475 easy:UnsetOpt_Protocols_Str 273
5.476 easy:UnsetOpt_Proxy 274
5.477 easy:UnsetOpt_ProxyAuth 274
5.478 easy:UnsetOpt_Proxy_CAInfo 274
5.479 easy:UnsetOpt_Proxy_CAInfo_Blob 274
5.480 easy:UnsetOpt_Proxy_CAPath 275
5.481 easy:UnsetOpt_Proxy_CRLFile 275
5.482 easy:UnsetOpt_ProxyHeader 275
5.483 easy:UnsetOpt_Proxy_IssuerCert 276

5.484 easy:UnsetOpt_Proxy_IssuerCert_Blob 276
5.485 easy:UnsetOpt_Proxy_KeyPasswd 276
5.486 easy:UnsetOpt_ProxyPassword 276
5.487 easy:UnsetOpt_Proxy_PinnedPublicKey 277
5.488 easy:UnsetOpt_ProxyPort 277
5.489 easy:UnsetOpt_Proxy_Service_Name 277
5.490 easy:UnsetOpt_Proxy_SSLCert 278
5.491 easy:UnsetOpt_Proxy_SSLCert_Blob 278
5.492 easy:UnsetOpt_Proxy_SSLCertType 278
5.493 easy:UnsetOpt_Proxy_SSL_Cipher_List 278
5.494 easy:UnsetOpt_Proxy_SSLKey 279
5.495 easy:UnsetOpt_Proxy_SSLKey_Blob 279
5.496 easy:UnsetOpt_Proxy_SSLKeyType 279
5.497 easy:UnsetOpt_Proxy_SSL_Options 280
5.498 easy:UnsetOpt_Proxy_SSL_VerifyHost 280
5.499 easy:UnsetOpt_Proxy_SSL_VerifyPeer 280
5.500 easy:UnsetOpt_Proxy_SSLVersion 280
5.501 easy:UnsetOpt_Proxy_TLSAuth_Password 281
5.502 easy:UnsetOpt_Proxy_TLSAuth_Type 281
5.503 easy:UnsetOpt_Proxy_TLSAuth_UserName 281
5.504 easy:UnsetOpt_Proxy_Transfer_Mode 282
5.505 easy:UnsetOpt_ProxyType 282
5.506 easy:UnsetOpt_ProxyUserName 282
5.507 easy:UnsetOpt_ProxyUserPwd 282
5.508 easy:UnsetOpt_Put 283
5.509 easy:UnsetOpt_Quick_Exit 283
5.510 easy:UnsetOpt_Quote 283
5.511 easy:UnsetOpt_Random_File 284
5.512 easy:UnsetOpt_Range 284
5.513 easy:UnsetOpt_ReadFunction 284
5.514 easy:UnsetOpt_Redir_Protocols 284
5.515 easy:UnsetOpt_Redir_Protocols_Str 285
5.516 easy:UnsetOpt_Referer 285
5.517 easy:UnsetOpt_Request_Target 285
5.518 easy:UnsetOpt_Resolve 286
5.519 easy:UnsetOpt_Resolver_Start_Function 286
5.520 easy:UnsetOpt_Resume_From 286
5.521 easy:UnsetOpt_Resume_From_Large 286
5.522 easy:UnsetOpt_RTSP_Client_CSeq 287
5.523 easy:UnsetOpt_RTSP_Request 287
5.524 easy:UnsetOpt_RTSP_Server_CSeq 287
5.525 easy:UnsetOpt_RTSP_Session_ID 288
5.526 easy:UnsetOpt_RTSP_Stream_URI 288
5.527 easy:UnsetOpt_RTSP_Transport 288
5.528 easy:UnsetOpt_SASL_AuthZID 288
5.529 easy:UnsetOpt_SASL_IR 289
5.530 easy:UnsetOpt_SeekFunction 289

5.531 easy:UnsetOpt_Service_Name 289
5.532 easy:UnsetOpt_Share 290
5.533 easy:UnsetOpt_Socks5_Auth 290
5.534 easy:UnsetOpt_Socks5_GSSAPI_NEC 290
5.535 easy:UnsetOpt_Socks5_GSSAPI_Service 290
5.536 easy:UnsetOpt_SSH_Auth_Types 291
5.537 easy:UnsetOpt_SSH_Compression 291
5.538 easy:UnsetOpt_SSH_HostKeyFunction 291
5.539 easy:UnsetOpt_SSH_Host_Public_Key_MD5 292
5.540 easy:UnsetOpt_SSH_KnownHosts 292
5.541 easy:UnsetOpt_SSH_Private_KeyFile 292
5.542 easy:UnsetOpt_SSH_Public_KeyFile 292
5.543 easy:UnsetOpt_SSLCert 293
5.544 easy:UnsetOpt_SSLCert_Blob 293
5.545 easy:UnsetOpt_SSLCertType 293
5.546 easy:UnsetOpt_SSL_Cipher_List 294
5.547 easy:UnsetOpt_SSL_EC_Curves 294
5.548 easy:UnsetOpt_SSL_Enable_Alpn 294
5.549 easy:UnsetOpt_SSL_Enable_Npn 294
5.550 easy:UnsetOpt_SSLEngine 295
5.551 easy:UnsetOpt_SSLEngine_Default 295
5.552 easy:UnsetOpt_SSL_FalseStart 295
5.553 easy:UnsetOpt_SSLKey 296
5.554 easy:UnsetOpt_SSLKey_Blob 296
5.555 easy:UnsetOpt_SSLKeyType 296
5.556 easy:UnsetOpt_SSL_Options 296
5.557 easy:UnsetOpt_SSL_SessionID_Cache 297
5.558 easy:UnsetOpt_SSL_VerifyHost 297
5.559 easy:UnsetOpt_SSL_VerifyPeer 297
5.560 easy:UnsetOpt_SSL_VerifyStatus 298
5.561 easy:UnsetOpt_SSLVersion 298
5.562 easy:UnsetOpt_Stream_Depends 298
5.563 easy:UnsetOpt_Stream_Depends_e 298
5.564 easy:UnsetOpt_Stream_Weight 299
5.565 easy:UnsetOpt_Suppress_Connect_Headers 299
5.566 easy:UnsetOpt_TCP_FastOpen 299
5.567 easy:UnsetOpt_TCP_KeepAlive 300
5.568 easy:UnsetOpt_TCP_KeepIdle 300
5.569 easy:UnsetOpt_TCP_KeepIntvl 300
5.570 easy:UnsetOpt_TCP_NoDelay 300
5.571 easy:UnsetOpt_TelnetOptions 301
5.572 easy:UnsetOpt_TFTP_BlkSize 301
5.573 easy:UnsetOpt_TFTP_No_Options 301
5.574 easy:UnsetOpt_TimeCondition 302
5.575 easy:UnsetOpt_Timeout 302
5.576 easy:UnsetOpt_Timeout_MS 302
5.577 easy:UnsetOpt_TimeValue 302

5.578 easy:UnsetOpt_TimeValue_Large . . . 303
5.579 easy:UnsetOpt_TLS13_Ciphers . . . 303
5.580 easy:UnsetOpt_TLSAuth_Password . . . 303
5.581 easy:UnsetOpt_TLSAuth_Type . . . 304
5.582 easy:UnsetOpt_TLSAuth_UserName . . . 304
5.583 easy:UnsetOpt_TrailerFunction . . . 304
5.584 easy:UnsetOpt_Transfer_Encoding . . . 304
5.585 easy:UnsetOpt_TransferText . . . 305
5.586 easy:UnsetOpt_Unix_Socket_Path . . . 305
5.587 easy:UnsetOpt_Unrestricted_Auth . . . 305
5.588 easy:UnsetOpt_Upkeep_Interval_MS . . . 306
5.589 easy:UnsetOpt_Upload . . . 306
5.590 easy:UnsetOpt_Upload_Buffersize . . . 306
5.591 easy:UnsetOpt_URL . . . 306
5.592 easy:UnsetOpt_UserAgent . . . 307
5.593 easy:UnsetOpt_UserName . . . 307
5.594 easy:UnsetOpt_UserPwd . . . 307
5.595 easy:UnsetOpt_Use_SSL . . . 308
5.596 easy:UnsetOpt_Verbose . . . 308
5.597 easy:UnsetOpt_WildcardMatch . . . 308
5.598 easy:UnsetOpt_WriteFunction . . . 308
5.599 easy:UnsetOpt_WS_Options . . . 309
5.600 easy:UnsetOpt_XOAuth2_Bearer . . . 309
5.601 easy:Upkeep . . . 309

6 Form methods . . . 311

6.1 form:AddBuffer . . . 311
6.2 form:AddContent . . . 311
6.3 form:AddFile . . . 312
6.4 form:AddFiles . . . 313
6.5 form:AddStream . . . 314
6.6 form:Free . . . 315
6.7 form:Get . . . 315

7 Mime methods . . . 317

7.1 mime:AddPart . . . 317
7.2 mime:Easy . . . 318
7.3 mime:Free . . . 318

8 Mime part methods . . . 319

8.1 mimepart:Data . . . 319
8.2 mimepart:Encoder . . . 319
8.3 mimepart:FileData . . . 320
8.4 mimepart:Filename . . . 320
8.5 mimepart:Free . . . 321

8.6 mimepart:Headers . . . 321
8.7 mimepart:Name . . . 321
8.8 mimepart:Subparts . . . 322
8.9 mimepart:Type . . . 322

9 Multi methods . . . 325

9.1 multi:AddHandle . . . 325
9.2 multi:Close . . . 325
9.3 multi:InfoRead . . . 326
9.4 multi:Perform . . . 327
9.5 multi:RemoveHandle . . . 327
9.6 multi:SetOpt . . . 328
9.7 multi:SetOpt_Chunk_Length_Penalty_Size . . . 329
9.8 multi:SetOpt_Content_Length_Penalty_Size . . . 329
9.9 multi:SetOpt_Max_Concurrent_Streams . . . 330
9.10 multi:SetOpt_MaxConnects . . . 330
9.11 multi:SetOpt_Max_Host_Connections . . . 331
9.12 multi:SetOpt_Max_Pipeline_Length . . . 331
9.13 multi:SetOpt_Max_Total_Connections . . . 331
9.14 multi:SetOpt_Pipelining . . . 332
9.15 multi:SetOpt_Pipelining_Server_Bl . . . 333
9.16 multi:SetOpt_Pipelining_Site_Bl . . . 333
9.17 multi:SetOpt_SocketFunction . . . 334
9.18 multi:SetOpt_TimerFunction . . . 334
9.19 multi:SocketAction . . . 335
9.20 multi:Timeout . . . 336
9.21 multi:Wait . . . 337

10 Share methods . . . 339

10.1 share:Close . . . 339
10.2 share:SetOpt . . . 339
10.3 share:SetOpt_Share . . . 339
10.4 share:SetOpt_Unshare . . . 340

11 URL methods . . . 343

11.1 url:Dup . . . 343
11.2 url:Free . . . 343
11.3 url:GetFragment . . . 343
11.4 url:GetHost . . . 344
11.5 url:GetOptions . . . 344
11.6 url:GetPassword . . . 345
11.7 url:GetPath . . . 345
11.8 url:GetPort . . . 345
11.9 url:GetQuery . . . 346
11.10 url:GetScheme . . . 346

11.11 url:GetURL 347
11.12 url:GetUser 347
11.13 url:GetZoneID 347
11.14 url:SetFragment 348
11.15 url:SetHost 348
11.16 url:SetOptions 349
11.17 url:SetPassword 349
11.18 url:SetPath 350
11.19 url:SetPort 350
11.20 url:SetQuery 350
11.21 url:SetScheme 351
11.22 url:SetURL 351
11.23 url:SetUser 352
11.24 url:SetZoneID 352

Appendix A Licenses 353

A.1 Curl license 353
A.2 LuaCurl license 353
A.3 OpenSSL license 353
A.4 libssh2 license 355

Index 357

1 General information

1.1 Introduction

hURL is a plugin for Hollywood that allows you to transfer data using many different protocols. Based on curl, hURL supports an incredibly wide range of transfer protocols, e.g. DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, Telnet and TFTP. Furthermore, hURL supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, HTTP/2, cookies, user+password authentication (Basic, Plain, Digest, CRAM-MD5, NTLM, Negotiate and Kerberos), file transfer resume, proxy tunneling and more. It really is the ultimate data transfer engine for Hollywood, leaving nothing to be desired.

There are two ways of using hURL: There is a high-level interface that can directly hook itself into Hollywood's network library, enhancing it with hURL functionality like SSL/TLS support. This makes it possible to use Hollywood commands like `DownloadFile()` to download files using custom protocols that Hollywood itself doesn't support, e.g. SSL/TLS.

Another way of using hURL is the low-level interface: This interface allows you to access the curl API directly from Hollywood scripts. This is extremely powerful because it allows you to access hundreds of different curl options, making it possible to fine-tune hURL to your specific needs. hURL contains over 500 commands to fulfil all your data transfer needs!

Finally, hURL comes with extensive documentation in various formats like PDF, HTML, AmigaGuide, and CHM that contains detailed descriptions about all functions and methods offered by the plugin.

All of this makes hURL the ultimate data transfer tool for Hollywood that contains everything you need to send and receive data via almost any transfer protocol on the planet.

1.2 Terms and conditions

hURL is © Copyright 2018-2026 by Andreas Falkenhahn (in the following referred to as "the author"). All rights reserved.

The program is provided "as-is" and the author cannot be made responsible of any possible harm done by it. You are using this program absolutely at your own risk. No warranties are implied or given by the author.

This plugin may be freely distributed as long as the following three conditions are met:

1. No modifications must be made to the plugin.
2. It is not allowed to sell this plugin.
3. If you want to put this plugin on a coverdisc, you need to ask for permission first.

This software uses curl by Daniel Stenberg. See Section A.1 [curl license], page 353, for details.

This software uses Lua-cURL by Alexey Melnichuk. See Section A.2 [Lua-cURL], page 353, for details.

This software uses libssh2. See Section A.4 [libssh2 license], page 355, for details.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) See Section A.3 [OpenSSL license], page 353, for details.

Target icon in hURL logo made by Vectors Market from www.flaticon.com is licensed by CC 3.0 BY.

All trademarks are the property of their respective owners.

DISCLAIMER: THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDER AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 Requirements

- Hollywood 8.0 or better; note that it's recommended to use at least Hollywood 9.0 with hURL because only Hollywood 9.0 (or better) allows you to take full advantage of hURL's high-level interface
- macOS (native SSL): 11.0 on arm64 systems and 10.8 on x86 and x64 systems is required
- macOS (OpenSSL): 11.0 on arm64 systems, 10.13 on x64 and 10.5 on ppc is required
- Android: at least Android 6.0 is required
- AmigaOS 3 and AmigaOS 4: AmiSSL 5 is required
- AROS: AmiSSL 4 is required
- MorphOS: at least MorphOS 3.16 is required

1.4 Installation

Installing hURL is straightforward and simple: Just copy the file `hurl.hwp` for the platform of your choice to Hollywood's plugins directory. The location where Hollywood plugins must be installed differs from platform to platform. Here is a breakdown of the different search paths that Hollywood will scan for plugins:

- AmigaOS and compatibles: Plugins can either be stored in a directory named `Plugins` that is in the same directory as the Hollywood interpreter or they can be installed

globally in `LIBS:Hollywood`. Hollywood will first look for plugins in the `Plugins` directory in its program directory, then it will scan `LIBS:Hollywood`.

- Android: The Android version of Hollywood also supports Hollywood plugins. You have to copy them to the directory `Hollywood/Plugins` on your SD card. Hollywood will scan this location on every startup and load all plugins from there.
- Linux: Plugins must be stored in a directory named `Plugins` (case sensitive!) that is in the same directory as the Hollywood interpreter.
- macOS: `~/Library/Application Support/AirsoftSoftwair/Hollywood/Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins inside Hollywood's app bundle in `Hollywood.app/Contents/Resources/Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges.
- Windows: `C:\Users\...\AppData\Local\AirsoftSoftwair\Hollywood\Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins in the `Plugins` directory in Hollywood's installation directory, e.g. in `C:\Program Files\Hollywood\Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges to write to Hollywood's installation directory.

On Windows you should also copy the file `hURL.chm` to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above). After you have copied `hURL.chm` to the `Docs` directory, you will be able to get online help by pressing F1 when the cursor is over a hURL function in the Hollywood IDE.

On macOS copy the `hURL` directory that is inside the `Docs` directory of hURL's distribution archive to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above).

On Linux copy the `hURL` directory that is inside the `Docs` directory of hURL's distribution archive to the `Docs` directory of your Hollywood installation.

2 About hURL

2.1 Versions

For certain architectures on Linux, macOS and Windows hURL comes in two different flavours: A version that uses the host operating system's SSL stack and a version that contains a statically linked version of OpenSSL.

Both versions have their advantages and disadvantages. hURL versions that use the host operating system's SSL stack are of course much smaller in size than the ones that have OpenSSL statically linked to them which are usually more than 5 megabytes in size because of the size of OpenSSL's cryptography libraries.

On the other hand, the versions of hURL that have OpenSSL statically linked often offer a greater amount of compatibility, especially on older systems where the SSL stack of the host operating system is often out of date. For example, if you plan to use hURL on Windows 7 or 8, it's advised to use the OpenSSL version of hURL because the host operating system's SSL stack is very outdated and won't be able to do secure handshakes with modern serves any more.

Also, on macOS the hURL version for the host operating system doesn't support all protocols that are supported by the OpenSSL version of hURL. For example, the SFTP protocol isn't supported by the non-OpenSSL version of hURL on macOS so if you want to use SFTP on macOS, you must use the OpenSSL version of hURL.

On Linux there is the problem that your installed OpenSSL version might not match the OpenSSL version hURL was built for. In that case, using the hURL version that contains a statically linked version of OpenSSL can also be a helpful alternative. On Linux, the hURL versions that use the host operating system's SSL stack were built for these OpenSSL versions: OpenSSL 3 (x64 and arm64), OpenSSL 1.1.1 (x86 and arm) and OpenSSL 1.0 (ppc). If these versions don't match the OpenSSL version on your Linux system, use the hURL version that has OpenSSL statically linked.

In case you aren't sure which version to use: A good idea is to first try the version that uses the host operating system's SSL stack. If that doesn't work, try the OpenSSL version instead.

2.2 Credits

hURL was written by Andreas Falkenhahn, based on work done by Alexey Melnichuk and Daniel Stenberg.

If you need to contact me, you can either send an e-mail to `andreas@airsoftsoftwair.de` or use the contact form on `http://www.hollywood-mal.com`.

2.3 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the forum because your problem might have been covered here.

Q: Is there a Hollywood forum where I can get in touch with other users?

A: Yes, please check out the "Community" section of the official Hollywood Portal online at http://www.hollywood-mal.com.

Q: Where can I ask for help?

A: There's an active forum at http://forums.hollywood-mal.com. You're welcome to join it and ask your question there.

Q: I have found a bug.

A: Please post about it in the "Bugs" section of the forum.

2.4 Future

Here are some things that are on my to do list:

- add more examples

Don't hesitate to contact me if hURL lacks a certain feature that is important for your project.

2.5 History

Please see the file `history.txt` for a complete change log of hURL.

3 Using hURL

3.1 Overview

There are two different ways of using hURL: You can either access the curl API directly through a low-level interface or you can use hURL's high-level interface which maps some of curl's features to standard Hollywood functions.

Using the high-level interface is really easy and extends Hollywood functions like `DownloadFile()` or `UploadFile()` to operate through curl, enabling them to use SSL/TLS for example. If you just want to download or upload files from/to HTTP(S) and you don't need any fine-tuned control over how the transfer is done, the high-level interface is the way to go for you.

The low-level interface, i.e. accessing curl's API directly, is useful if you need more fine-tuned control over the transfer. The low-level interface allows you to configure all kinds of options in hURL and makes it possible to access all of curl's advanced features, allowing you to meticulously take control over how transfers are managed.

3.2 Using the high-level interface

Using hURL's high-level interface is really easy. It is mostly used to extend Hollywood's `DownloadFile()` and `UploadFile()` commands to support SSL/TLS connections, which Hollywood itself doesn't support. To download a file using an SSL/TLS connection with hURL through the high-level interface, just do the following:

```
@REQUIRE "hurl"
url$ = "https://www.paypal.com/"
DownloadFile(url$, {File = "index.html", Adapter = "hurl"})
```

The code above will download the main page of https://www.paypal.com/ and save it as `index.html`.

By passing `hurl` in the `Adapter` tag you tell `DownloadFile()` to let hURL handle the download. The same is possible with `UploadFile()` and `OpenConnection()`. If you get the `Adapter` tag to `hurl` for those functions, the connection will automatically be managed by hURL, allowing you to use SSL/TLS encryption, for example.

Hollywood's `DownloadFile()`, `UploadFile()`, and `OpenConnection()` functions also have an `SSL` tag which you can get to `True` to tell hURL to enforce a connection via SSL/TLS. This is normally not necessary when passing schemes like `https://` or `ftps://` but can be useful for custom connections.

3.3 Using the low-level interface

Using hURL's low-level interface is more difficult than using the high-level interface because it allows you to access curl's APIs directly. This means that you should first make yourself familiar with curl's API so that you know how it is designed and how it can serve your purposes.

Basically, using a curl API directly involves the following three steps:

1. Create a curl object handle, e.g. a curl easy handle.

2. Do something with the handle, e.g. start a transfer.
3. Destroy the handle.

For example, to transfer a file using curl's easy interface, you could use the following code:

```
@REQUIRE "hurl"

; this function will be called whenever there is new data
Function p_WriteData(data$)
   WriteBytes(1, data$)
EndFunction

OpenFile(1, "test.html", #MODE_WRITE)

; create easy object and configure it
e = hurl.Easy({URL = "https://www.paypal.com/", WriteFunction =
        p_WriteData, FollowLocation = True})

; transfer data
e:Perform()

; destroy easy object
e:Close()
CloseFile(1)
```

The code above downloads the page at `https://www.paypal.com/` and saves it to the file `test.html` using curl's easy interface. It does so by first creating an easy object using hurl.Easy() and then setting the options `#CURLOPT_URL`, `#CURLOPT_WRITEFUNCTION`, and `#CURLOPT_FOLLOWLOCATION` on that easy object.

As shown above, curl options can be get directly when creating curl objects. Alternatively, you can also create an empty curl object and get the options afterwards, like so:

```
e = hurl.Easy()
e:SetOpt_URL("https://www.paypal.com/")
e:SetOpt_WriteFunction(p_WriteData)
e:SetOpt_FollowLocation(True)
```

This code does the same thing as the code in corresponding section above. The only difference is that options aren't get at creation time but after creation. Furthermore, you can also get multiple options at once after object creation. Here is another alternative for the two code snippets above:

```
e = hurl.Easy()
e:SetOpt({URL = "https://www.paypal.com/", WriteFunction = p_WriteData,
            FollowLocation = True})
```

Finally, you can also use easy:SetOpt() to get curl options on curl easy handles. So there is even a fourth way of doing what the code snippets above do. Here it is:

```
e = hurl.Easy()
e:SetOpt(#CURLOPT_URL, "https://www.paypal.com/")
e:SetOpt(#CURLOPT_WRITEFUNCTION, p_WriteData)
```

```
e:SetOpt(#CURLOPT_FOLLOWLOCATION, True)
```

For more information on the function of curl's various options, please refer to the following chapters.

4 General functions

4.1 hurl.Easy

NAME

hurl.Easy – start a libcurl easy session

SYNOPSIS

```
handle = hurl.Easy([table])
```

FUNCTION

This function must be the first function to call, and it returns a curl easy handle that you must use as input to other functions in the easy interface. This call must have a corresponding call to easy:Close() when the operation is complete.

The optional `table` argument allows you to get additional options for the easy object. It is possible to use all options here that can also be get separately using the easy:SetOpt() command. For example, to get `#CURLOPT_URL`, `#CURLOPT_VERBOSE`, and `#CURLOPT_FOLLOWLOCATION` at creation time, just do the following:

```
e = hurl.Easy({URL = "http://www.hollywood-mal.com",
        Verbose = True, FollowLocation = True})
```

This code does the same as:

```
e = hurl.Easy()
e:SetOpt_URL("http://www.hollywood-mal.com")
e:SetOpt_Verbose(True)
e:SetOpt_FollowLocation(True)
```

Alternatively, you could also use easy:SetOpt() to get those options, like so:

```
e = hurl.Easy()
e:SetOpt(#CURLOPT_URL, "http://www.hollywood-mal.com")
e:SetOpt(#CURLOPT_VERBOSE, True)
e:SetOpt(#CURLOPT_FOLLOWLOCATION, True)
```

All of the code snippets above do exactly the same thing.

INPUTS

`table` optional: table argument containing further options (see above)

RESULTS

`handle` curl easy handle

4.2 hurl.Form

NAME

hurl.Form – create HTTP multipart/formdata object

SYNOPSIS

```
handle = hurl.Form()
```

FUNCTION

This function creates a HTTP multipart/formdata object and returns it. You can then use functions like form:AddFile(), form:AddBuffer(), or form:AddContent() to fill it with content. This init call must have a corresponding call to form:Free() when the operation is complete.

INPUTS

none

RESULTS

`handle` HTTP multipart/formdata object

4.3 hurl.Multi

NAME

hurl.Multi – create a multi handle

SYNOPSIS

```
handle = hurl.Multi([table])
```

FUNCTION

This function returns a curl multi handle to be used as input to all the other multi-functions, sometimes referred to as a multi handle in some places in the documentation. This init call must have a corresponding call to multi:Close() when the operation is complete.

The optional `table` argument allows you to get additional options for the multi object. It is possible to use all options here that can also be get separately using the multi:SetOpt() command. See Section 4.1 [hurl:Easy], page 11, for an example.

INPUTS

`table` optional: table argument containing further options

RESULTS

`handle` curl multi handle

4.4 hurl.Share

NAME

hurl.Share – create a shared object

SYNOPSIS

```
handle = hurl.Share([table])
```

FUNCTION

This function returns a curl share handle to be used as input to all the other share-functions, sometimes referred to as a share handle in some places in the documentation. This init call must have a corresponding call to share:Close() when all operations using the share are complete.

This share handle is what you pass to curl using the `#CURLOPT_SHARE` option with easy:SetOpt() to make that specific curl handle use the data in this share.

The optional `table` argument allows you to get additional options for the share object. It is possible to use all options here that can also be get separately using the share:SetOpt() command. See Section 4.1 [hurl:Easy], page 11, for an example.

INPUTS

`table` optional: table argument containing further options

RESULTS

`handle` curl share handle

4.5 hurl.URL

NAME

hurl.URL – create a URL object (V2.0)

SYNOPSIS

```
handle = hurl.URL([url$, flags])
```

FUNCTION

Traditionally, URLs are passed to hURL using the easy:SetOpt_URL() method or its counterparts like `#CURLOPT_URL`. Starting with hURL 2.0, however, you can also pass URLs via URL objects created by this function. Once `hurl.URL()` returns, you can initialize the new URL object using methods like url:SetURL() or url:SetPort() and pass them to an easy handle by using easy:SetOpt_CURLU. Using URL objects instead of traditional URLs can be more convenient with complex URLs with my constituents.

Optionally, you can also initialize the URL object by passing a URL in `url$`. If you don't pass `url$`, you need to initialize the URL object later using url:SetURL(). It's also possible to pass a combination of the following flags:

`#CURLU_NON_SUPPORT_SCHEME`
: If get, allows you to get a non-supported scheme.

`#CURLU_URLENCODE`
: When get, libcurl URL encodes the part on entry, except for scheme, port and URL. When setting the path component with URL encoding enabled, the slash character will be skipped. The query part gets space-to-plus conversion before the URL conversion. This URL encoding is charset unaware and will convert the input on a byte-by-byte manner.

`#CURLU_DEFAULT_SCHEME`
: If get, will make libcurl allow the URL to be get without a scheme and then sets that to the default scheme: HTTPS. Overrides the `#CURLU_GUESS_SCHEME` option if both are get.

`#CURLU_GUESS_SCHEME`
: If get, will make libcurl allow the URL to be get without a scheme and it instead "guesses" which scheme that was intended based on the host name. If

the outermost sub-domain name matches DICT, FTP, IMAP, LDAP, POP3 or SMTP then that scheme will be used, otherwise it picks HTTP. Conflicts with the `#CURLU_DEFAULT_SCHEME` option which takes precedence if both are get.

`#CURLU_NO_AUTHORITY`
: If get, skips authority checks. The RFC allows individual schemes to omit the host part (normally the only mandatory part of the authority), but libcurl cannot know whether this is permitted for custom schemes. Specifying the flag permits empty authority sections, similar to how file scheme is handled.

`#CURLU_PATH_AS_IS`
: When get for CURLUPART_URL, this makes libcurl skip the normalization of the path. That is the procedure where curl otherwise removes sequences of dot-slash and dot-dot etc. The same option used for transfers is called `#CURLOPT_PATH_AS_IS`.

`#CURLU_ALLOW_SPACE`
: If get, the URL parser allows space (ASCII 32) where possible. The URL syntax does normally not allow spaces anywhere, but they should be encoded as %20 or '+'. When spaces are allowed, they are still not allowed in the scheme. When space is used and allowed in a URL, it will be stored as-is unless `#CURLU_URLENCODE` is also get, which then makes libcurl URL-encode the space before stored. This affects how the URL will be constructed when curl_url_get is subsequently used to extract the full URL or individual parts.

`#CURLU_DISALLOW_USER`
: If get, the URL parser will not accept embedded credentials for the `#CURLUPART_URL`, and will instead return for such URLs.

`#CURLU_APPENDQUERY`
: Can only be used with url:SetQuery(). The provided new part will then instead be appended at the end of the existing query - and if the previous part did not end with an ampersand , an ampersand gets inserted before the new appended part. When `#CURLU_APPENDQUERY` is used together with `#CURLU_URLENCODE`, the first '=' symbol will not be URL encoded.

When using the getter methods like url:GetURL() or url:GetPort() the flags will have a different function and there are some more flags. Here is a description of the flags that can be used with getter methods:

`#CURLU_DEFAULT_PORT`
: If the handle has no port stored, this option will make curl return the default port for the used scheme.

`#CURLU_DEFAULT_SCHEME`
: If the handle has no scheme stored, this option will make curl return the default scheme instead of error.

`#CURLU_NO_DEFAULT_PORT`
: Instructs curl to not return a port number if it matches the default port for the scheme.

`#CURLU_URLDECODE`
: Asks curl to URL decode the contents before returning it. It will not attempt to decode the scheme, the port number or the full URL. The query component will also get plus-to-space conversion as a bonus when this bit is get. Note that this URL decoding is charset unaware and you will get a string back with data that could be intended for a particular encoding. If there's any byte values lower than 32 in the decoded string, the get operation will return an error instead.

`#CURLU_URLENCODE`
: If get, it will make curl URL encode the host name part when a full URL is retrieved. If not get (default), libcurl returns the URL with the host name `"raw"` to support IDN names to appear as-is. IDN host names are typically using non-ASCII bytes that otherwise will be percent-encoded. Note that even when not asking for URL encoding, the '%' (byte 37) will be URL encoded to make sure the host name remains valid.

`#CURLU_PUNYCODE`
: If get and `#CURLU_URLENCODE` is not get, and asked to retrieve the host or URL parts, libcurl returns the host name in its punycode version if it contains any non-ASCII octets (and is an IDN name). If libcurl is built without IDN capabilities, using this bit will make curl return if the host name contains anything outside the ASCII range.

INPUTS

`url$` optional: URL to initialize object with

`flags` optional: flags to use on initialization (see above)

RESULTS

`handle` URL object

EXAMPLE

```
e = hurl.Easy()
u = hurl.URL("https://www.paypal.com/")
e:SetOpt_CURLU(u)
e:SetOpt_WriteFunction(p_WriteData)
e:SetOpt_FollowLocation(True)
e:Perform()
e:Close()
```

The code above shows how to create and use a URL object with hURL's easy interface.

4.6 hurl.Version

NAME

hurl.Version – returns the libcurl version string

SYNOPSIS

```
v$ = hurl.Version()
```

FUNCTION

Returns a human readable string with the version number of libcurl and some of its important components (like OpenSSL version).

We recommend using hurl.VersionInfo()!

INPUTS

none

RESULTS

`v$` libcurl version string

4.7 hurl.VersionInfo

NAME

hurl.VersionInfo – returns run-time libcurl version info

SYNOPSIS

```
t = hurl.VersionInfo()
```

FUNCTION

This function returns detailed information about the run-time libcurl version.

The table argument will contain the following fields:

`Version:` An ASCII string for the libcurl version.

`VersionNum:`
A 24 bit number created like this: `<`8 bits major number`> | <`8 bits minor number`> | <`8 bits patch number`>`. Version 7.9.8 is therefore returned as 0x070908.

`Host:` An ASCII string showing what host information that this libcurl was built for. As discovered by a configure script or get by the build environment.

`Features:`
This is a table that contains the following boolean fields, all of which are either get to `True` or `False`, depending on whether or not the specific feature is available.

- `IPV6:` Supports IPv6
- `Kerberos4:` Supports Kerberos V4 (when using FTP)
- `Kerberos5:` Supports Kerberos V5 authentication for FTP, IMAP, POP3, SMTP and SOCKSv5 proxy (Added in 7.40.0)
- `SSL:` Supports SSL (HTTPS/FTPS) (Added in 7.10)
- `Libz:` Supports HTTP deflate using libz (Added in 7.10)
- `NTLM:` Supports HTTP NTLM (added in 7.10.6)

`GSSNegotiate:`
Supports HTTP GSS-Negotiate (added in 7.10.6)

`Debug:` libcurl was built with debug capabilities (added in 7.10.6)

`CurlDebug:`
libcurl was built with memory tracking debug capabilities. This is mainly of interest for libcurl hackers. (added in 7.19.6)

`AsynchDNS:`
libcurl was built with support for asynchronous name lookups, which allows more exact timeouts (even on Windows) and less blocking when using the multi interface. (added in 7.10.7)

`SPNEGO:` libcurl was built with support for SPNEGO authentication (Simple and Protected GSS-API Negotiation Mechanism, defined in RFC 2478.) (added in 7.10.8)

`LargeFile:`
libcurl was built with support for large files. (Added in 7.11.1)

`IDN:` libcurl was built with support for IDNA, domain names with international letters. (Added in 7.12.0)

`SSPI:` libcurl was built with support for SSPI. This is only available on Windows and makes libcurl use Windows-provided functions for Kerberos, NTLM, SPNEGO and Digest authentication. It also allows libcurl to use the current user credentials without the app having to pass them on. (Added in 7.13.2)

`GSSAPI:` libcurl was built with support for GSS-API. This makes libcurl use provided functions for Kerberos and SPNEGO authentication. It also allows libcurl to use the current user credentials without the app having to pass them on. (Added in 7.38.0)

`CONV:` libcurl was built with support for character conversions, as provided by the #CURLOPT_CONV_* callbacks. (Added in 7.15.4)

`TLSAuthSRP:`
libcurl was built with support for TLS-SRP. (Added in 7.21.4)

`NTLM_WB:` libcurl was built with support for NTLM delegation to a winbind helper. (Added in 7.22.0)

`HTTP2:` libcurl was built with support for HTTP2. (Added in 7.33.0)

`HTTPSProxy:`
libcurl was built with support for HTTPS-proxy. (Added in 7.52.0)

`MultiSSL:`
libcurl was built with multiple SSL backends. (Added in 7.56.0)

`Brotli:` Supports HTTP Brotli content encoding using libbrotlidec. (Added in 7.57.0)

`AltSvc:` HTTP Alt-Svc parsing and the associated options. (Added in 7.64.1)

`HTTP3:` HTTP/3 and QUIC support are built-in. (Added in 7.66.0)

`zstd:` Supports HTTP zstd content encoding using zstd library. (Added in 7.72.0)

`HSTS:` ibcurl was built with support for HSTS. (HTTP Strict Transport Security) (Added in 7.74.0)

`SSLVersion:`
An ASCII string for the TLS library name `+` version used. For example `"Schannel"`, `"SecureTransport"` or `"OpenSSL/1.1.0g"`.

`SSLVersionNum:`
Always 0.

`LibzVersion:`
An ASCII string (there is no numerical version).

`Protocols:`
This is get to a table of strings, containing the names protocols that libcurl supports (using lowercase letters). The protocol names are the same as would be used in URLs.

INPUTS

none

RESULTS

`t` table containing information about the libcurl version

5 Easy methods

5.1 easy:Close

NAME

easy:Close – end a libcurl easy handle

SYNOPSIS

```
easy:Close()
```

FUNCTION

This function must be the last function to call for an easy session. It is the opposite of the hurl.Easy() function and must be called with the same handle as input that a hurl.Easy() call returned.

This might close all connections this handle has used and possibly has kept open until now - unless it was attached to a multi handle while doing the transfers. Don't call this function if you intend to transfer more files, re-using handles is a key to good performance with libcurl.

Occasionally you may get your progress callback or header callback called from within easy:Close() (if previously get for the handle using easy:SetOpt()). Like if libcurl decides to shut down the connection and the protocol is of a kind that requires a command/response sequence before disconnect. Examples of such protocols are FTP, POP3 and IMAP.

Any use of the handle after this function has been called and have returned, is illegal. easy:Close() kills the handle and all memory associated with it!

INPUTS

none

5.2 easy:Escape

NAME

easy:Escape – URL encodes the given string

SYNOPSIS

```
e$ = easy:Escape(s$)
```

FUNCTION

This function converts the given input string `s$` to a URL encoded string and returns that. All input characters that are not a-z, A-Z, 0-9, '-', '.', '_' or '~' are converted to their "URL escaped" version (%NN where NN is a two-digit hexadecimal number).

libcurl is typically not aware of, nor does it care about, character encodings. easy:Escape() encodes the data byte-by-byte into the URL encoded version without knowledge or care for what particular character encoding the application or the receiving server may assume that the data uses.

The caller of easy:Escape() must make sure that the data passed in to the function is encoded correctly.

INPUTS

`s$` string to escape

RESULTS

`e$` escaped string

5.3 easy:GetInfo

NAME

easy:GetInfo – extract information from a curl handle

SYNOPSIS

```
info = easy:GetInfo(type)
```

FUNCTION

Request internal information from the curl session with this function. The `type` argument specifies what information should be retrieved. Use this function AFTER a performed transfer if you want to get transfer related data.

The following types are currently supported for `type`:

`#CURLINFO_APPCONNECT_TIME`
: Time from start until SSL/SSH handshake completed. See Section 5.4 [easy:GetInfo_AppConnect_Time], page 25, for details.

`#CURLINFO_APPCONNECT_TIME_T`
: Time from start until SSL/SSH handshake completed. See Section 5.5 [easy:GetInfo_AppConnect_Time_t], page 25, for details. (V2.0)

`#CURLINFO_CAINFO`
: Get the default built-in CA certificate path. See Section 5.6 [easy:GetInfo_CAInfo], page 26, for details. (V2.0)

`#CURLINFO_CAPATH`
: Get the default built-in CA path string. See Section 5.7 [easy:GetInfo_CAPath], page 26, for details. (V2.0)

`#CURLINFO_CERTINFO`
: Certificate chain. See Section 5.8 [easy:GetInfo_CertInfo], page 27, for details.

`#CURLINFO_CONDITION_UNMET`
: Whether or not a time conditional was met. See Section 5.9 [easy:GetInfo_Condition_Unmet], page 27, for details.

`#CURLINFO_CONNECT_TIME`
: Time from start until remote host or proxy completed. See Section 5.10 [easy:GetInfo_Connect_Time], page 27, for details.

`#CURLINFO_CONNECT_TIME_T`
: Time from start until remote host or proxy completed. See Section 5.11 [easy:GetInfo_Connect_Time_t], page 28, for details. (V2.0)

#CURLINFO_CONTENT_LENGTH_DOWNLOAD
(Deprecated) Content length from the Content-Length header. See Section 5.12 [easy:GetInfo_Content_Length_Download], page 28, for details.

#CURLINFO_CONTENT_LENGTH_DOWNLOAD_T
Content length from the Content-Length header. See Section 5.13 [easy:GetInfo_Content_Length_Download_t], page 29, for details.

#CURLINFO_CONTENT_LENGTH_UPLOAD
(Deprecated) Upload size. See Section 5.14 [easy:GetInfo_Content_Length_Upload], page 29, for details.

#CURLINFO_CONTENT_LENGTH_UPLOAD_T
Upload size. See Section 5.15 [easy:GetInfo_Content_Length_Upload_t], page 29, for details.

#CURLINFO_CONTENT_TYPE
Content type from the Content-Type header. See Section 5.16 [easy:GetInfo_Content_Type], page 30, for details.

#CURLINFO_COOKIELIST
List of all known cookies. See Section 5.17 [easy:GetInfo_CookieList], page 30, for details.

#CURLINFO_EFFECTIVE_METHOD
Last used HTTP method. See Section 5.18 [easy:GetInfo_Effective_Method], page 31, for details. (V2.0)

#CURLINFO_EFFECTIVE_URL
Last used URL. See Section 5.19 [easy:GetInfo_Effective_URL], page 31, for details.

#CURLINFO_FILETIME
Remote time of the retrieved document. See Section 5.20 [easy:GetInfo_FileTime], page 31, for details.

#CURLINFO_FTP_ENTRY_PATH
The entry path after logging in to an FTP server. See Section 5.21 [easy:GetInfo_FTP_Entry_Path], page 32, for details.

#CURLINFO_HEADER_SIZE
Number of bytes of all headers received. See Section 5.22 [easy:GetInfo_Header_Size], page 32, for details.

#CURLINFO_HTTP_CONNECTCODE
Last proxy CONNECT response code. See Section 5.24 [easy:GetInfo_HTTP_ConnectCode], page 33, for details.

#CURLINFO_HTTP_VERSION
The http version used in the connection. See Section 5.25 [easy:GetInfo_HTTP_Version], page 33, for details.

`#CURLINFO_HTTPAUTH_AVAIL`
: Available HTTP authentication methods. See Section 5.23 [easy:GetInfo_HTTPAuth_Avail], page 33, for details.

`#CURLINFO_LASTSOCKET`
: Last socket used. See Section 5.26 [easy:GetInfo_LastSocket], page 34, for details.

`#CURLINFO_LOCAL_IP`
: Local-end IP address of last connection. See Section 5.27 [easy:GetInfo_Local_IP], page 34, for details.

`#CURLINFO_LOCAL_PORT`
: Local-end port of last connection. See Section 5.28 [easy:GetInfo_Local_Port], page 35, for details.

`#CURLINFO_NAMELOOKUP_TIME`
: Time from start until name resolving completed. See Section 5.29 [easy:GetInfo_NameLookup_Time], page 35, for details.

`#CURLINFO_NAMELOOKUP_TIME_T`
: Time from start until name resolving completed. See Section 5.30 [easy:GetInfo_NameLookup_Time_t], page 35, for details. (V2.0)

`#CURLINFO_NUM_CONNECTS`
: Number of new successful connections used for previous transfer. See Section 5.31 [easy:GetInfo_Num_Connects], page 36, for details.

`#CURLINFO_OS_ERRNO`
: The errno from the last failure to connect. See Section 5.32 [easy:GetInfo_OS_ErrNo], page 36, for details.

`#CURLINFO_PRETRANSFER_TIME`
: Time from start until just before the transfer begins. See Section 5.33 [easy:GetInfo_PreTransfer_Time], page 36, for details.

`#CURLINFO_PRETRANSFER_TIME_T`
: Time from start until just before the transfer begins. See Section 5.34 [easy:GetInfo_PreTransfer_Time_t], page 37, for details. (V2.0)

`#CURLINFO_PRIMARY_IP`
: IP address of the last connection. See Section 5.35 [easy:GetInfo_Primary_IP], page 37, for details.

`#CURLINFO_PRIMARY_PORT`
: Port of the last connection. See Section 5.36 [easy:GetInfo_Primary_Port], page 38, for details.

`#CURLINFO_PROTOCOL`
: The protocol used for the connection. See Section 5.37 [easy:GetInfo_Protocol], page 38, for details.

`#CURLINFO_PROXY_ERROR`
: Detailed proxy error. See Section 5.39 [easy:GetInfo_Proxy_Error], page 39, for details. (V2.0)

#CURLINFO_PROXY_SSL_VERIFYRESULT
: Proxy certificate verification result. See Section 5.40 [easy:GetInfo_Proxy_SSL_VerifyResult], page 40, for details.

#CURLINFO_PROXYAUTH_AVAIL
: Available HTTP proxy authentication methods. See Section 5.38 [easy:GetInfo_ProxyAuth_Avail], page 39, for details.

#CURLINFO_REDIRECT_COUNT
: Total number of redirects that were followed. See Section 5.41 [easy:GetInfo_Redirect_Count], page 40, for details.

#CURLINFO_REDIRECT_TIME
: Time taken for all redirect steps before the final transfer. See Section 5.42 [easy:GetInfo_Redirect_Time], page 40, for details.

#CURLINFO_REDIRECT_TIME_T
: Time taken for all redirect steps before the final transfer. See Section 5.43 [easy:GetInfo_Redirect_Time_t], page 41, for details. (V2.0)

#CURLINFO_REDIRECT_URL
: URL a redirect would take you to, had you enabled redirects. See Section 5.44 [easy:GetInfo_Redirect_URL], page 41, for details.

#CURLINFO_REFERER
: Referrer header. See Section 5.45 [easy:GetInfo_Referer], page 42, for details. (V2.0)

#CURLINFO_REQUEST_SIZE
: Number of bytes sent in the issued HTTP requests. See Section 5.46 [easy:GetInfo_Request_Size], page 42, for details.

#CURLINFO_RESPONSE_CODE
: Last received response code. See Section 5.47 [easy:GetInfo_Response_Code], page 42, for details.

#CURLINFO_RETRY_AFTER
: The value from the from the Retry-After header. See Section 5.48 [easy:GetInfo_Retry_After], page 43, for details. (V2.0)

#CURLINFO_RTSP_CLIENT_CSEQ
: RTSP CSeq that will next be used. See Section 5.49 [easy:GetInfo_RTSP_Client_CSeq], page 43, for details.

#CURLINFO_RTSP_CSEQ_RECV
: RTSP CSeq last received. See Section 5.50 [easy:GetInfo_RTSP_CSeq_Recv], page 44, for details.

#CURLINFO_RTSP_SERVER_CSEQ
: RTSP CSeq that will next be expected. See Section 5.51 [easy:GetInfo_RTSP_Server_CSeq], page 44, for details.

#CURLINFO_RTSP_SESSION_ID
: RTSP session ID. See Section 5.52 [easy:GetInfo_RTSP_Session_ID], page 44, for details.

`#CURLINFO_SCHEME`
: The scheme used for the connection. See Section 5.53 [easy:GetInfo_Scheme], page 45, for details.

`#CURLINFO_SIZE_DOWNLOAD`
: (Deprecated) Number of bytes downloaded. See Section 5.54 [easy:GetInfo_Size_Download], page 45, for details.

`#CURLINFO_SIZE_DOWNLOAD_T`
: Number of bytes downloaded. See Section 5.55 [easy:GetInfo_Size_Download_t], page 46, for details.

`#CURLINFO_SIZE_UPLOAD`
: (Deprecated) Number of bytes uploaded. See Section 5.56 [easy:GetInfo_Size_Upload], page 46, for details.

`#CURLINFO_SIZE_UPLOAD_T`
: Number of bytes uploaded. See Section 5.57 [easy:GetInfo_Size_Upload_t], page 46, for details.

`#CURLINFO_SPEED_DOWNLOAD`
: (Deprecated) Average download speed. See Section 5.58 [easy:GetInfo_Speed_Download], page 47, for details.

`#CURLINFO_SPEED_DOWNLOAD_T`
: Average download speed. See Section 5.59 [easy:GetInfo_Speed_Download_t], page 47, for details.

`#CURLINFO_SPEED_UPLOAD`
: (Deprecated) Average upload speed. See Section 5.60 [easy:GetInfo_Speed_Upload], page 48, for details.

`#CURLINFO_SPEED_UPLOAD_T`
: Average upload speed. See Section 5.61 [easy:GetInfo_Speed_Upload_t], page 48, for details.

`#CURLINFO_SSL_ENGINES`
: A list of OpenSSL crypto engines. See Section 5.62 [easy:GetInfo_SSL_Engines], page 48, for details.

`#CURLINFO_SSL_VERIFYRESULT`
: Certificate verification result. See Section 5.63 [easy:GetInfo_SSL_VerifyResult], page 49, for details.

`#CURLINFO_STARTTRANSFER_TIME`
: Time from start until just when the first byte is received. See Section 5.64 [easy:GetInfo_StartTransfer_Time], page 49, for details.

`#CURLINFO_STARTTRANSFER_TIME_T`
: Time from start until just when the first byte is received. See Section 5.65 [easy:GetInfo_StartTransfer_Time_t], page 50, for details. (V2.0)

`#CURLINFO_TOTAL_TIME`
: Total time of previous transfer. See Section 5.66 [easy:GetInfo_Total_Time], page 50, for details.

`#CURLINFO_TOTAL_TIME_T`
: Total time of previous transfer. See Section 5.67 [easy:GetInfo_Total_Time_t], page 50, for details. (V2.0)

INPUTS

`type` type of information to retrieve

RESULTS

`info` output value

5.4 easy:GetInfo_AppConnect_Time

NAME

easy:GetInfo_AppConnect_Time – get the time until the SSL/SSH handshake is completed

SYNOPSIS

```
timep = easy:GetInfo_AppConnect_Time()
```

FUNCTION

Returns the time, in seconds, it took from the start until the SSL/SSH connect/handshake to the remote host was completed. This time is most often very near to the `#CURLINFO_PRETRANSFER_TIME` time, except for cases such as HTTP pipelining where the pretransfer time can be delayed due to waits in line for the pipeline and more.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.5 easy:GetInfo_AppConnect_Time_t

NAME

easy:GetInfo_AppConnect_Time_t – get the time until the SSL/SSH handshake is completed (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_AppConnect_Time_t()
```

FUNCTION

Returns the time, in microseconds, it took from the start until the SSL/SSH connect/handshake to the remote host was completed. This time is most often close to the `#CURLINFO_PRETRANSFER_TIME_T` time, except for cases such as HTTP pipelining where the pretransfer time can be delayed due to waits in line for the pipeline and more.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.6 easy:GetInfo_CAInfo

NAME

easy:GetInfo_CAInfo – get the default built-in CA certificate path (V2.0)

SYNOPSIS

```
path = easy:GetInfo_CAInfo()
```

FUNCTION

Returns a string holding the default built-in path used for the `#CURLOPT_CAINFO` option unless get by the user.

Note that in a situation where libcurl has been built to support multiple TLS libraries, this option might return a string even if the specific TLS library currently get to be used does not support `#CURLOPT_CAINFO`.

This is a path identifying a single file containing CA certificates.

INPUTS

none

RESULTS

`path` output value

5.7 easy:GetInfo_CAPath

NAME

easy:GetInfo_CAPath – get the default built-in CA path string (V2.0)

SYNOPSIS

```
path = easy:GetInfo_CAPath()
```

FUNCTION

Returns a string holding the default built-in path used for the `#CURLOPT_CAPATH` option unless get by the user.

Note that in a situation where libcurl has been built to support multiple TLS libraries, this option might return a string even if the specific TLS library currently get to be used does not support `#CURLOPT_CAPATH`.

This is a path identifying a directory.

INPUTS

none

RESULTS

`path` output value

5.8 easy:GetInfo_CertInfo

NAME

easy:GetInfo_CertInfo – get the TLS certificate chain

SYNOPSIS

```
chainp = easy:GetInfo_CertInfo()
```

FUNCTION

Returns a table that holds a number of string lists with info about the certificate chain, assuming you had `#CURLOPT_CERTINFO` enabled when the request was made. The table reports how many certs it found and then you can extract info for each of those certs by following the string lists. The info chain is provided in a series of data in the format `"name:content"` where the content is for the specific named data.

INPUTS

none

RESULTS

`chainp` output value

5.9 easy:GetInfo_Condition_Unmet

NAME

easy:GetInfo_Condition_Unmet – get info on unmet time conditional

SYNOPSIS

```
unmet = easy:GetInfo_Condition_Unmet()
```

FUNCTION

Returns the number 1 if the condition provided in the previous request didn't match (see `#CURLOPT_TIMECONDITION`). Alas, if this returns a 1 you know that the reason you didn't get data in return is because it didn't fulfill the condition. Returns zero if the condition instead was met.

INPUTS

none

RESULTS

`unmet` output value

5.10 easy:GetInfo_Connect_Time

NAME

easy:GetInfo_Connect_Time – get the time until connect

SYNOPSIS

```
timep = easy:GetInfo_Connect_Time()
```

FUNCTION

Returns the total time in seconds from the start until the connection to the remote host (or proxy) was completed.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.11 easy:GetInfo_Connect_Time_t

NAME

easy:GetInfo_Connect_Time_t – get the time until connect (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_Connect_Time_t()
```

FUNCTION

Returns the total time in microseconds from the start until the connection to the remote host (or proxy) was completed.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.12 easy:GetInfo_Content_Length_Download

NAME

easy:GetInfo_Content_Length_Download – get content-length of download

SYNOPSIS

```
content_length = easy:GetInfo_Content_Length_Download()
```

FUNCTION

Returns the content-length of the download. This is the value read from the Content-Length: field. Since 7.19.4, this returns -1 if the size isn't known.

`#CURLINFO_CONTENT_LENGTH_DOWNLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS

none

RESULTS

`content_length`
output value

5.13 easy:GetInfo_Content_Length_Download_t

NAME

easy:GetInfo_Content_Length_Download_t – get content-length of download

SYNOPSIS

```
content_length = easy:GetInfo_Content_Length_Download_t()
```

FUNCTION

Returns the content-length of the download. This is the value read from the Content-Length: field. Stores -1 if the size isn't known.

INPUTS

none

RESULTS

`content_length`
: output value

5.14 easy:GetInfo_Content_Length_Upload

NAME

easy:GetInfo_Content_Length_Upload – get the specified size of the upload

SYNOPSIS

```
content_length = easy:GetInfo_Content_Length_Upload()
```

FUNCTION

Returns the specified size of the upload. Since 7.19.4, this returns -1 if the size isn't known.

`#CURLINFO_CONTENT_LENGTH_UPLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS

none

RESULTS

`content_length`
: output value

5.15 easy:GetInfo_Content_Length_Upload_t

NAME

easy:GetInfo_Content_Length_Upload_t – get the specified size of the upload

SYNOPSIS

```
content_length = easy:GetInfo_Content_Length_Upload_t()
```

FUNCTION

Returns the specified size of the upload. Returns -1 if the size isn't known.

INPUTS
none

RESULTS

`content_length`
output value

5.16 easy:GetInfo_Content_Type

NAME
easy:GetInfo_Content_Type – get Content-Type

SYNOPSIS
```
ct = easy:GetInfo_Content_Type()
```

FUNCTION
Returns the content-type of the downloaded object. This is the value read from the Content-Type: field. If you get `Nil`, it means that the server didn't send a valid Content-Type header or that the protocol used doesn't support this.

INPUTS
none

RESULTS

`ct` output value

5.17 easy:GetInfo_CookieList

NAME
easy:GetInfo_CookieList – get all known cookies

SYNOPSIS
```
cookies = easy:GetInfo_CookieList()
```

FUNCTION
Returns a list of all cookies curl knows (expired ones, too). If there are no cookies (cookies for the handle have not been enabled or simply none have been received) `Nil` will be returned.

Since 7.43.0 cookies that were imported in the Set-Cookie format without a domain name are not exported by this option.

INPUTS
none

RESULTS

`cookies` output value

5.18 easy:GetInfo_Effective_Method

NAME

easy:GetInfo_Effective_Method – get the last used HTTP method (V2.0)

SYNOPSIS

```
methodp = easy:GetInfo_Effective_Method()
```

FUNCTION

Returns a string containing the last used effective HTTP method.

In cases when you have asked libcurl to follow redirects, the method may not be the same method the first request would use.

INPUTS

none

RESULTS

`methodp` output value

5.19 easy:GetInfo_Effective_URL

NAME

easy:GetInfo_Effective_URL – get the last used URL

SYNOPSIS

```
urlp = easy:GetInfo_Effective_URL()
```

FUNCTION

Returns the last used effective URL.

In cases when you've asked libcurl to follow redirects, it may very well not be the same value you get with `#CURLOPT_URL`.

INPUTS

none

RESULTS

`urlp` output value

5.20 easy:GetInfo_FileTime

NAME

easy:GetInfo_FileTime – get the remote time of the retrieved document

SYNOPSIS

```
timep = easy:GetInfo_FileTime()
```

FUNCTION

Returns the remote time of the retrieved document (in number of seconds since 1 jan 1970 in the GMT/UTC time zone). If you get -1, it can be because of many reasons (it

might be unknown, the server might hide it or the server doesn't support the command that tells document time etc) and the time of the document is unknown.

You must tell libcurl to collect this information before the transfer is made, by using the `#CURLOPT_FILETIME` option to easy:SetOpt() or you will unconditionally get a -1 back.

Consider using `#CURLINFO_FILETIME_T` to be able to extract dates beyond the year 2038 on systems using 32 bit longs.

INPUTS

none

RESULTS

`timep` output value

5.21 easy:GetInfo_FTP_Entry_Path

NAME

easy:GetInfo_FTP_Entry_Path – get entry path in FTP server

SYNOPSIS

```
path = easy:GetInfo_FTP_Entry_Path()
```

FUNCTION

Returns a string holding the path of the entry path. That is the initial path libcurl ended up in when logging on to the remote FTP server. This returns `Nil` if something is wrong.

INPUTS

none

RESULTS

`path` output value

5.22 easy:GetInfo_Header_Size

NAME

easy:GetInfo_Header_Size – get size of retrieved headers

SYNOPSIS

```
sizep = easy:GetInfo_Header_Size()
```

FUNCTION

Returns the total size of all the headers received. Measured in number of bytes.

The total includes the size of any received headers suppressed by `#CURLOPT_SUPPRESS_CONNECT_HEADERS`.

INPUTS

none

RESULTS

`sizep` output value

5.23 easy:GetInfo_HTTPAuth_Avail

NAME

easy:GetInfo_HTTPAuth_Avail – get available HTTP authentication methods

SYNOPSIS

```
authp = easy:GetInfo_HTTPAuth_Avail()
```

FUNCTION

Returns a bitmask indicating the authentication method(s) available according to the previous response. The meaning of the bits is explained in the `#CURLOPT_HTTPAUTH` option for easy:SetOpt().

INPUTS

none

RESULTS

`authp` output value

5.24 easy:GetInfo_HTTP_ConnectCode

NAME

easy:GetInfo_HTTP_ConnectCode – get the CONNECT response code

SYNOPSIS

```
p = easy:GetInfo_HTTP_ConnectCode()
```

FUNCTION

Returns the last received HTTP proxy response code to a `CONNECT` request. The returned value will be zero if no such response code was available.

INPUTS

none

RESULTS

`p` output value

5.25 easy:GetInfo_HTTP_Version

NAME

easy:GetInfo_HTTP_Version – get the http version used in the connection

SYNOPSIS

```
p = easy:GetInfo_HTTP_Version()
```

FUNCTION

Returns the version used in the last http connection. The returned value will be `#CURL_HTTP_VERSION_1_0`, `#CURL_HTTP_VERSION_1_1`, or `#CURL_HTTP_VERSION_2_0`, or 0 if the version can't be determined.

INPUTS
none

RESULTS
`p` output value

5.26 easy:GetInfo_LastSocket

NAME
easy:GetInfo_LastSocket – get the last socket used

SYNOPSIS
```
socket = easy:GetInfo_LastSocket()
```

FUNCTION
Deprecated since 7.45.0. Use `#CURLINFO_ACTIVESOCKET` instead.

Returns the last socket used by this curl session. If the socket is no longer valid, -1 is returned. When you finish working with the socket, you must call easy:Close() as usual and let libcurl close the socket and cleanup other resources associated with the handle. This is typically used in combination with `#CURLOPT_CONNECT_ONLY`.

NOTE: this API is deprecated since it is not working on win64 where the SOCKET type is 64 bits large while its 'long' is 32 bits. Use the `#CURLINFO_ACTIVESOCKET` instead, if possible.

INPUTS
none

RESULTS
`socket` output value

5.27 easy:GetInfo_Local_IP

NAME
easy:GetInfo_Local_IP – get local IP address of last connection

SYNOPSIS
```
ip = easy:GetInfo_Local_IP()
```

FUNCTION
Returns a string holding the IP address of the local end of most recent connection done with this curl handle. This string may be IPv6 when that is enabled.

INPUTS
none

RESULTS
`ip` output value

5.28 easy:GetInfo_Local_Port

NAME

easy:GetInfo_Local_Port – get the latest local port number

SYNOPSIS

```
portp = easy:GetInfo_Local_Port()
```

FUNCTION

Returns the local port number of the most recent connection done with this curl handle.

INPUTS

none

RESULTS

`portp` output value

5.29 easy:GetInfo_NameLookup_Time

NAME

easy:GetInfo_NameLookup_Time – get the name lookup time

SYNOPSIS

```
timep = easy:GetInfo_NameLookup_Time()
```

FUNCTION

Returns the total time in seconds from the start until the name resolving was completed.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.30 easy:GetInfo_NameLookup_Time_t

NAME

easy:GetInfo_NameLookup_Time_t – get the name lookup time in microseconds (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_NameLookup_Time_t()
```

FUNCTION

Returns the total time in microseconds from the start until the name resolving was completed.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.31 easy:GetInfo_Num_Connects

NAME

easy:GetInfo_Num_Connects – get number of created connections

SYNOPSIS

```
nump = easy:GetInfo_Num_Connects()
```

FUNCTION

Returns how many new connections libcurl had to create to achieve the previous transfer (only the successful connects are counted). Combined with `#CURLINFO_REDIRECT_COUNT` you are able to know how many times libcurl successfully reused existing connection(s) or not. See the connection options of easy:SetOpt() to see how libcurl tries to make persistent connections to save time.

INPUTS

none

RESULTS

`nump` output value

5.32 easy:GetInfo_OS_ErrNo

NAME

easy:GetInfo_OS_ErrNo – get errno number from last connect failure

SYNOPSIS

```
errnop = easy:GetInfo_OS_ErrNo()
```

FUNCTION

Returns the errno variable from a connect failure. Note that the value is only get on failure, it is not reset upon a successful operation. The number is OS and system specific.

INPUTS

none

RESULTS

`errnop` output value

5.33 easy:GetInfo_PreTransfer_Time

NAME

easy:GetInfo_PreTransfer_Time – get the time until the file transfer start

SYNOPSIS

```
timep = easy:GetInfo_PreTransfer_Time()
```

FUNCTION

Returns the time, in seconds, it took from the start until the file transfer is just about to begin. This includes all pre-transfer commands and negotiations that are specific to the particular protocol(s) involved. It does not involve the sending of the protocol- specific request that triggers a transfer.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.34 easy:GetInfo_PreTransfer_Time_t

NAME

easy:GetInfo_PreTransfer_Time_t – get the time until the file transfer start (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_PreTransfer_Time_t()
```

FUNCTION

Returns the time, in microseconds, it took from the start until the file transfer is just about to begin. This includes all pre-transfer commands and negotiations that are specific to the particular protocol(s) involved. It does not involve the sending of the protocol-specific request that triggers a transfer.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.35 easy:GetInfo_Primary_IP

NAME

easy:GetInfo_Primary_IP – get IP address of last connection

SYNOPSIS

```
ip = easy:GetInfo_Primary_IP()
```

FUNCTION

Returns a string holding the IP address of the most recent connection done with this curl handle. This string may be IPv6 when that is enabled.

INPUTS

none

RESULTS

`ip` output value

5.36 easy:GetInfo_Primary_Port

NAME

easy:GetInfo_Primary_Port – get the latest destination port number

SYNOPSIS

```
portp = easy:GetInfo_Primary_Port()
```

FUNCTION

Returns the destination port of the most recent connection done with this curl handle.

INPUTS

none

RESULTS

`portp` output value

5.37 easy:GetInfo_Protocol

NAME

easy:GetInfo_Protocol – get the protocol used in the connection

SYNOPSIS

```
p = easy:GetInfo_Protocol()
```

FUNCTION

Returns the version used in the last http connection. The returned value will be exactly one of the `#CURLPROTO_XXX` values:

```
#CURLPROTO_DICT
#CURLPROTO_FILE
#CURLPROTO_FTP
#CURLPROTO_FTPS
#CURLPROTO_GOPHER
#CURLPROTO_HTTP
#CURLPROTO_HTTPS
#CURLPROTO_IMAP
#CURLPROTO_IMAPS
#CURLPROTO_LDAP
#CURLPROTO_LDAPS
#CURLPROTO_POP3
#CURLPROTO_POP3S
#CURLPROTO_RTMP
#CURLPROTO_RTMPE
#CURLPROTO_RTMPS
```

```
#CURLPROTO_RTMPT
#CURLPROTO_RTMPTE
#CURLPROTO_RTMPTS
#CURLPROTO_RTSP
#CURLPROTO_SCP
#CURLPROTO_SFTP
#CURLPROTO_SMB
#CURLPROTO_SMBS
#CURLPROTO_SMTP
#CURLPROTO_SMTPS
#CURLPROTO_TELNET
#CURLPROTO_TFTP
```

INPUTS

none

RESULTS

`p` output value

5.38 easy:GetInfo_ProxyAuth_Avail

NAME

easy:GetInfo_ProxyAuth_Avail – get available HTTP proxy authentication methods

SYNOPSIS

```
authp = easy:GetInfo_ProxyAuth_Avail()
```

FUNCTION

Returns a bitmask indicating the authentication method(s) available according to the previous response. The meaning of the bits is explained in the `#CURLOPT_PROXYAUTH` option for easy:SetOpt().

INPUTS

none

RESULTS

`authp` output value

5.39 easy:GetInfo_Proxy_Error

NAME

easy:GetInfo_Proxy_Error – get the detailed (SOCKS) proxy error (V2.0)

SYNOPSIS

```
detail = easy:GetInfo_Proxy_Error()
```

FUNCTION

Returns a detailed error code when the most recent transfer returned a `#CURLE_PROXY` error.

The error code will be zero (`#CURLPX_OK`) if no response code was available.

INPUTS
none

RESULTS
`detail` output value

5.40 easy:GetInfo_Proxy_SSL_VerifyResult

NAME
easy:GetInfo_Proxy_SSL_VerifyResult – get the result of the proxy certificate verification

SYNOPSIS
```
result = easy:GetInfo_Proxy_SSL_VerifyResult()
```

FUNCTION
Returns the result of the certificate verification that was requested (using the `#CURLOPT_PROXY_SSL_VERIFYPEER` option. This is only used for HTTPS proxies.

INPUTS
none

RESULTS
`result` output value

5.41 easy:GetInfo_Redirect_Count

NAME
easy:GetInfo_Redirect_Count – get the number of redirects

SYNOPSIS
```
countp = easy:GetInfo_Redirect_Count()
```

FUNCTION
Returns the total number of redirections that were actually followed.

INPUTS
none

RESULTS
`countp` output value

5.42 easy:GetInfo_Redirect_Time

NAME
easy:GetInfo_Redirect_Time – get the time for all redirection steps

SYNOPSIS
```
timep = easy:GetInfo_Redirect_Time()
```

FUNCTION

Returns the total time, in seconds, it took for all redirection steps include name lookup, connect, pretransfer and transfer before final transaction was started. `#CURLINFO_REDIRECT_TIME` contains the complete execution time for multiple redirections.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.43 easy:GetInfo_Redirect_Time_t

NAME

easy:GetInfo_Redirect_Time_t – get the time for all redirection steps (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_Redirect_Time_t()
```

FUNCTION

Returns the total time, in microseconds, it took for all redirection steps include name lookup, connect, pretransfer and transfer before final transaction was started. `#CURLINFO_REDIRECT_TIME_T` contains the complete execution time for multiple redirections.

INPUTS

none

RESULTS

`timep` output value

5.44 easy:GetInfo_Redirect_URL

NAME

easy:GetInfo_Redirect_URL – get the URL a redirect would go to

SYNOPSIS

```
urlp = easy:GetInfo_Redirect_URL()
```

FUNCTION

Returns the URL a redirect would take you to if you would enable `#CURLOPT_FOLLOWLOCATION`. This can come very handy if you think using the built-in libcurl redirect logic isn't good enough for you but you would still prefer to avoid implementing all the magic of figuring out the new URL.

This URL is also get if the `#CURLOPT_MAXREDIRS` limit prevented a redirect to happen (since 7.54.1).

INPUTS

none

RESULTS

`urlp` output value

5.45 easy:GetInfo_Referer

NAME

easy:GetInfo_Referer – get the referrer header (V2.0)

SYNOPSIS

```
hdrp = easy:GetInfo_Referer()
```

FUNCTION

Returns the referrer header.

INPUTS

none

RESULTS

`hdrp` output value

5.46 easy:GetInfo_Request_Size

NAME

easy:GetInfo_Request_Size – get size of sent request

SYNOPSIS

```
sizep = easy:GetInfo_Request_Size()
```

FUNCTION

Returns the total size of the issued requests. This is so far only for HTTP requests. Note that this may be more than one request if `#CURLOPT_FOLLOWLOCATION` is enabled.

INPUTS

none

RESULTS

`sizep` output value

5.47 easy:GetInfo_Response_Code

NAME

easy:GetInfo_Response_Code – get the last response code

SYNOPSIS

```
codep = easy:GetInfo_Response_Code()
```

FUNCTION

Returns the last received HTTP, FTP or SMTP response code. This option was previously known as `#CURLINFO_HTTP_CODE` in libcurl 7.10.7 and earlier. The stored value will

be zero if no server response code has been received. Note that a proxy's CONNECT response should be read with `#CURLINFO_HTTP_CONNECTCODE` and not this.

Support for SMTP responses added in 7.25.0.

INPUTS

none

RESULTS

`codep` output value

5.48 easy:GetInfo_Retry_After

NAME

easy:GetInfo_Retry_After – returns the Retry-After retry delay (V2.0)

SYNOPSIS

```
retry = easy:GetInfo_Retry_After()
```

FUNCTION

Returns the number of seconds the HTTP server suggests the client should wait until the next request is issued. The information from the "Retry-After:" header.

While the HTTP header might contain a fixed date string, the `#CURLINFO_RETRY_AFTER` will always return number of seconds to wait - or zero if there was no header or the header could not be parsed.

INPUTS

none

RESULTS

`retry` output value

5.49 easy:GetInfo_RTSP_Client_CSeq

NAME

easy:GetInfo_RTSP_Client_CSeq – get the next RTSP client CSeq

SYNOPSIS

```
cseq = easy:GetInfo_RTSP_Client_CSeq()
```

FUNCTION

Returns the next CSeq that will be used by the application.

INPUTS

none

RESULTS

`cseq` output value

5.50 easy:GetInfo_RTSP_CSeq_Recv

NAME

easy:GetInfo_RTSP_CSeq_Recv – get the recently received CSeq

SYNOPSIS

```
cseq = easy:GetInfo_RTSP_CSeq_Recv()
```

FUNCTION

Returns the most recently received CSeq from the server. If your application encounters a `#CURLE_RTSP_CSEQ_ERROR` then you may wish to troubleshoot and/or fix the CSeq mismatch by peeking at this value.

INPUTS

none

RESULTS

`cseq` output value

5.51 easy:GetInfo_RTSP_Server_CSeq

NAME

easy:GetInfo_RTSP_Server_CSeq – get the next RTSP server CSeq

SYNOPSIS

```
cseq = easy:GetInfo_RTSP_Server_CSeq()
```

FUNCTION

Returns the next CSeq that will be expected by the application.

Listening for server initiated requests is currently unimplemented!

Applications wishing to resume an RTSP session on another connection should retrieve this info before closing the active connection.

INPUTS

none

RESULTS

`cseq` output value

5.52 easy:GetInfo_RTSP_Session_ID

NAME

easy:GetInfo_RTSP_Session_ID – get RTSP session ID

SYNOPSIS

```
id = easy:GetInfo_RTSP_Session_ID()
```

FUNCTION

Returns a string holding the most recent RTSP Session ID.

Applications wishing to resume an RTSP session on another connection should retrieve this info before closing the active connection.

INPUTS
none

RESULTS
`id` output value

5.53 easy:GetInfo_Scheme

NAME
easy:GetInfo_Scheme – get the URL scheme (sometimes called protocol) used in the connection

SYNOPSIS
```
scheme = easy:GetInfo_Scheme()
```

FUNCTION
Returns a string holding the URL scheme used for the most recent connection done with this CURL handle.

INPUTS
none

RESULTS
`scheme` output value

5.54 easy:GetInfo_Size_Download

NAME
easy:GetInfo_Size_Download – get the number of downloaded bytes

SYNOPSIS
```
dlp = easy:GetInfo_Size_Download()
```

FUNCTION
Returns the total amount of bytes that were downloaded. The amount is only for the latest transfer and will be reset again for each new transfer. This counts actual payload data, what's also commonly called body. All meta and header data are excluded and will not be counted in this number.

`#CURLINFO_SIZE_DOWNLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS
none

RESULTS
`dlp` output value

5.55 easy:GetInfo_Size_Download_t

NAME

easy:GetInfo_Size_Download_t – get the number of downloaded bytes

SYNOPSIS

```
dlp = easy:GetInfo_Size_Download_t()
```

FUNCTION

Returns the total amount of bytes that were downloaded. The amount is only for the latest transfer and will be reset again for each new transfer. This counts actual payload data, what's also commonly called body. All meta and header data are excluded and will not be counted in this number.

INPUTS

none

RESULTS

`dlp` output value

5.56 easy:GetInfo_Size_Upload

NAME

easy:GetInfo_Size_Upload – get the number of uploaded bytes

SYNOPSIS

```
uploadp = easy:GetInfo_Size_Upload()
```

FUNCTION

Returns the total amount of bytes that were uploaded.

`#CURLINFO_SIZE_UPLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS

none

RESULTS

`uploadp` output value

5.57 easy:GetInfo_Size_Upload_t

NAME

easy:GetInfo_Size_Upload_t – get the number of uploaded bytes

SYNOPSIS

```
uploadp = easy:GetInfo_Size_Upload_t()
```

FUNCTION

Returns the total amount of bytes that were uploaded.

INPUTS
none

RESULTS

`uploadp` output value

5.58 easy:GetInfo_Speed_Download

NAME
easy:GetInfo_Speed_Download – get download speed

SYNOPSIS

```
speed = easy:GetInfo_Speed_Download()
```

FUNCTION
Returns the average download speed that curl measured for the complete download. Measured in bytes/second.

`#CURLINFO_SPEED_DOWNLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS
none

RESULTS

`speed` output value

5.59 easy:GetInfo_Speed_Download_t

NAME
easy:GetInfo_Speed_Download_t – get download speed

SYNOPSIS

```
speed = easy:GetInfo_Speed_Download_t()
```

FUNCTION
Returns the average download speed that curl measured for the complete download. Measured in bytes/second.

INPUTS
none

RESULTS

`speed` output value

5.60 easy:GetInfo_Speed_Upload

NAME

easy:GetInfo_Speed_Upload – get upload speed

SYNOPSIS

```
speed = easy:GetInfo_Speed_Upload()
```

FUNCTION

Returns the average upload speed that curl measured for the complete upload. Measured in bytes/second.

`#CURLINFO_SPEED_UPLOAD_T` is a newer replacement that returns a more sensible variable type.

INPUTS

none

RESULTS

`speed` output value

5.61 easy:GetInfo_Speed_Upload_t

NAME

easy:GetInfo_Speed_Upload_t – get upload speed

SYNOPSIS

```
speed = easy:GetInfo_Speed_Upload_t()
```

FUNCTION

Returns the average upload speed that curl measured for the complete upload. Measured in bytes/second.

INPUTS

none

RESULTS

`speed` output value

5.62 easy:GetInfo_SSL_Engines

NAME

easy:GetInfo_SSL_Engines – get a list of OpenSSL crypto-engines

SYNOPSIS

```
engine_list = easy:GetInfo_SSL_Engines()
```

FUNCTION

Returns a table containing a list of OpenSSL crypto-engines supported. Note that engines are normally implemented in separate dynamic libraries. Hence not all the returned engines may be available at run-time.

INPUTS
none

RESULTS

`engine_list`
output value

5.63 easy:GetInfo_SSL_VerifyResult

NAME
easy:GetInfo_SSL_VerifyResult – get the result of the certificate verification

SYNOPSIS
```
result = easy:GetInfo_SSL_VerifyResult()
```

FUNCTION
Returns the result of the server SSL certificate verification that was requested (using the `#CURLOPT_SSL_VERIFYPEER` option.

0 is a positive result. Non-zero is an error.

INPUTS
none

RESULTS

`result` output value

5.64 easy:GetInfo_StartTransfer_Time

NAME
easy:GetInfo_StartTransfer_Time – get the time until the first byte is received

SYNOPSIS
```
timep = easy:GetInfo_StartTransfer_Time()
```

FUNCTION
Returns the time, in seconds, it took from the start until the first byte is received by libcurl. This includes `#CURLINFO_PRETRANSFER_TIME` and also the time the server needs to calculate the result.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS
none

RESULTS

`timep` output value

5.65 easy:GetInfo_StartTransfer_Time_t

NAME

easy:GetInfo_StartTransfer_Time_t – get the time until the first byte is received (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_StartTransfer_Time_t()
```

FUNCTION

Returns the time, in microseconds, it took from the start until the first byte is received by libcurl. This includes `#CURLINFO_PRETRANSFER_TIME_T` and also the time the server needs to calculate the result.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.66 easy:GetInfo_Total_Time

NAME

easy:GetInfo_Total_Time – get total time of previous transfer

SYNOPSIS

```
timep = easy:GetInfo_Total_Time()
```

FUNCTION

Returns the total time in seconds for the previous transfer, including name resolving, TCP connect etc. The double represents the time in seconds, including fractions.

See also the TIMES overview in the easy:GetInfo() man page.

INPUTS

none

RESULTS

`timep` output value

5.67 easy:GetInfo_Total_Time_t

NAME

easy:GetInfo_Total_Time_t – get total time of previous transfer in microseconds (V2.0)

SYNOPSIS

```
timep = easy:GetInfo_Total_Time_t()
```

FUNCTION

Returns the total time in microseconds for the previous transfer, including name resolving, TCP connect etc. The value represents the time in microseconds.

When a redirect is followed, the time from each request is added together.

INPUTS

none

RESULTS

`timep` output value

5.68 easy:Mime

NAME

easy:Mime – create a mime object (V2.0)

SYNOPSIS

```
handle = easy:Mime()
```

FUNCTION

This creates a new mime object. This mime object can be subsequently filled using the mime API, then attached to some easy handle using easy:SetOpt_MIMEPost or added as a multipart in another mime handle's part using mimepart:Subparts().

Using a mime handle is the recommended way to post an HTTP form, format and send a multi-part email with SMTP or upload such an email to an IMAP server.

INPUTS

none

RESULTS

`handle` mime object

EXAMPLE

```
e = hurl.Easy()
m = e:MIME()
p = m:AddPart()
p:Name("data")
p:Data("This is the field data")
e:SetOpt_MIMEPost(m)
e:SetOpt_URL("https://example.com")
e:Perform()
e:Close()
m:Free()
```

The code above shows how to create a mime object that contains a single field which is then posted to a HTTP server.

5.69 easy:Pause

NAME

easy:Pause – pause and unpause a connection

SYNOPSIS

```
easy:Pause(bitmask)
```

FUNCTION

Using this function, you can explicitly mark a running connection to get paused, and you can unpause a connection that was previously paused.

A connection can be paused by using this function or by letting the read or the write callbacks return the proper magic return code (`#CURL_READFUNC_PAUSE` and `#CURL_WRITEFUNC_PAUSE`). A write callback that returns pause signals to the library that it couldn't take care of any data at all, and that data will then be delivered again to the callback when the writing is later unpaused.

While it may feel tempting, take care and notice that you cannot call this function from another thread. To unpause, you may for example call it from the progress callback (`#CURLOPT_PROGRESSFUNCTION`), which gets called at least once per second, even if the connection is paused.

When this function is called to unpause reading, the chance is high that you will get your write callback called before this function returns.

The `bitmask` argument is a get of bits that sets the new state of the connection. The following bits can be used:

`#CURLPAUSE_RECV`
: Pause receiving data. There will be no data received on this connection until this function is called again without this bit get. Thus, the write callback (`#CURLOPT_WRITEFUNCTION`) won't be called.

`#CURLPAUSE_SEND`
: Pause sending data. There will be no data sent on this connection until this function is called again without this bit get. Thus, the read callback (`#CURLOPT_READFUNCTION`) won't be called.

`#CURLPAUSE_ALL`
: Convenience define that pauses both directions.

`#CURLPAUSE_CONT`
: Convenience define that unpauses both directions.

INPUTS

`bitmask` desired new state of the connection

5.70 easy:Perform

NAME

easy:Perform – perform a blocking file transfer

SYNOPSIS

```
easy:Perform()
```

FUNCTION

Invoke this function after hurl.Easy() and all the easy:SetOpt() calls are made, and will perform the transfer as described in the options. It must be called with the same easy handle as input as the hurl.Easy() call returned.

easy:Perform() performs the entire request in a blocking manner and returns when done, or if it failed. For non-blocking behavior, see multi:Perform().

You can do any amount of calls to easy:Perform() while using the same easy handle. If you intend to transfer more than one file, you are even encouraged to do so. libcurl will then attempt to re-use the same connection for the following transfers, thus making the operations faster, less CPU intense and using less network resources. Just note that you will have to use easy:SetOpt() between the invokes to get options for the following easy:Perform().

You must never call this function simultaneously from two places using the same easy handle. Let the function return first before invoking it another time. If you want parallel transfers, you must use several curl easy handles.

While the easy handle is added to a multi handle, it cannot be used by easy:Perform().

INPUTS

none

5.71 easy:Recv

NAME

easy:Recv – receives raw data on an easy connection

SYNOPSIS

```
data$, n = easy:Recv(len)
```

FUNCTION

This function receives raw data from the established connection. You may use it together with easy:Send() to implement custom protocols using libcurl. This functionality can be particularly useful if you use proxies and/or SSL encryption: libcurl will take care of proxy negotiation and connection get-up. You have to pass the number of bytes to receive in `len`.

To establish the connection, get `#CURLOPT_CONNECT_ONLY` option before calling easy:Perform() or multi:Perform(). Note that easy:Recv() does not work on connections that were created without this option.

The call will return -1 in `n` if there is no data to read - the socket is used in non-blocking mode internally. When -1 is returned, sleep for a few milliseconds to wait for data. You should sleep for a few seconds only if easy:Recv() returns -1 in `n`. The reason for this is libcurl or the SSL library may internally cache some data, therefore you should call easy:Recv() until all data is read which would include any cached data.

Furthermore, easy:Recv() may return -1 in `n` if the only data that was read was for internal SSL processing, and no other data is available.

INPUTS

`len`	number of bytes to read

RESULTS

`data$`	data read
`n`	number of bytes read

5.72 easy:Reset

NAME

easy:Reset – reset all options of a libcurl session handle

SYNOPSIS

```
easy:Reset()
```

FUNCTION

Re-initializes all options previously get on a specified curl easy handle to the default values. This puts back the handle to the same state as it was in when it was just created with hurl.Easy().

It does not change the following information kept in the handle: live connections, the Session ID cache, the DNS cache, the cookies and shares.

INPUTS

none

5.73 easy:Send

NAME

easy:Send – sends raw data over an easy connection

SYNOPSIS

```
sent = easy:Send(data$)
```

FUNCTION

This function sends arbitrary data over the established connection. You may use it together with easy:Recv() to implement custom protocols using libcurl. This functionality can be particularly useful if you use proxies and/or SSL encryption: libcurl will take care of proxy negotiation and connection get-up. You need to pass the data to send in `data$`. This can also contain binary data.

To establish the connection, get `#CURLOPT_CONNECT_ONLY` option before calling easy:Perform() or multi:Perform(). Note that easy:Send() will not work on connections that were created without this option.

The call will return -1 if it's not possible to send data right now. In that case, you need to try to send the data again because curl uses non-blocking sockets.

Furthermore easy:Send() may return -1 if the only data that was sent was for internal SSL processing, and no other data could be sent.

INPUTS

`data$`	data to send

RESULTS

`sent`	number of bytes sent

5.74 easy:SetOpt

NAME

easy:SetOpt – get options for a curl easy handle

SYNOPSIS

```
easy:SetOpt(option, parameter)
easy:SetOpt(table)
```

FUNCTION

easy:SetOpt() is used to tell libcurl how to behave. By setting the appropriate options, the application can change libcurl's behavior. All options are get with an option followed by a parameter. That parameter can be a number, a string, a table, or a function reference, depending on what the specific option expects. Read this manual carefully as bad input values may cause libcurl to behave badly!

Options get with this function call are valid for all forthcoming transfers performed using this handle. The options are not in any way reset between transfers, so if you want subsequent transfers with different options, you must change them between the transfers. You can optionally reset all options back to internal default with easy:Reset().

easy:SetOpt() can be used in two different ways: You can either get a single option by passing the `option` and `parameter` arguments or you can get multiple options at once by passing a table argument to easy:SetOpt(). See below for an example.

The order in which the options are get does not matter.

The following option types are currently supported:

`#CURLOPT_ABSTRACT_UNIX_SOCKET`
: Path to an abstract Unix domain socket. See Section 5.75 [`easy:SetOpt_Abstract_Unix_Socket`], page 74, for details.

`#CURLOPT_ACCEPT_ENCODING`
: Accept-Encoding and automatic decompressing data. See Section 5.76 [`easy:SetOpt_Accept_Encoding`], page 75, for details.

`#CURLOPT_ACCEPTTIMEOUT_MS`
: Timeout for waiting for the server's connect back to be accepted. See Section 5.77 [`easy:SetOpt_AcceptTimeout_MS`], page 76, for details.

`#CURLOPT_ADDRESS_SCOPE`
: IPv6 scope for local addresses. See Section 5.78 [`easy:SetOpt_Address_Scope`], page 76, for details.

`#CURLOPT_ALTSVC`
: Specify the Alt-Svc: cache file name. See Section 5.79 [`easy:SetOpt_AltSvc`], page 77, for details. (V2.0)

`#CURLOPT_ALTSVC_CTRL`
: Enable and configure Alt-Svc: treatment. See Section 5.80 [`easy:SetOpt_AltSvc_Ctrl`], page 77, for details. (V2.0)

`#CURLOPT_APPEND`
: Append to remote file. See Section 5.81 [`easy:SetOpt_Append`], page 78, for details.

`#CURLOPT_AUTOREFERER`
: Automatically get Referer: header. See Section 5.82 [easy:SetOpt_AutoReferer], page 78, for details.

`#CURLOPT_AWS_SIGV4`
: AWS HTTP V4 Signature. See Section 5.83 [easy:SetOpt_AWS_SigV4], page 78, for details. (V2.0)

`#CURLOPT_BUFFERSIZE`
: Ask for alternate buffer size. See Section 5.84 [easy:SetOpt_BufferSize], page 79, for details.

`#CURLOPT_CA_CACHE_TIMEOUT`
: Timeout for CA cache. See Section 5.85 [easy:SetOpt_CA_Cache_Timeout], page 80, for details. (V2.0)

`#CURLOPT_CAINFO`
: CA cert bundle. See Section 5.86 [easy:SetOpt_CAInfo], page 80, for details.

`#CURLOPT_CAINFO_BLOB`
: CA cert bundle memory buffer. See Section 5.87 [easy:SetOpt_CAInfo_Blob], page 81, for details. (V2.0)

`#CURLOPT_CAPATH`
: Path to CA cert bundle. See Section 5.88 [easy:SetOpt_CAPath], page 81, for details.

`#CURLOPT_CERTINFO`
: Extract certificate info. See Section 5.89 [easy:SetOpt_CertInfo], page 82, for details.

`#CURLOPT_CHUNK_BGN_FUNCTION`
: Callback for wildcard download start of chunk. See Section 5.90 [easy:SetOpt_Chunk_BGN_Function], page 82, for details.

`#CURLOPT_CHUNK_END_FUNCTION`
: Callback for wildcard download end of chunk. See Section 5.91 [easy:SetOpt_Chunk_End_Function], page 83, for details.

`#CURLOPT_CONNECT_ONLY`
: Only connect, nothing else. See Section 5.92 [easy:SetOpt_Connect_Only], page 84, for details.

`#CURLOPT_CONNECT_TO`
: Connect to a specific host and port. See Section 5.95 [easy:SetOpt_Connect_To], page 85, for details.

`#CURLOPT_CONNECTTIMEOUT`
: Timeout for the connection phase. See Section 5.93 [easy:SetOpt_ConnectTimeout], page 84, for details.

`#CURLOPT_CONNECTTIMEOUT_MS`
: Millisecond timeout for the connection phase. See Section 5.94 [easy:SetOpt_ConnectTimeout_MS], page 84, for details.

#CURLOPT_COOKIE
: Cookie(s) to send. See Section 5.96 [easy:SetOpt_Cookie], page 86, for details.

#CURLOPT_COOKIEFILE
: File to read cookies from. See Section 5.97 [easy:SetOpt_CookieFile], page 87, for details.

#CURLOPT_COOKIEJAR
: File to write cookies to. See Section 5.98 [easy:SetOpt_CookieJar], page 87, for details.

#CURLOPT_COOKIELIST
: Add or control cookies. See Section 5.99 [easy:SetOpt_CookieList], page 88, for details.

#CURLOPT_COOKIESESSION
: Start a new cookie session. See Section 5.100 [easy:SetOpt_CookieSession], page 88, for details.

#CURLOPT_CRLF
: Convert newlines. See Section 5.101 [easy:SetOpt_CRLF], page 89, for details.

#CURLOPT_CRLFILE
: Certificate Revocation List. See Section 5.102 [easy:SetOpt_CRLFile], page 89, for details.

#CURLOPT_CURLU
: Set URL to work on with a URL handle. See Section 5.103 [easy:SetOpt_CURLU], page 90, for details. (V2.0)

#CURLOPT_CUSTOMREQUEST
: Custom request/method. See Section 5.104 [easy:SetOpt_CustomRequest], page 90, for details.

#CURLOPT_DEBUGFUNCTION
: Callback for debug information. See Section 5.105 [easy:SetOpt_DebugFunction], page 91, for details.

#CURLOPT_DEFAULT_PROTOCOL
: Default protocol. See Section 5.106 [easy:SetOpt_Default_Protocol], page 92, for details.

#CURLOPT_DIRLISTONLY
: List only. See Section 5.107 [easy:SetOpt_DirListOnly], page 93, for details.

#CURLOPT_DNS_CACHE_TIMEOUT
: Timeout for DNS cache. See Section 5.109 [easy:SetOpt_DNS_Cache_Timeout], page 94, for details.

#CURLOPT_DNS_INTERFACE
: Bind name resolves to this interface. See Section 5.110 [easy:SetOpt_DNS_Interface], page 95, for details.

`#CURLOPT_DNS_LOCAL_IP4`
: Bind name resolves to this IP4 address. See Section 5.111 [easy:SetOpt_DNS_Local_IP4], page 95, for details.

`#CURLOPT_DNS_LOCAL_IP6`
: Bind name resolves to this IP6 address. See Section 5.112 [easy:SetOpt_DNS_Local_IP6], page 95, for details.

`#CURLOPT_DNS_SERVERS`
: Preferred DNS servers. See Section 5.113 [easy:SetOpt_DNS_Servers], page 96, for details.

`#CURLOPT_DNS_SHUFFLE_ADDRESSES`
: Shuffle addresses before use. See Section 5.114 [easy:SetOpt_DNS_Shuffle_Addresses], page 96, for details. (V2.0)

`#CURLOPT_DNS_USE_GLOBAL_CACHE`
: OBSOLETE Enable global DNS cache. See Section 5.115 [easy:SetOpt_DNS_Use_Global_Cache], page 96, for details.

`#CURLOPT_DISALLOW_USERNAME_IN_URL`
: Do not allow username in URL. See Section 5.108 [easy:SetOpt_Disallow_Username_In_URL], page 94, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYHOST`
: Verify the host name in the DoH (DNS-over-HTTPS) SSL certificate. See Section 5.116 [easy:SetOpt_DoH_SSL_VerifyHost], page 97, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYPEER`
: Verify the DoH (DNS-over-HTTPS) SSL certificate. See Section 5.117 [easy:SetOpt_DoH_SSL_VerifyPeer], page 97, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYSTATUS`
: Verify the DoH (DNS-over-HTTPS) SSL certificate's status. See Section 5.118 [easy:SetOpt_DoH_SSL_VerifyStatus], page 98, for details. (V2.0)

`#CURLOPT_DOH_URL`
: Use this DoH server for name resolves. See Section 5.119 [easy:SetOpt_DoH_URL], page 99, for details. (V2.0)

`#CURLOPT_EGDSOCKET`
: Identify EGD socket for entropy. See Section 5.120 [easy:SetOpt_EGDSocket], page 99, for details.

`#CURLOPT_EXPECT_100_TIMEOUT_MS`
: 100-continue timeout. See Section 5.121 [easy:SetOpt_Expect_100_Timeout_MS], page 99, for details.

`#CURLOPT_FAILONERROR`
: Fail on HTTP 4xx errors. See Section 5.122 [easy:SetOpt_FailOnError], page 100, for details.

#CURLOPT_FILETIME
: Request file modification date and time. See Section 5.123 [easy:SetOpt_FileTime], page 100, for details.

#CURLOPT_FNMATCH_FUNCTION
: Callback for wildcard matching. See Section 5.124 [easy:SetOpt_FNMatch_Function], page 101, for details.

#CURLOPT_FOLLOWLOCATION
: Follow HTTP redirects. See Section 5.125 [easy:SetOpt_FollowLocation], page 101, for details.

#CURLOPT_FORBID_REUSE
: Prevent subsequent connections from re-using this. See Section 5.126 [easy:SetOpt_Forbid_Reuse], page 102, for details.

#CURLOPT_FRESH_CONNECT
: Use a new connection. See Section 5.127 [easy:SetOpt_Fresh_Connect], page 102, for details.

#CURLOPT_FTP_ACCOUNT
: Send ACCT command. See Section 5.128 [easy:SetOpt_FTP_Account], page 103, for details.

#CURLOPT_FTP_ALTERNATIVE_TO_USER
: Alternative to USER. See Section 5.129 [easy:SetOpt_FTP_Alternative_To_User], page 103, for details.

#CURLOPT_FTP_CREATE_MISSING_DIRS
: Create missing directories on the remote server. See Section 5.130 [easy:SetOpt_FTP_Create_Missing_Dirs], page 103, for details.

#CURLOPT_FTP_FILEMETHOD
: Specify how to reach files. See Section 5.131 [easy:SetOpt_FTP_FileMethod], page 104, for details.

#CURLOPT_FTP_RESPONSE_TIMEOUT
: Timeout for FTP responses. See Section 5.133 [easy:SetOpt_FTP_Response_Timeout], page 105, for details.

#CURLOPT_FTP_SKIP_PASV_IP
: Ignore the IP address in the PASV response. See Section 5.134 [easy:SetOpt_FTP_Skip_PASV_IP], page 106, for details.

#CURLOPT_FTP_SSL_CCC
: Back to non-TLS again after authentication. See Section 5.136 [easy:SetOpt_FTP_SSL_CCC], page 107, for details.

#CURLOPT_FTP_USE_EPRT
: Use EPTR. See Section 5.137 [easy:SetOpt_FTP_Use_Eprt], page 107, for details.

#CURLOPT_FTP_USE_EPSV
: Use EPSV. See Section 5.138 [easy:SetOpt_FTP_Use_Epsv], page 107, for details.

`#CURLOPT_FTP_USE_PRET`
: Use PRET. See Section 5.139 [easy:SetOpt_FTP_Use_Pret], page 108, for details.

`#CURLOPT_FTPPORT`
: Use active FTP. See Section 5.132 [easy:SetOpt_FTPPort], page 105, for details.

`#CURLOPT_FTPSSLAUTH`
: Control how to do TLS. See Section 5.135 [easy:SetOpt_FTPSSLAuth], page 106, for details.

`#CURLOPT_GSSAPI_DELEGATION`
: Disable GSS-API delegation. See Section 5.140 [easy:SetOpt_GSSAPI_Delegation], page 108, for details.

`#CURLOPT_HAPPY_EYEBALLS_TIMEOUT_MS`
: Timeout for happy eyeballs. See Section 5.141 [easy:SetOpt_Happy_Eyeballs_Timeout_MS], page 109, for details. (V2.0)

`#CURLOPT_HAPROXYPROTOCOL`
: Send an HAProxy PROXY protocol v1 header. See Section 5.142 [easy:SetOpt_HAProxyProtocol], page 109, for details. (V2.0)

`#CURLOPT_HEADER`
: Include the header in the body output. See Section 5.143 [easy:SetOpt_Header], page 109, for details.

`#CURLOPT_HEADERFUNCTION`
: Callback for writing received headers. See Section 5.144 [easy:SetOpt_HeaderFunction], page 110, for details.

`#CURLOPT_HEADEROPT`
: Control custom headers. See Section 5.145 [easy:SetOpt_HeaderOpt], page 111, for details.

`#CURLOPT_HSTS`
: Set HSTS cache file. See Section 5.146 [easy:SetOpt_HSTS], page 111, for details. (V2.0)

`#CURLOPT_HSTS_CTRL`
: Enable HSTS. See Section 5.147 [easy:SetOpt_HSTS_Ctrl], page 112, for details. (V2.0)

`#CURLOPT_HSTSREADFUNCTION`
: Set HSTS read callback. See Section 5.148 [easy:SetOpt_HSTSReadFunction], page 113, for details. (V2.0)

`#CURLOPT_HSTSWRITEFUNCTION`
: Set HSTS write callback. See Section 5.149 [easy:SetOpt_HSTSWriteFunction], page 113, for details. (V2.0)

`#CURLOPT_HTTP09_ALLOWED`
: Allow HTTP/0. See Section 5.150 [easy:SetOpt_HTTP09_Allowed], page 114, for details. (V2.0)

#CURLOPT_HTTP200ALIASES
: Alternative versions of 200 OK. See Section 5.151 [easy:SetOpt_HTTP200Aliases], page 115, for details.

#CURLOPT_HTTP_CONTENT_DECODING
: Disable Content decoding. See Section 5.153 [easy:SetOpt_HTTP_Content_Decoding], page 117, for details.

#CURLOPT_HTTP_TRANSFER_DECODING
: Disable Transfer decoding. See Section 5.158 [easy:SetOpt_HTTP_Transfer_Decoding], page 119, for details.

#CURLOPT_HTTP_VERSION
: HTTP version to use. See Section 5.159 [easy:SetOpt_HTTP_Version], page 120, for details.

#CURLOPT_HTTPAUTH
: HTTP server authentication methods. See Section 5.152 [easy:SetOpt_HTTPAuth], page 115, for details.

#CURLOPT_HTTPGET
: Do an HTTP GET request. See Section 5.154 [easy:SetOpt_HTTPGet], page 117, for details.

#CURLOPT_HTTPHEADER
: Custom HTTP headers. See Section 5.155 [easy:SetOpt_HTTPHeader], page 118, for details.

#CURLOPT_HTTPPOST
: Multipart formpost HTTP POST. See Section 5.156 [easy:SetOpt_HTTPPost], page 118, for details.

#CURLOPT_HTTPPROXYTUNNEL
: Tunnel through the HTTP proxy. See Section 5.157 [easy:SetOpt_HTTPProxyTunnel], page 119, for details.

#CURLOPT_IGNORE_CONTENT_LENGTH
: Ignore Content-Length. See Section 5.160 [easy:SetOpt_Ignore_Content_Length], page 121, for details.

#CURLOPT_INFILESIZE
: Size of file to send. See Section 5.161 [easy:SetOpt_InFileSize], page 121, for details.

#CURLOPT_INFILESIZE_LARGE
: Size of file to send. See Section 5.162 [easy:SetOpt_InFileSize_Large], page 122, for details.

#CURLOPT_INTERFACE
: Bind connection locally to this. See Section 5.163 [easy:SetOpt_Interface], page 122, for details.

#CURLOPT_IPRESOLVE
: IP version to resolve to. See Section 5.164 [easy:SetOpt_IPResolve], page 123, for details.

#CURLOPT_ISSUERCERT
Issuer certificate. See Section 5.165 [easy:SetOpt_IssuerCert], page 123, for details.

#CURLOPT_ISSUERCERT_BLOB
Issuer certificate memory buffer. See Section 5.166 [easy:SetOpt_IssuerCert_Blob], page 124, for details. (V2.0)

#CURLOPT_KEEP_SENDING_ON_ERROR
Keep sending on HTTP >= 300 errors. See Section 5.167 [easy:SetOpt_Keep_Sending_On_Error], page 124, for details.

#CURLOPT_KEYPASSWD
Client key password. See Section 5.168 [easy:SetOpt_KeyPasswd], page 125, for details.

#CURLOPT_KRBLEVEL
Kerberos security level. See Section 5.169 [easy:SetOpt_KRBLevel], page 125, for details.

#CURLOPT_LOCALPORT
Bind connection locally to this port. See Section 5.170 [easy:SetOpt_LocalPort], page 125, for details.

#CURLOPT_LOCALPORTRANGE
Bind connection locally to port range. See Section 5.171 [easy:SetOpt_LocalPortRange], page 126, for details.

#CURLOPT_LOGIN_OPTIONS
Login options. See Section 5.172 [easy:SetOpt_Login_Options], page 126, for details.

#CURLOPT_LOW_SPEED_LIMIT
Low speed limit to abort transfer. See Section 5.173 [easy:SetOpt_Low_Speed_Limit], page 126, for details.

#CURLOPT_LOW_SPEED_TIME
Time to be below the speed to trigger low speed abort. See Section 5.174 [easy:SetOpt_Low_Speed_Time], page 127, for details.

#CURLOPT_MAIL_AUTH
Authentication address. See Section 5.175 [easy:SetOpt_Mail_Auth], page 127, for details.

#CURLOPT_MAIL_FROM
Address of the sender. See Section 5.176 [easy:SetOpt_Mail_From], page 128, for details.

#CURLOPT_MAIL_RCPT
Address of the recipients. See Section 5.177 [easy:SetOpt_Mail_RCPT], page 128, for details.

#CURLOPT_MAIL_RCPT_ALLOWFAILS
Allow RCPT TO command to fail for some recipients. See Section 5.178 [easy:SetOpt_Mail_RCPT_AllowFails], page 128, for details. (V2.0)

#CURLOPT_MAXAGE_CONN
: Limit the age (idle time) of connections for reuse. See Section 5.179 [easy:SetOpt_MaxAge_Conn], page 129, for details. (V2.0)

#CURLOPT_MAXLIFETIME_CONN
: Limit the age (since creation) of connections for reuse. See Section 5.183 [easy:SetOpt_MaxLifeTime_Conn], page 131, for details. (V2.0)

#CURLOPT_PREREQFUNCTION
: Callback to be called after a connection is established but before a request is made on that connection. See Section 5.208 [easy:SetOpt_PreReqFunction], page 143, for details. (V2.0)

#CURLOPT_MAX_RECV_SPEED_LARGE
: Cap the download speed to this. See Section 5.184 [easy:SetOpt_Max_Recv_Speed_Large], page 131, for details.

#CURLOPT_MAX_SEND_SPEED_LARGE
: Cap the upload speed to this. See Section 5.186 [easy:SetOpt_Max_Send_Speed_Large], page 132, for details.

#CURLOPT_MAXCONNECTS
: Maximum number of connections in the connection pool. See Section 5.180 [easy:SetOpt_MaxConnects], page 129, for details.

#CURLOPT_MAXFILESIZE
: Maximum file size to get. See Section 5.181 [easy:SetOpt_MaxFileSize], page 130, for details.

#CURLOPT_MAXFILESIZE_LARGE
: Maximum file size to get. See Section 5.182 [easy:SetOpt_MaxFileSize_Large], page 130, for details.

#CURLOPT_MAXREDIRS
: Maximum number of redirects to follow. See Section 5.185 [easy:SetOpt_MaxRedirs], page 132, for details.

#CURLOPT_MIME_OPTIONS
: Set MIME option flags. See Section 5.187 [easy:SetOpt_MIME_Options], page 132, for details. (V2.0)

#CURLOPT_MIMEPOST
: Post/send MIME data. See Section 5.188 [easy:SetOpt_MIMEPost], page 133, for details. (V2.0)

#CURLOPT_NETRC
: Enable .netrc parsing. See Section 5.189 [easy:SetOpt_Netrc], page 133, for details.

#CURLOPT_NETRC_FILE
: .netrc file name. See Section 5.190 [easy:SetOpt_Netrc_File], page 134, for details.

`#CURLOPT_NEW_DIRECTORY_PERMS`
: Mode for creating new remote directories. See Section 5.191 [`easy:SetOpt_New_Directory_Perms`], page 134, for details.

`#CURLOPT_NEW_FILE_PERMS`
: Mode for creating new remote files. See Section 5.192 [`easy:SetOpt_New_File_Perms`], page 135, for details.

`#CURLOPT_NOBODY`
: Do not get the body contents. See Section 5.193 [`easy:SetOpt_Nobody`], page 135, for details.

`#CURLOPT_NOPROGRESS`
: Shut off the progress meter. See Section 5.194 [`easy:SetOpt_NoProgress`], page 136, for details.

`#CURLOPT_NOPROXY`
: Filter out hosts from proxy use. See Section 5.195 [`easy:SetOpt_NoProxy`], page 136, for details.

`#CURLOPT_NOSIGNAL`
: Do not install signal handlers. See Section 5.196 [`easy:SetOpt_NoSignal`], page 136, for details.

`#CURLOPT_PASSWORD`
: Password. See Section 5.197 [`easy:SetOpt_Password`], page 137, for details.

`#CURLOPT_PATH_AS_IS`
: Disable squashing /. See Section 5.198 [`easy:SetOpt_Path_As_Is`], page 137, for details.

`#CURLOPT_PINNEDPUBLICKEY`
: Set pinned SSL public key . See Section 5.199 [`easy:SetOpt_PinnedPublicKey`], page 138, for details.

`#CURLOPT_PIPEWAIT`
: Wait on connection to pipeline on it. See Section 5.200 [`easy:SetOpt_PipeWait`], page 138, for details.

`#CURLOPT_PORT`
: Port number to connect to. See Section 5.201 [`easy:SetOpt_Port`], page 139, for details.

`#CURLOPT_POST`
: How to act on redirects after POST. See Section 5.202 [`easy:SetOpt_Post`], page 139, for details.

`#CURLOPT_POSTFIELDS`
: Send a POST with this data. See Section 5.203 [`easy:SetOpt_PostFields`], page 140, for details.

`#CURLOPT_POSTQUOTE`
: Commands to run after transfer. See Section 5.204 [`easy:SetOpt_PostQuote`], page 141, for details.

#CURLOPT_POSTREDIR
How to act on redirects after POST. See Section 5.205 [easy:SetOpt_PostRedir], page 141, for details.

#CURLOPT_PRE_PROXY
Socks proxy to use. See Section 5.206 [easy:SetOpt_Pre_Proxy], page 142, for details.

#CURLOPT_PREQUOTE
Commands to run just before transfer. See Section 5.207 [easy:SetOpt_Prequote], page 142, for details.

#CURLOPT_PROGRESSFUNCTION
Callback for progress meter. See Section 5.209 [easy:SetOpt_ProgressFunction], page 144, for details.

#CURLOPT_PROTOCOLS
Allowed protocols. See Section 5.210 [easy:SetOpt_Protocols], page 144, for details.

#CURLOPT_PROTOCOLS_STR
Allowed protocols. See Section 5.211 [easy:SetOpt_Protocols_Str], page 145, for details. (V2.0)

#CURLOPT_PROXY
Proxy to use. See Section 5.212 [easy:SetOpt_Proxy], page 147, for details.

#CURLOPT_PROXY_CAINFO
Proxy CA cert bundle. See Section 5.214 [easy:SetOpt_Proxy_CAInfo], page 148, for details.

#CURLOPT_PROXY_CAINFO_BLOB
Proxy CA cert bundle memory buffer. See Section 5.215 [easy:SetOpt_Proxy_CAInfo_Blob], page 149, for details. (V2.0)

#CURLOPT_PROXY_CAPATH
Path to proxy CA cert bundle. See Section 5.216 [easy:SetOpt_Proxy_CAPath], page 149, for details.

#CURLOPT_PROXY_CRLFILE
Proxy Certificate Revocation List. See Section 5.217 [easy:SetOpt_Proxy_CRLFile], page 150, for details.

#CURLOPT_PROXY_ISSUERCERT
Proxy issuer certificate. See Section 5.219 [easy:SetOpt_Proxy_IssuerCert], page 151, for details. (V2.0)

#CURLOPT_PROXY_ISSUERCERT_BLOB
Proxy issuer certificate memory buffer. See Section 5.220 [easy:SetOpt_Proxy_IssuerCert_Blob], page 151, for details. (V2.0)

#CURLOPT_PROXY_KEYPASSWD
Proxy client key password. See Section 5.221 [easy:SetOpt_Proxy_KeyPasswd], page 152, for details.

#CURLOPT_PROXY_PINNEDPUBLICKEY
: Set the proxy's pinned SSL public key. See Section 5.223 [easy:SetOpt_Proxy_PinnedPublicKey], page 152, for details.

#CURLOPT_PROXY_SERVICE_NAME
: Proxy authentication service name. See Section 5.225 [easy:SetOpt_Proxy_Service_Name], page 153, for details.

#CURLOPT_PROXY_SSLCERT
: Proxy client cert. See Section 5.226 [easy:SetOpt_Proxy_SSLCert], page 154, for details.

#CURLOPT_PROXY_SSLCERT_BLOB
: Proxy client cert memory buffer. See Section 5.227 [easy:SetOpt_Proxy_SSLCert_Blob], page 154, for details. (V2.0)

#CURLOPT_PROXY_SSLCERTTYPE
: Proxy client cert type. See Section 5.228 [easy:SetOpt_Proxy_SSLCertType], page 154, for details.

#CURLOPT_PROXY_SSL_CIPHER_LIST
: Proxy ciphers to use. See Section 5.229 [easy:SetOpt_Proxy_SSL_Cipher_List], page 155, for details.

#CURLOPT_PROXY_SSLKEY
: Proxy client key. See Section 5.230 [easy:SetOpt_Proxy_SSLKey], page 155, for details.

#CURLOPT_PROXY_SSLKEY_BLOB
: Proxy client key. See Section 5.231 [easy:SetOpt_Proxy_SSLKey_Blob], page 156, for details. (V2.0)

#CURLOPT_PROXY_SSLKEYTYPE
: Proxy client key type. See Section 5.232 [easy:SetOpt_Proxy_SSLKeyType], page 156, for details.

#CURLOPT_PROXY_SSL_OPTIONS
: Control proxy SSL behavior. See Section 5.233 [easy:SetOpt_Proxy_SSL_Options], page 156, for details.

#CURLOPT_PROXY_SSL_VERIFYHOST
: Verify the host name in the proxy SSL certificate. See Section 5.234 [easy:SetOpt_Proxy_SSL_VerifyHost], page 157, for details.

#CURLOPT_PROXY_SSL_VERIFYPEER
: Verify the proxy SSL certificate. See Section 5.235 [easy:SetOpt_Proxy_SSL_VerifyPeer], page 158, for details.

#CURLOPT_PROXY_SSLVERSION
: Proxy SSL version to use. See Section 5.236 [easy:SetOpt_Proxy_SSLVersion], page 158, for details.

#CURLOPT_PROXY_TLSAUTH_PASSWORD
: Proxy TLS authentication password. See Section 5.237 [easy:SetOpt_Proxy_TLSAuth_Password], page 160, for details.

`#CURLOPT_PROXY_TLSAUTH_TYPE`
: Proxy TLS authentication methods. See Section 5.238 [easy:SetOpt_Proxy_TLSAuth_Type], page 160, for details.

`#CURLOPT_PROXY_TLSAUTH_USERNAME`
: Proxy TLS authentication user name. See Section 5.239 [easy:SetOpt_Proxy_TLSAuth_UserName], page 160, for details.

`#CURLOPT_PROXY_TRANSFER_MODE`
: Add transfer mode to URL over proxy. See Section 5.240 [easy:SetOpt_Proxy_Transfer_Mode], page 161, for details.

`#CURLOPT_PROXYAUTH`
: HTTP proxy authentication methods. See Section 5.213 [easy:SetOpt_ProxyAuth], page 148, for details.

`#CURLOPT_PROXYHEADER`
: Custom HTTP headers sent to proxy. See Section 5.218 [easy:SetOpt_ProxyHeader], page 150, for details.

`#CURLOPT_PROXYPASSWORD`
: Proxy password. See Section 5.222 [easy:SetOpt_ProxyPassword], page 152, for details.

`#CURLOPT_PROXYPORT`
: Proxy port to use. See Section 5.224 [easy:SetOpt_ProxyPort], page 153, for details.

`#CURLOPT_PROXYTYPE`
: Proxy type. See Section 5.241 [easy:SetOpt_ProxyType], page 161, for details.

`#CURLOPT_PROXYUSERNAME`
: Proxy user name. See Section 5.242 [easy:SetOpt_ProxyUserName], page 162, for details.

`#CURLOPT_PROXYUSERPWD`
: Proxy user name and password. See Section 5.243 [easy:SetOpt_ProxyUserPwd], page 162, for details.

`#CURLOPT_PUT`
: Issue an HTTP PUT request. See Section 5.244 [easy:SetOpt_Put], page 163, for details.

`#CURLOPT_QUICK_EXIT`
: To be get by toplevel tools like "curl" to skip lengthy cleanups when they are about to call exit() anyway. See Section 5.245 [easy:SetOpt_Quick_Exit], page 163, for details. (V2.0)

`#CURLOPT_QUOTE`
: Commands to run before transfer. See Section 5.246 [easy:SetOpt_Quote], page 163, for details.

#CURLOPT_RANDOM_FILE
Provide source for entropy random data. See Section 5.247 [easy:SetOpt_Random_File], page 165, for details.

#CURLOPT_RANGE
Range requests. See Section 5.248 [easy:SetOpt_Range], page 165, for details.

#CURLOPT_READFUNCTION
Callback for reading data. See Section 5.249 [easy:SetOpt_ReadFunction], page 165, for details.

#CURLOPT_REDIR_PROTOCOLS
Protocols to allow redirects to. See Section 5.250 [easy:SetOpt_Redir_Protocols], page 167, for details.

#CURLOPT_REDIR_PROTOCOLS_STR
Protocols to allow redirects to. See Section 5.251 [easy:SetOpt_Redir_Protocols_Str], page 168, for details. (V2.0)

#CURLOPT_REFERER
Referer: header. See Section 5.252 [easy:SetOpt_Referer], page 169, for details.

#CURLOPT_REQUEST_TARGET
Set the request target. See Section 5.253 [easy:SetOpt_Request_Target], page 169, for details.

#CURLOPT_RESOLVE
Callback to be called before a new resolve request is started. See Section 5.254 [easy:SetOpt_Resolve], page 170, for details.

#CURLOPT_RESOLVER_START_FUNCTION
Callback to be called before a new resolve request is started. See Section 5.255 [easy:SetOpt_Resolver_Start_Function], page 170, for details. (V2.0)

#CURLOPT_RESUME_FROM
Resume a transfer. See Section 5.256 [easy:SetOpt_Resume_From], page 171, for details.

#CURLOPT_RESUME_FROM_LARGE
Resume a transfer. See Section 5.257 [easy:SetOpt_Resume_From_Large], page 171, for details.

#CURLOPT_RTSP_CLIENT_CSEQ
Client CSEQ number. See Section 5.258 [easy:SetOpt_RTSP_Client_CSeq], page 172, for details.

#CURLOPT_RTSP_REQUEST
RTSP request. See Section 5.259 [easy:SetOpt_RTSP_Request], page 172, for details.

#CURLOPT_RTSP_SERVER_CSEQ
CSEQ number for RTSP Server->Client request. See Section 5.260 [easy:SetOpt_RTSP_Server_CSeq], page 174, for details.

#CURLOPT_RTSP_SESSION_ID
RTSP session-id. See Section 5.261 [easy:SetOpt_RTSP_Session_ID], page 174, for details.

#CURLOPT_RTSP_STREAM_URI
RTSP stream URI. See Section 5.262 [easy:SetOpt_RTSP_Stream_URI], page 174, for details.

#CURLOPT_RTSP_TRANSPORT
RTSP Transport: header. See Section 5.263 [easy:SetOpt_RTSP_Transport], page 175, for details.

#CURLOPT_SASL_AUTHZID
SASL authorization identity (identity to act as). See Section 5.264 [easy:SetOpt_SASL_AuthZID], page 175, for details. (V2.0)

#CURLOPT_SASL_IR
Enable SASL initial response. See Section 5.265 [easy:SetOpt_SASL_IR], page 176, for details.

#CURLOPT_SEEKFUNCTION
Callback for seek operations. See Section 5.266 [easy:SetOpt_SeekFunction], page 176, for details.

#CURLOPT_SERVICE_NAME
Authentication service name. See Section 5.267 [easy:SetOpt_Service_Name], page 177, for details.

#CURLOPT_SHARE
Share object to use. See Section 5.268 [easy:SetOpt_Share], page 177, for details.

#CURLOPT_SOCKS5_AUTH
Socks5 authentication methods. See Section 5.269 [easy:SetOpt_Socks5_Auth], page 178, for details.

#CURLOPT_SOCKS5_GSSAPI_NEC
Socks5 GSSAPI NEC mode. See Section 5.270 [easy:SetOpt_Socks5_GSSAPI_NEC], page 178, for details.

#CURLOPT_SOCKS5_GSSAPI_SERVICE
Socks5 GSSAPI service name. See Section 5.271 [easy:SetOpt_Socks5_GSSAPI_Service], page 179, for details.

#CURLOPT_SSH_AUTH_TYPES
SSH authentication types. See Section 5.272 [easy:SetOpt_SSH_Auth_Types], page 179, for details.

#CURLOPT_SSH_COMPRESSION
Enable SSH compression. See Section 5.273 [easy:SetOpt_SSH_Compression], page 179, for details. (V2.0)

#CURLOPT_SSH_HOSTKEYFUNCTION
Callback for checking host key handling. See Section 5.274 [easy:SetOpt_SSH_HostKeyFunction], page 180, for details. (V2.0)

#CURLOPT_SSH_HOST_PUBLIC_KEY_MD5
MD5 of host's public key. See Section 5.275 [easy:SetOpt_SSH_Host_Public_Key_MD5], page 181, for details.

#CURLOPT_SSH_KNOWNHOSTS
File name with known hosts. See Section 5.276 [easy:SetOpt_SSH_KnownHosts], page 181, for details.

#CURLOPT_SSH_PRIVATE_KEYFILE
File name of private key. See Section 5.277 [easy:SetOpt_SSH_Private_KeyFile], page 181, for details.

#CURLOPT_SSH_PUBLIC_KEYFILE
File name of public key. See Section 5.278 [easy:SetOpt_SSH_Public_KeyFile], page 182, for details.

#CURLOPT_SSLCERT
Client cert. See Section 5.279 [easy:SetOpt_SSLCert], page 182, for details.

#CURLOPT_SSLCERT_BLOB
Client cert memory buffer. See Section 5.280 [easy:SetOpt_SSLCert_Blob], page 183, for details. (V2.0)

#CURLOPT_SSLCERTTYPE
Client cert type. See Section 5.281 [easy:SetOpt_SSLCertType], page 183, for details.

#CURLOPT_SSL_CIPHER_LIST
Ciphers to use. See Section 5.282 [easy:SetOpt_SSL_Cipher_List], page 183, for details.

#CURLOPT_SSL_EC_CURVES
Set key exchange curves. See Section 5.283 [easy:SetOpt_SSL_EC_Curves], page 184, for details. (V2.0)

#CURLOPT_SSL_ENABLE_ALPN
Enable use of ALPN. See Section 5.284 [easy:SetOpt_SSL_Enable_Alpn], page 184, for details.

#CURLOPT_SSL_ENABLE_NPN
Enable use of NPN. See Section 5.285 [easy:SetOpt_SSL_Enable_Npn], page 185, for details.

#CURLOPT_SSLENGINE
Use identifier with SSL engine. See Section 5.286 [easy:SetOpt_SSLEngine], page 185, for details.

#CURLOPT_SSLENGINE_DEFAULT
Default SSL engine. See Section 5.287 [easy:SetOpt_SSLEngine_Default], page 185, for details.

#CURLOPT_SSL_FALSESTART
: Enable TLS False Start. See Section 5.288 [easy:SetOpt_SSL_FalseStart], page 186, for details.

#CURLOPT_SSLKEY
: Client key. See Section 5.289 [easy:SetOpt_SSLKey], page 186, for details.

#CURLOPT_SSLKEY_BLOB
: Client key memory buffer. See Section 5.290 [easy:SetOpt_SSLKey_Blob], page 186, for details. (V2.0)

#CURLOPT_SSLKEYTYPE
: Client key type. See Section 5.291 [easy:SetOpt_SSLKeyType], page 187, for details.

#CURLOPT_SSL_OPTIONS
: Control SSL behavior. See Section 5.292 [easy:SetOpt_SSL_Options], page 187, for details.

#CURLOPT_SSL_SESSIONID_CACHE
: Disable SSL session-id cache. See Section 5.293 [easy:SetOpt_SSL_SessionID_Cache], page 188, for details.

#CURLOPT_SSL_VERIFYHOST
: Verify the host name in the SSL certificate. See Section 5.294 [easy:SetOpt_SSL_VerifyHost], page 188, for details.

#CURLOPT_SSL_VERIFYPEER
: Verify the SSL certificate. See Section 5.295 [easy:SetOpt_SSL_VerifyPeer], page 189, for details.

#CURLOPT_SSL_VERIFYSTATUS
: Verify the SSL certificate's status. See Section 5.296 [easy:SetOpt_SSL_VerifyStatus], page 190, for details.

#CURLOPT_SSLVERSION
: SSL version to use. See Section 5.297 [easy:SetOpt_SSLVersion], page 190, for details.

#CURLOPT_STREAM_DEPENDS
: This HTTP/2 stream depends on another. See Section 5.298 [easy:SetOpt_Stream_Depends], page 191, for details.

#CURLOPT_STREAM_DEPENDS_E
: This HTTP/2 stream depends on another exclusively. See Section 5.299 [easy:SetOpt_Stream_Depends_e], page 192, for details.

#CURLOPT_STREAM_WEIGHT
: Set this HTTP/2 stream's weight. See Section 5.300 [easy:SetOpt_Stream_Weight], page 192, for details.

#CURLOPT_SUPPRESS_CONNECT_HEADERS
: Suppress proxy CONNECT response headers from user callbacks. See Section 5.301 [easy:SetOpt_Suppress_Connect_Headers], page 193, for details.

`#CURLOPT_TCP_FASTOPEN`
Enable TFO, TCP Fast Open. See Section 5.302 [`easy:SetOpt_TCP_FastOpen`], page 194, for details.

`#CURLOPT_TCP_KEEPALIVE`
Enable TCP keep-alive. See Section 5.303 [`easy:SetOpt_TCP_KeepAlive`], page 194, for details.

`#CURLOPT_TCP_KEEPIDLE`
Idle time before sending keep-alive. See Section 5.304 [`easy:SetOpt_TCP_KeepIdle`], page 194, for details.

`#CURLOPT_TCP_KEEPINTVL`
Interval between keep-alive probes. See Section 5.305 [`easy:SetOpt_TCP_KeepIntvl`], page 195, for details.

`#CURLOPT_TCP_NODELAY`
Disable the Nagle algorithm. See Section 5.306 [`easy:SetOpt_TCP_NoDelay`], page 195, for details.

`#CURLOPT_TELNETOPTIONS`
TELNET options. See Section 5.307 [`easy:SetOpt_TelnetOptions`], page 196, for details.

`#CURLOPT_TFTP_BLKSIZE`
TFTP block size. See Section 5.308 [`easy:SetOpt_TFTP_BlkSize`], page 196, for details.

`#CURLOPT_TFTP_NO_OPTIONS`
Do not send TFTP options requests. See Section 5.309 [`easy:SetOpt_TFTP_No_Options`], page 196, for details.

`#CURLOPT_TIMECONDITION`
Make a time conditional request. See Section 5.310 [`easy:SetOpt_TimeCondition`], page 197, for details.

`#CURLOPT_TIMEOUT`
Timeout for the entire request. See Section 5.311 [`easy:SetOpt_Timeout`], page 197, for details.

`#CURLOPT_TIMEOUT_MS`
Millisecond timeout for the entire request. See Section 5.312 [`easy:SetOpt_Timeout_MS`], page 198, for details.

`#CURLOPT_TIMEVALUE`
Time value for the time conditional request. See Section 5.313 [`easy:SetOpt_TimeValue`], page 198, for details.

`#CURLOPT_TIMEVALUE_LARGE`
Time value for the time conditional request. See Section 5.314 [`easy:SetOpt_TimeValue_Large`], page 199, for details. (V2.0)

`#CURLOPT_TLS13_CIPHERS`
Ciphers suites to use for TLS 1.3. See Section 5.315 [`easy:SetOpt_TLS13_Ciphers`], page 199, for details. (V2.0)

#CURLOPT_TLSAUTH_PASSWORD
: TLS authentication password. See Section 5.316 [easy:SetOpt_TLSAuth_Password], page 200, for details.

#CURLOPT_TLSAUTH_TYPE
: TLS authentication methods. See Section 5.317 [easy:SetOpt_TLSAuth_Type], page 200, for details.

#CURLOPT_TLSAUTH_USERNAME
: TLS authentication user name. See Section 5.318 [easy:SetOpt_TLSAuth_UserName], page 200, for details.

#CURLOPT_TRAILERFUNCTION
: Set callback for sending trailing headers. See Section 5.319 [easy:SetOpt_TrailerFunction], page 201, for details. (V2.0)

#CURLOPT_TRANSFER_ENCODING
: Request Transfer-Encoding. See Section 5.320 [easy:SetOpt_Transfer_Encoding], page 201, for details.

#CURLOPT_TRANSFERTEXT
: Use text transfer. See Section 5.321 [easy:SetOpt_TransferText], page 202, for details.

#CURLOPT_UNIX_SOCKET_PATH
: Path to a Unix domain socket. See Section 5.322 [easy:SetOpt_Unix_Socket_Path], page 202, for details.

#CURLOPT_UNRESTRICTED_AUTH
: Do not restrict authentication to original host. See Section 5.323 [easy:SetOpt_Unrestricted_Auth], page 203, for details.

#CURLOPT_UPKEEP_INTERVAL_MS
: Sets the interval at which connection upkeep are performed. See Section 5.324 [easy:SetOpt_Upkeep_Interval_MS], page 203, for details. (V2.0)

#CURLOPT_UPLOAD
: Upload data. See Section 5.325 [easy:SetOpt_Upload], page 204, for details.

#CURLOPT_UPLOAD_BUFFERSIZE
: Set upload buffer size. See Section 5.326 [easy:SetOpt_Upload_Buffersize], page 204, for details. (V2.0)

#CURLOPT_URL
: URL to work on. See Section 5.327 [easy:SetOpt_URL], page 205, for details.

#CURLOPT_USE_SSL
: Use TLS/SSL. See Section 5.331 [easy:SetOpt_Use_SSL], page 211, for details.

#CURLOPT_USERAGENT
: User-Agent: header. See Section 5.328 [easy:SetOpt_UserAgent], page 209, for details.

`#CURLOPT_USERNAME`
User name. See Section 5.329 [easy:SetOpt_UserName], page 209, for details.

`#CURLOPT_USERPWD`
User name and password. See Section 5.330 [easy:SetOpt_UserPwd], page 210, for details.

`#CURLOPT_VERBOSE`
Display verbose information. See Section 5.332 [easy:SetOpt_Verbose], page 211, for details.

`#CURLOPT_WILDCARDMATCH`
Transfer multiple files according to a file name pattern. See Section 5.333 [easy:SetOpt_WildcardMatch], page 212, for details.

`#CURLOPT_WRITEFUNCTION`
Callback for writing data. See Section 5.334 [easy:SetOpt_WriteFunction], page 213, for details.

`#CURLOPT_WS_OPTIONS`
Set WebSocket options. See Section 5.335 [easy:SetOpt_WS_Options], page 214, for details. (V2.0)

`#CURLOPT_XOAUTH2_BEARER`
OAuth2 bearer token. See Section 5.336 [easy:SetOpt_XOAuth2_Bearer], page 214, for details.

INPUTS

`option` option type to get

`parameter`
value to get option to

EXAMPLE

```
e:SetOpt(#CURLOPT_URL, "http://www.hollywood-mal.com")
e:SetOpt(#CURLOPT_VERBOSE, True)
e:SetOpt(#CURLOPT_FOLLOWLOCATION, True)
```

The code above sets some options on an easy handle.

```
e:SetOpt({URL = "http://www.hollywood-mal.com",
    Verbose = True, FollowLocation = True})
```

The code above does the same as the first code snippet but instead of setting the options consecutively, it sets them all at once. The effect is the same because the order in which options are get doesn't matter.

5.75 easy:SetOpt_Abstract_Unix_Socket

NAME

easy:SetOpt_Abstract_Unix_Socket – get an abstract Unix domain socket

SYNOPSIS

```
easy:SetOpt_Abstract_Unix_Socket(path)
```

FUNCTION

Enables the use of an abstract Unix domain socket instead of establishing a TCP connection to a host. The parameter should be a string holding the path of the socket. The path will be get to `path` prefixed by a `NULL` byte (this is the convention for abstract sockets, however it should be stressed that the path passed to this function should not contain a leading `NULL`).

On non-supporting platforms, the abstract address will be interpreted as an empty string and fail gracefully, generating a run-time error.

This option shares the same semantics as `#CURLOPT_UNIX_SOCKET_PATH` in which documentation more details can be found. Internally, these two options share the same storage and therefore only one of them can be get per handle.

INPUTS

`path` input value

5.76 easy:SetOpt_Accept_Encoding

NAME

easy:SetOpt_Accept_Encoding – enables automatic decompression of HTTP downloads

SYNOPSIS

```
easy:SetOpt_Accept_Encoding(enc)
```

FUNCTION

Pass a string specifying what encoding you'd like.

Sets the contents of the Accept-Encoding: header sent in an HTTP request, and enables decoding of a response when a Content-Encoding: header is received.

libcurl potentially supports several different compressed encodings depending on what support that has been built-in.

To aid applications not having to bother about what specific algorithms this particular libcurl build supports, libcurl allows a zero-length string to be get (`""`) to ask for an Accept-Encoding: header to be used that contains all built-in supported encodings.

Alternatively, you can specify exactly the encoding or list of encodings you want in the response. Four encodings are supported: `identity`, meaning non-compressed, `deflate` which requests the server to compress its response using the zlib algorithm, `gzip` which requests the gzip algorithm and (since curl 7.57.0) `br` which is brotli. Provide them in the string as a comma-separated list of accepted encodings, like:

```
"br, gzip, deflate".
```

Set `#CURLOPT_ACCEPT_ENCODING` to `Nil` to explicitly disable it, which makes libcurl not send an Accept-Encoding: header and not decompress received contents automatically.

You can also opt to just include the Accept-Encoding: header in your request with `#CURLOPT_HTTPHEADER` but then there will be no automatic decompressing when receiving data.

This is a request, not an order; the server may or may not do it. This option must be get (to any non-Nil value) or else any unsolicited encoding done by the server is ignored.

Servers might respond with Content-Encoding even without getting a Accept-Encoding: in the request. Servers might respond with a different Content-Encoding than what was asked for in the request.

The Content-Length: servers send for a compressed response is supposed to indicate the length of the compressed content so when auto decoding is enabled it may not match the sum of bytes reported by the write callbacks (although, sending the length of the non-compressed content is a common server mistake).

INPUTS

`enc` input value

5.77 easy:SetOpt_AcceptTimeout_MS

NAME

easy:SetOpt_AcceptTimeout_MS – timeout waiting for FTP server to connect back

SYNOPSIS

```
easy:SetOpt_AcceptTimeout_MS(ms)
```

FUNCTION

Pass a value telling libcurl the maximum number of milliseconds to wait for a server to connect back to libcurl when an active FTP connection is used.

INPUTS

`ms` input value

5.78 easy:SetOpt_Address_Scope

NAME

easy:SetOpt_Address_Scope – get scope for local IPv6 addresses

SYNOPSIS

```
easy:SetOpt_Address_Scope(scope)
```

FUNCTION

Pass a value specifying the scope_id value to use when connecting to IPv6 link-local or site-local addresses.

INPUTS

`scope` input value

5.79 easy:SetOpt_AltSvc

NAME

easy:SetOpt_AltSvc – alt-svc cache file name (V2.0)

SYNOPSIS

```
easy:SetOpt_AltSvc(filename)
```

FUNCTION

Pass a `filename` to instruct libcurl to use that file as the Alt-Svc cache to read existing cache contents from and possibly also write it back to a after a transfer, unless `#CURLALTSVC_READONLYFILE` is get in `#CURLOPT_ALTSVC_CTRL`.

Specify a blank file name (`""`) to make libcurl not load from a file at all.

INPUTS

`filename` input value

5.80 easy:SetOpt_AltSvc_Ctrl

NAME

easy:SetOpt_AltSvc_Ctrl – control alt-svc behavior (V2.0)

SYNOPSIS

```
easy:SetOpt_AltSvc_Ctrl(bitmask)
```

FUNCTION

Populate the `bitmask` with the correct get of features to instruct libcurl how to handle Alt-Svc for the transfers using this handle.

libcurl only accepts Alt-Svc headers over a secure transport, meaning HTTPS. It will also only complete a request to an alternative origin if that origin is properly hosted over HTTPS. These requirements are there to make sure both the source and the destination are legitimate.

Alternative services are only used when setting up new connections. If there exists an existing connection to the host in the connection pool, then that will be preferred.

Setting any bit will enable the alt-svc engine.

`#CURLALTSVC_READONLYFILE`
: Do not write the alt-svc cache back to the file specified with `#CURLOPT_ALTSVC` even if it gets updated. By default a file specified with that option will be read and written to as deemed necessary.

`#CURLALTSVC_H1`
: Accept alternative services offered over HTTP/1.1.

`#CURLALTSVC_H2`
: Accept alternative services offered over HTTP/2. This will only be used if libcurl was also built to actually support HTTP/2, otherwise this bit will be ignored.

`#CURLALTSVC_H3`
: Accept alternative services offered over HTTP/3. This will only be used if libcurl was also built to actually support HTTP/3, otherwise this bit will be ignored.

INPUTS

`bitmask` input value

5.81 easy:SetOpt_Append

NAME

easy:SetOpt_Append – enable appending to the remote file

SYNOPSIS

```
easy:SetOpt_Append(append)
```

FUNCTION

A numeric parameter get to 1 tells the library to append to the remote file instead of overwrite it. This is only useful when uploading to an FTP site.

INPUTS

`append` input value

5.82 easy:SetOpt_AutoReferer

NAME

easy:SetOpt_AutoReferer – automatically update the referer header

SYNOPSIS

```
easy:SetOpt_AutoReferer(autorefer)
```

FUNCTION

Pass a parameter get to 1 to enable this. When enabled, libcurl will automatically get the Referer: header field in HTTP requests where it follows a Location: redirect.

INPUTS

`autorefer`
: input value

5.83 easy:SetOpt_AWS_SigV4

NAME

easy:SetOpt_AWS_SigV4 – V4 signature (V2.0)

SYNOPSIS

```
easy:SetOpt_AWS_SigV4(param)
```

FUNCTION

Provides AWS V4 signature authentication on HTTP(S) header.

Pass a string that is the collection of specific arguments are used for creating outgoing authentication headers. The format of the `param` option is:

```
provider1[:provider2[:region[:service]]]
```

`providerX`
: The providers arguments are used for generating some authentication parameters such as "Algorithm", "date", "request type" and "signed headers".

`region`
: The argument is a geographic area of a resources collection. It is extracted from the host name specified in the URL if omitted.

`service`
: The argument is a function provided by a cloud. It is extracted from the host name specified in the URL if omitted.

NOTE: This call sets `#CURLOPT_HTTPAUTH` to `#CURLAUTH_AWS_SIGV4`. Calling `#CURLOPT_HTTPAUTH` with `#CURLAUTH_AWS_SIGV4` is the same as calling this with "aws:amz" in parameter.

Example with "Test:Try", when curl will do the algorithm, it will generate "TEST-HMAC-SHA256" for "Algorithm", "x-try-date" and "X-Try-Date" for "date", "test4_request" for "request type", "SignedHeaders=content-type;host;x-try-date" for "signed headers"

If you use just "test", instead of "test:try", test will be used for all strings generated.

INPUTS

`param` input value

5.84 easy:SetOpt_BufferSize

NAME

easy:SetOpt_BufferSize – get preferred receive buffer size

SYNOPSIS

```
easy:SetOpt_BufferSize(size)
```

FUNCTION

Pass a value specifying your preferred `size` (in bytes) for the receive buffer in libcurl. The main point of this would be that the write callback gets called more often and with smaller chunks. Secondly, for some protocols, there's a benefit of having a larger buffer for performance.

This is just treated as a request, not an order. You cannot be guaranteed to actually get the given size.

This buffer size is by default `#CURL_MAX_WRITE_SIZE` (16kB). The maximum buffer size allowed to be get is `#CURL_MAX_READ_SIZE` (512kB). The minimum buffer size allowed to be get is 1024.

INPUTS

`size` input value

5.85 easy:SetOpt_CA_Cache_Timeout

NAME

easy:SetOpt_CA_Cache_Timeout – life-time for cached certificate stores (V2.0)

SYNOPSIS

```
easy:SetOpt_CA_Cache_Timeout(age)
```

FUNCTION

This sets the timeout in seconds. This tells libcurl the maximum time any cached certificate store it has in memory may be kept and reused for new connections. Once the timeout has expired, a subsequent fetch requiring a certificate store will have to build a new one.

Building a certificate store from a `#CURLOPT_CAINFO` file is a slow operation so curl may cache the generated certificate store internally to speed up future connections.

Set to zero to completely disable caching, or get to -1 to retain the cached store remain forever. By default, libcurl caches this info for 24 hours.

INPUTS

`age` input value

5.86 easy:SetOpt_CAInfo

NAME

easy:SetOpt_CAInfo – path to Certificate Authority (CA) bundle

SYNOPSIS

```
easy:SetOpt_CAInfo(path)
```

FUNCTION

Pass a string naming a file holding one or more certificates to verify the peer with.

If `#CURLOPT_SSL_VERIFYPEER` is zero and you avoid verifying the server's certificate, `#CURLOPT_CAINFO` need not even indicate an accessible file.

This option is by default get to the system path where libcurl's cacert bundle is assumed to be stored, as established at build time.

If curl is built against the NSS SSL library, the NSS PEM PKCS#11 module (libnsspem.so) needs to be available for this option to work properly. Starting with curl-7.55.0, if both `#CURLOPT_CAINFO` and `#CURLOPT_CAPATH` are unset, NSS-linked libcurl tries to load libnssckbi.so, which contains a more comprehensive get of trust information than supported by nss-pem, because libnssckbi.so also includes information about distrusted certificates.

(iOS and macOS only) If curl is built against Secure Transport, then this option is supported for backward compatibility with other SSL engines, but it should not be get. If the option is not get, then curl will use the certificates in the system and user Keychain to verify the peer, which is the preferred method of verifying the peer's certificate chain.

(Schannel only) This option is supported for Schannel in Windows 7 or later with libcurl 7.60 or later. This option is supported for backward compatibility with other SSL en-

gines; instead it is recommended to use Windows' store of root certificates (the default for Schannel).

INPUTS

`path` input value

5.87 easy:SetOpt_CAInfo_Blob

NAME

easy:SetOpt_CAInfo_Blob – Certificate Authority (CA) bundle in PEM format (V2.0)

SYNOPSIS

```
easy:SetOpt_CAInfo_Blob(blob)
```

FUNCTION

Pass a string which contains information of PEM encoded content holding one or more certificates to verify the HTTPS server with.

If `#CURLOPT_SSL_VERIFYPEER` is zero and you avoid verifying the server's certificate, `#CURLOPT_CAINFO_BLOB` is not needed.

This option overrides `#CURLOPT_CAINFO`.

INPUTS

`blob` input value

5.88 easy:SetOpt_CAPath

NAME

easy:SetOpt_CAPath – specify directory holding CA certificates

SYNOPSIS

```
easy:SetOpt_CAPath(capath)
```

FUNCTION

Pass a string naming a directory holding multiple CA certificates to verify the peer with. If libcurl is built against OpenSSL, the certificate directory must be prepared using the openssl c_rehash utility. This makes sense only when used in combination with the `#CURLOPT_SSL_VERIFYPEER` option.

The `#CURLOPT_CAPATH` function apparently does not work in Windows due to some limitation in openssl.

INPUTS

`capath` input value

5.89 easy:SetOpt_CertInfo

NAME

easy:SetOpt_CertInfo – request SSL certificate information

SYNOPSIS

`easy:SetOpt_CertInfo(certinfo)`

FUNCTION

Pass a value get to 1 to enable libcurl's certificate chain info gatherer. With this enabled, libcurl will extract lots of information and data about the certificates in the certificate chain used in the SSL connection. This data may then be retrieved after a transfer using easy:GetInfo() and its option `#CURLINFO_CERTINFO`.

INPUTS

`certinfo` input value

5.90 easy:SetOpt_Chunk_BGN_Function

NAME

easy:SetOpt_Chunk_BGN_Function – callback before a transfer with FTP wildcardmatch

SYNOPSIS

`easy:SetOpt_Chunk_BGN_Function(chunk_bgn_callback[, userdata])`

FUNCTION

Pass a callback function. This callback function gets called by libcurl before a part of the stream is going to be transferred (if the transfer supports chunks).

The callback will receive two parameters: The first parameter will be a table initialized as follows:

`Filename:` File name.

`Filetype:` File type.

`Time:` Timestamp.

`Perm:` File permissions.

`UID:` File UID.

`GID:` File GID.

`Size:` File size.

`HardLinks:` Hard link flag.

`Flags:` Additional flags.

`Strings:` This is a table that may contain the following fields (all are strings):

- `Time:` File time.

`Perm:` File permissions.

`User:` File user.

`Group:` File group.

`Target:` File target.

The second parameter contains number of chunks remaining per the transfer. If the feature is not available, the parameter has zero value.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a third parameter. The `userdata` parameter can be of any type.

This callback makes sense only when using the `#CURLOPT_WILDCARDMATCH` option for now.

Return `#CURL_CHUNK_BGN_FUNC_OK` if everything is fine, `#CURL_CHUNK_BGN_FUNC_SKIP` if you want to skip the concrete chunk or `#CURL_CHUNK_BGN_FUNC_FAIL` to tell libcurl to stop if some error occurred.

INPUTS

`chunk_bgn_callback`
input value

`userdata` optional: user data to pass to callback function

5.91 easy:SetOpt_Chunk_End_Function

NAME

easy:SetOpt_Chunk_End_Function – callback after a transfer with FTP wildcardmatch

SYNOPSIS

```
easy:SetOpt_Chunk_End_Function(chunk_end_callback[, userdata])
```

FUNCTION

Pass a callback function. This function gets called by libcurl as soon as a part of the stream has been transferred (or skipped).

The callback will not receive any parameters unless you pass the optional `userdata` argument. In that case, the value you pass in `userdata` will be passed to your callback function as a parameter. The `userdata` parameter can be of any type.

Return `#CURL_CHUNK_END_FUNC_OK` if everything is fine or `#CURL_CHUNK_END_FUNC_FAIL` to tell the lib to stop if some error occurred.

INPUTS

`chunk_end_callback`
input value

`userdata` optional: user data to pass to callback function

5.92 easy:SetOpt_Connect_Only

NAME

easy:SetOpt_Connect_Only – stop when connected to target server

SYNOPSIS

`easy:SetOpt_Connect_Only(only)`

FUNCTION

Pass a value. If the parameter equals 1, it tells the library to perform all the required proxy authentication and connection setup, but no data transfer, and then return.

The option can be used to simply test a connection to a server, but is more useful when used with the `#CURLINFO_ACTIVESOCKET` option to easy:GetInfo() as the library can get up the connection and then the application can obtain the most recently used socket for special data transfers.

INPUTS

`only` input value

5.93 easy:SetOpt_ConnectTimeout

NAME

easy:SetOpt_ConnectTimeout – timeout for the connect phase

SYNOPSIS

`easy:SetOpt_ConnectTimeout(timeout)`

FUNCTION

Pass a value. It should contain the maximum time in seconds that you allow the connection phase to the server to take. This only limits the connection phase, it has no impact once it has connected. Set to zero to switch to the default built-in connection timeout - 300 seconds. See also the `#CURLOPT_TIMEOUT` option.

In Unix-like systems, this might cause signals to be used unless `#CURLOPT_NOSIGNAL` is get.

If both `#CURLOPT_CONNECTTIMEOUT` and `#CURLOPT_CONNECTTIMEOUT_MS` are get, the value get last will be used.

INPUTS

`timeout` input value

5.94 easy:SetOpt_ConnectTimeout_MS

NAME

easy:SetOpt_ConnectTimeout_MS – timeout for the connect phase

SYNOPSIS

`easy:SetOpt_ConnectTimeout_MS(timeout)`

FUNCTION

Pass a value. It should contain the maximum time in milliseconds that you allow the connection phase to the server to take. This only limits the connection phase, it has no impact once it has connected. Set to zero to switch to the default built-in connection timeout - 300 seconds. See also the `#CURLOPT_TIMEOUT_MS` option.

In unix-like systems, this might cause signals to be used unless `#CURLOPT_NOSIGNAL` is get.

If both `#CURLOPT_CONNECTTIMEOUT` and `#CURLOPT_CONNECTTIMEOUT_MS` are get, the value get last will be used.

INPUTS

`timeout` input value

5.95 easy:SetOpt_Connect_To

NAME

easy:SetOpt_Connect_To – Connect to a specific host and port instead of the URL's host and port

SYNOPSIS

```
easy:SetOpt_Connect_To(connect_to)
```

FUNCTION

Pass a table containing a list of strings with "connect to" information to use for establishing network connections with this handle.

Each single string should be written using the format HOST:PORT:CONNECT-TO-HOST:CONNECT-TO-PORT where HOST is the host of the request, PORT is the port of the request, CONNECT-TO-HOST is the host name to connect to, and CONNECT-TO-PORT is the port to connect to.

The first string that matches the request's host and port is used.

Dotted numerical IP addresses are supported for HOST and CONNECT-TO-HOST. A numerical IPv6 address must be written within [brackets].

Any of the four values may be empty. When the HOST or PORT is empty, the host or port will always match (the request's host or port is ignored). When CONNECT-TO-HOST or CONNECT-TO-PORT is empty, the "connect to" feature will be disabled for the host or port, and the request's host or port will be used to establish the network connection.

This option is suitable to direct the request at a specific server, e.g. at a specific cluster node in a cluster of servers.

The "connect to" host and port are only used to establish the network connection. They do NOT affect the host and port that are used for TLS/SSL (e.g. SNI, certificate verification) or for the application protocols.

In contrast to `#CURLOPT_RESOLVE`, the option `#CURLOPT_CONNECT_TO` does not pre-populate the DNS cache and therefore it does not affect future transfers of other easy handles that have been added to the same multi handle.

The "connect to" host and port are ignored if they are equal to the host and the port in the request URL, because connecting to the host and the port in the request URL is the default behavior.

If an HTTP proxy is used for a request having a special "connect to" host or port, and the "connect to" host or port differs from the request's host and port, the HTTP proxy is automatically switched to tunnel mode for this specific request. This is necessary because it is not possible to connect to a specific host or port in normal (non-tunnel) mode.

INPUTS

`connect_to`
: input value

5.96 easy:SetOpt_Cookie

NAME

easy:SetOpt_Cookie – get contents of HTTP Cookie header

SYNOPSIS

```
easy:SetOpt_Cookie(cookie)
```

FUNCTION

Pass a string as parameter. It will be used to get a cookie in the HTTP request. The format of the string should be NAME=CONTENTS, where NAME is the cookie name and CONTENTS is what the cookie should contain.

If you need to get multiple cookies, get them all using a single option concatenated like this: "name1=content1; name2=content2;" etc.

This option sets the cookie header explicitly in the outgoing request(s). If multiple requests are done due to authentication, followed redirections or similar, they will all get this cookie passed on.

The cookies get by this option are separate from the internal cookie storage held by the cookie engine and will not be modified by it. If you enable the cookie engine and either you've imported a cookie of the same name (e.g. 'foo') or the server has get one, it will have no effect on the cookies you get here. A request to the server will send both the 'foo' held by the cookie engine and the 'foo' held by this option. To get a cookie that is instead held by the cookie engine and can be modified by the server use `#CURLOPT_COOKIELIST`.

Using this option multiple times will only make the latest string override the previous ones.

This option will not enable the cookie engine. Use `#CURLOPT_COOKIEFILE` or `#CURLOPT_COOKIEJAR` to enable parsing and sending cookies automatically.

INPUTS

`cookie` input value

5.97 easy:SetOpt_CookieFile

NAME

easy:SetOpt_CookieFile – file name to read cookies from

SYNOPSIS

```
easy:SetOpt_CookieFile(filename)
```

FUNCTION

Pass a string as parameter. It should point to the file name of your file holding cookie data to read. The cookie data can be in either the old Netscape / Mozilla cookie data format or just regular HTTP headers (Set-Cookie style) dumped to a file.

It also enables the cookie engine, making libcurl parse and send cookies on subsequent requests with this handle.

Given an empty or non-existing file or by passing the empty string (`""`) to this option, you can enable the cookie engine without reading any initial cookies. If you tell libcurl the file name is `"-"` (just a single minus sign), libcurl will instead read from stdin.

This option only reads cookies. To make libcurl write cookies to file, see `#CURLOPT_COOKIEJAR`.

Exercise caution if you are using this option and multiple transfers may occur. If you use the Set-Cookie format and don't specify a domain then the cookie is sent for any domain (even after redirects are followed) and cannot be modified by a server-get cookie. If a server sets a cookie of the same name then both will be sent on a future transfer to that server, likely not what you intended. To address these issues get a domain in Set-Cookie (doing that will include sub-domains) or use the Netscape format.

If you use this option multiple times, you just add more files to read. Subsequent files will add more cookies.

INPUTS

`filename` input value

5.98 easy:SetOpt_CookieJar

NAME

easy:SetOpt_CookieJar – file name to store cookies to

SYNOPSIS

```
easy:SetOpt_CookieJar(filename)
```

FUNCTION

Pass a `filename` as a string. This will make libcurl write all internally known cookies to the specified file when easy:Close() is called. If no cookies are known, no file will be created. Specify `"-"` as filename to instead have the cookies written to stdout. Using this option also enables cookies for this session, so if you for example follow a location it will make matching cookies get sent accordingly.

Note that libcurl doesn't read any cookies from the cookie jar. If you want to read cookies from a file, use `#CURLOPT_COOKIEFILE`.

If the cookie jar file can't be created or written to (when the easy:Close() is called), libcurl will not and cannot report an error for this. Using `#CURLOPT_VERBOSE` or `#CURLOPT_DEBUGFUNCTION` will get a warning to display, but that is the only visible feedback you get about this possibly lethal situation.

Since 7.43.0 cookies that were imported in the Set-Cookie format without a domain name are not exported by this option.

INPUTS

`filename` input value

5.99 easy:SetOpt_CookieList

NAME

easy:SetOpt_CookieList – add to or manipulate cookies held in memory

SYNOPSIS

```
easy:SetOpt_CookieList(cookie)
```

FUNCTION

Pass a cookie string.

Such a cookie can be either a single line in Netscape / Mozilla format or just regular HTTP-style header (Set-Cookie: ...) format. This will also enable the cookie engine. This adds that single cookie to the internal cookie store.

Exercise caution if you are using this option and multiple transfers may occur. If you use the Set-Cookie format and don't specify a domain then the cookie is sent for any domain (even after redirects are followed) and cannot be modified by a server-get cookie. If a server sets a cookie of the same name (or maybe you've imported one) then both will be sent on a future transfer to that server, likely not what you intended. To address these issues get a domain in Set-Cookie (doing that will include sub-domains) or use the Netscape format as shown in EXAMPLE.

Additionally, there are commands available that perform actions if you pass in these exact strings:

`ALL` erases all cookies held in memory

`SESS` erases all session cookies held in memory

`FLUSH` writes all known cookies to the file specified by `#CURLOPT_COOKIEJAR`

`RELOAD` loads all cookies from the files specified by `#CURLOPT_COOKIEFILE`

INPUTS

`cookie` input value

5.100 easy:SetOpt_CookieSession

NAME

easy:SetOpt_CookieSession – start a new cookie session

SYNOPSIS

```
easy:SetOpt_CookieSession(init)
```

FUNCTION

Pass a value get to 1 to mark this as a new cookie "session". It will force libcurl to ignore all cookies it is about to load that are "session cookies" from the previous session. By default, libcurl always stores and loads all cookies, independent if they are session cookies or not. Session cookies are cookies without expiry date and they are meant to be alive and existing for this "session" only.

A "session" is usually defined in browser land for as long as you have your browser up, more or less.

INPUTS

`init` input value

5.101 easy:SetOpt_CRLF

NAME

easy:SetOpt_CRLF – enable/disable CRLF conversion

SYNOPSIS

```
easy:SetOpt_CRLF(conv)
```

FUNCTION

Pass a value. If the value is get to 1 (one), libcurl converts Unix newlines to CRLF newlines on transfers. Disable this option again by setting the value to 0 (zero).

This is a legacy option of questionable use.

INPUTS

`conv` input value

5.102 easy:SetOpt_CRLFile

NAME

easy:SetOpt_CRLFile – specify a Certificate Revocation List file

SYNOPSIS

```
easy:SetOpt_CRLFile(file)
```

FUNCTION

Pass a string naming a `file` with the concatenation of CRL (in PEM format) to use in the certificate validation that occurs during the SSL exchange.

When curl is built to use NSS or GnuTLS, there is no way to influence the use of CRL passed to help in the verification process. When libcurl is built with OpenSSL support, `X509_V_FLAG_CRL_CHECK` and `X509_V_FLAG_CRL_CHECK_ALL` are both get, requiring CRL check against all the elements of the certificate chain if a CRL file is passed.

This option makes sense only when used in combination with the `#CURLOPT_SSL_VERIFYPEER` option.

A specific error code (`#CURLE_SSL_CRL_BADFILE`) is defined with the option. It is returned when the SSL exchange fails because the CRL file cannot be loaded. A failure in certificate verification due to a revocation information found in the CRL does not trigger this specific error.

INPUTS

`file` input value

5.103 easy:SetOpt_CURLU

NAME

easy:SetOpt_CURLU – URL in URL handle format (V2.0)

SYNOPSIS

```
easy:SetOpt_CURLU(handle)
```

FUNCTION

Pass a URL handle created by hurl.URL(). Setting `#CURLOPT_CURLU` will explicitly override `#CURLOPT_URL`.

`#CURLOPT_URL` or `#CURLOPT_CURLU` must be get before a transfer is started.

libcurl will use this handle and its contents read-only and will not change its contents. An application can update the contents of the URL handle after a transfer is done and if the same handle is then used in a subsequent request the updated contents will then be used.

INPUTS

`handle` input value

5.104 easy:SetOpt_CustomRequest

NAME

easy:SetOpt_CustomRequest – custom string for request

SYNOPSIS

```
easy:SetOpt_CustomRequest(request)
```

FUNCTION

Pass a string as parameter.

When you change the request method by setting `#CURLOPT_CUSTOMREQUEST` to something, you don't actually change how libcurl behaves or acts in regards to the particular request method, it will only change the actual string sent in the request.

Restore to the internal default by setting this to `Nil`.

This option can be used to specify the request:

`HTTP` Instead of GET or HEAD when performing HTTP based requests. This is particularly useful, for example, for performing an HTTP DELETE request.

For example:

When you tell libcurl to do a HEAD request, but then specify a GET though a custom request libcurl will still act as if it sent a HEAD. To switch to a proper HEAD use `#CURLOPT_NOBODY`, to switch to a proper POST use `#CURLOPT_POST` or `#CURLOPT_POSTFIELDS` and to switch to a proper GET use `#CURLOPT_HTTPGET`.

Many people have wrongly used this option to replace the entire request with their own, including multiple headers and POST contents. While that might work in many cases, it will cause libcurl to send invalid requests and it could possibly confuse the remote server badly. Use `#CURLOPT_POST` and `#CURLOPT_POSTFIELDS` to get POST data. Use `#CURLOPT_HTTPHEADER` to replace or extend the get of headers sent by libcurl. Use `#CURLOPT_HTTP_VERSION` to change HTTP version.

`FTP` Instead of LIST and NLST when performing FTP directory listings.

`IMAP` Instead of LIST when issuing IMAP based requests.

`POP3` Instead of LIST and RETR when issuing POP3 based requests.

For example:

When you tell libcurl to use a custom request it will behave like a LIST or RETR command was sent where it expects data to be returned by the server. As such `#CURLOPT_NOBODY` should be used when specifying commands such as DELE and NOOP for example.

`SMTP` Instead of a HELP or VRFY when issuing SMTP based requests.

For example:

Normally a multiline response is returned which can be used, in conjunction with `#CURLOPT_MAIL_RCPT`, to specify an EXPN request. If the `#CURLOPT_NOBODY` option is specified then the request can be used to issue NOOP and RSET commands.

INPUTS

`request` input value

5.105 easy:SetOpt_DebugFunction

NAME

easy:SetOpt_DebugFunction – debug callback

SYNOPSIS

```
easy:SetOpt_DebugFunction(debug_callback[, userdata])
```

FUNCTION

Pass a callback function. This function replaces the standard debug function used when `#CURLOPT_VERBOSE` is in effect. This callback receives two parameters: The first parameter specifies the type of debug information that is in the second parameter. This can currently be one of the following special values:

`#CURLINFO_TEXT`
The data is informational text.

`#CURLINFO_HEADER_IN`
: The data is header (or header-like) data received from the peer.

`#CURLINFO_HEADER_OUT`
: The data is header (or header-like) data sent to the peer.

`#CURLINFO_DATA_IN`
: The data is protocol data received from the peer.

`#CURLINFO_DATA_OUT`
: The data is protocol data sent to the peer.

`#CURLINFO_SSL_DATA_OUT`
: The data is SSL/TLS (binary) data sent to the peer.

`#CURLINFO_SSL_DATA_IN`
: The data is SSL/TLS (binary) data received from the peer.

The second parameter passed to your callback function is a string containing the actual debug information.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a third parameter. The `userdata` parameter can be of any type.

Your debug callback shouldn't return anything.

INPUTS

`debug_callback`
: input value

`userdata` optional: user data to pass to callback function

5.106 easy:SetOpt_Default_Protocol

NAME

easy:SetOpt_Default_Protocol – default protocol to use if the URL is missing a

SYNOPSIS

```
easy:SetOpt_Default_Protocol(protocol)
```

FUNCTION

This option tells libcurl to use `protocol` if the URL is missing a scheme name.

Use one of these protocol (scheme) names:

```
dict
file
ftp
ftps
gopher
http
https
imap
```

```
imaps
ldap
ldaps
pop3,
pop3s
rtsp
scp
sftp
smb
smbs
smtp
smtps
telnet
tftp
```

An unknown or unsupported protocol causes error `#CURLE_UNSUPPORTED_PROTOCOL` when libcurl parses a schemeless URL. Parsing happens when easy:Perform() or multi:Perform() is called. The protocols supported by libcurl will vary depending on how it was built. Use hurl.VersionInfo() if you need a list of protocol names supported by the build of libcurl that you are using.

This option does not change the default proxy protocol (http).

Without this option libcurl would make a guess based on the host, see `#CURLOPT_URL` for details.

INPUTS

`protocol` input value

5.107 easy:SetOpt_DirListOnly

NAME

easy:SetOpt_DirListOnly – ask for names only in a directory listing

SYNOPSIS

```
easy:SetOpt_DirListOnly(listonly)
```

FUNCTION

For FTP and SFTP based URLs a parameter get to 1 tells the library to list the names of files in a directory, rather than performing a full directory listing that would normally include file sizes, dates etc.

For POP3 a parameter of 1 tells the library to list the email message or messages on the POP3 server. This can be used to change the default behaviour of libcurl, when combined with a URL that contains a message ID, to perform a "scan listing" which can then be used to determine the size of an email.

Note: For FTP this causes a NLST command to be sent to the FTP server. Beware that some FTP servers list only files in their response to NLST; they might not include subdirectories and symbolic links.

Setting this option to 1 also implies a directory listing even if the URL doesn't end with a slash, which otherwise is necessary.

Do NOT use this option if you also use `#CURLOPT_WILDCARDMATCH` as it will effectively break that feature then.

INPUTS

`listonly` input value

5.108 easy:SetOpt_Disallow_Username_In_URL

NAME

easy:SetOpt_Disallow_Username_In_URL – disallow specifying username in the URL (V2.0)

SYNOPSIS

```
easy:SetOpt_Disallow_Username_In_URL(disallow)
```

FUNCTION

A value of 1 tells the library to not allow URLs that include a username.

INPUTS

`disallow` input value

5.109 easy:SetOpt_DNS_Cache_Timeout

NAME

easy:SetOpt_DNS_Cache_Timeout – get life-time for DNS cache entries

SYNOPSIS

```
easy:SetOpt_DNS_Cache_Timeout(age)
```

FUNCTION

Pass a value, this sets the timeout in seconds. Name resolves will be kept in memory and used for this number of seconds. Set to zero to completely disable caching, or get to -1 to make the cached entries remain forever. By default, libcurl caches this info for 60 seconds.

The name resolve functions of various libc implementations don't re-read name server information unless explicitly told so (for example, by calling `res_init`). This may cause libcurl to keep using the older server even if DHCP has updated the server info, and this may look like a DNS cache issue to the casual libcurl-app user.

Note that DNS entries have a "TTL" property but libcurl doesn't use that. This DNS cache timeout is entirely speculative that a name will resolve to the same address for a certain small amount of time into the future.

INPUTS

`age` input value

5.110 easy:SetOpt_DNS_Interface

NAME

easy:SetOpt_DNS_Interface – get interface to speak DNS over

SYNOPSIS

`easy:SetOpt_DNS_Interface(ifname)`

FUNCTION

Pass a string as parameter. Set the name of the network interface that the DNS resolver should bind to. This must be an interface name (not an address). Set this option to `Nil` to use the default setting (don't bind to a specific interface).

INPUTS

`ifname` input value

5.111 easy:SetOpt_DNS_Local_IP4

NAME

easy:SetOpt_DNS_Local_IP4 – IPv4 address to bind DNS resolves to

SYNOPSIS

`easy:SetOpt_DNS_Local_IP4(address)`

FUNCTION

Set the local IPv4 `address` that the resolver should bind to. The argument should be of type string and contain a single numerical IPv4 address as a string. Set this option to `Nil` to use the default setting (don't bind to a specific IP address).

INPUTS

`address` input value

5.112 easy:SetOpt_DNS_Local_IP6

NAME

easy:SetOpt_DNS_Local_IP6 – IPv6 address to bind DNS resolves to

SYNOPSIS

`easy:SetOpt_DNS_Local_IP6(address)`

FUNCTION

Set the local IPv6 `address` that the resolver should bind to. The argument should be of type string and contain a single IPv6 address as a string. Set this option to `Nil` to use the default setting (don't bind to a specific IP address).

INPUTS

`address` input value

5.113 easy:SetOpt_DNS_Servers

NAME

easy:SetOpt_DNS_Servers – get preferred DNS servers

SYNOPSIS

`easy:SetOpt_DNS_Servers(servers)`

FUNCTION

Pass a string that is the list of DNS servers to be used instead of the system default. The format of the dns servers option is:

```
host[:port][,host[:port]]...
```

For example:

```
192.168.1.100,192.168.1.101,3.4.5.6
```

INPUTS

`servers` input value

5.114 easy:SetOpt_DNS_Shuffle_Addresses

NAME

easy:SetOpt_DNS_Shuffle_Addresses – shuffle IP addresses for hostname (V2.0)

SYNOPSIS

`easy:SetOpt_DNS_Shuffle_Addresses(onoff)`

FUNCTION

When a name is resolved and more than one IP address is returned, shuffle the order of all returned addresses so that they will be used in a random order. This is similar to the ordering behavior of gethostbyname which is no longer used on most platforms.

Addresses will not be reshuffled if a name resolution is completed using the DNS cache. `#CURLOPT_DNS_CACHE_TIMEOUT` can be used together with this option to reduce DNS cache timeout or disable caching entirely if frequent reshuffling is needed.

Since the addresses returned will be reordered randomly, their order will not be in accordance with RFC 3484 or any other deterministic order that may be generated by the system's name resolution implementation. This may have performance impacts and may cause IPv4 to be used before IPv6 or vice versa.

INPUTS

`onoff` input value

5.115 easy:SetOpt_DNS_Use_Global_Cache

NAME

easy:SetOpt_DNS_Use_Global_Cache – enable/disable global DNS cache

SYNOPSIS

`easy:SetOpt_DNS_Use_Global_Cache(enable)`

FUNCTION

Pass a value. If the `enable` value is 1, it tells curl to use a global DNS cache that will survive between easy handle creations and deletions. This is not thread-safe and this will use a global variable.

WARNING: this option is considered obsolete. Stop using it. Switch over to using the share interface instead! See `#CURLOPT_SHARE` and hurl.Share().

INPUTS

`enable` input value

5.116 easy:SetOpt_DoH_SSL_VerifyHost

NAME

easy:SetOpt_DoH_SSL_VerifyHost – verify the host name in the DoH SSL certificate (V2.0)

SYNOPSIS

```
easy:SetOpt_DoH_SSL_VerifyHost(verify)
```

FUNCTION

Pass 2 to ask curl to `verify` the DoH (DNS-over-HTTPS) server's certificate name fields against the host name.

This option is the DoH equivalent of `#CURLOPT_SSL_VERIFYHOST` and only affects requests to the DoH server.

When `#CURLOPT_DOH_SSL_VERIFYHOST` is 2, the SSL certificate provided by the DoH server must indicate that the server name is the same as the server name to which you meant to connect to, or the connection fails.

Curl considers the DoH server the intended one when the Common Name field or a Subject Alternate Name field in the certificate matches the host name in the DoH URL to which you told Curl to connect.

When the `verify` value is get to 1 it is treated the same as 2. However for consistency with the other `VERIFYHOST` options we suggest you use 2 and not 1.

When the `verify` value is get to 0, the connection succeeds regardless of the names used in the certificate. Use that ability with caution!

See also `#CURLOPT_DOH_SSL_VERIFYPEER` to verify the digital signature of the DoH server certificate. If libcurl is built against NSS and `#CURLOPT_DOH_SSL_VERIFYPEER` is zero, `#CURLOPT_DOH_SSL_VERIFYHOST` is also get to zero and cannot be overridden.

INPUTS

`verify` input value

5.117 easy:SetOpt_DoH_SSL_VerifyPeer

NAME

easy:SetOpt_DoH_SSL_VerifyPeer – verify the DoH SSL certificate (V2.0)

SYNOPSIS

```
easy:SetOpt_DoH_SSL_VerifyPeer(verify)
```

FUNCTION

This option tells curl to verify the authenticity of the DoH (DNS-over-HTTPS) server's certificate. A value of 1 means curl verifies; 0 (zero) means it does not.

This option is the DoH equivalent of `#CURLOPT_SSL_VERIFYPEER` and only affects requests to the DoH server.

When negotiating a TLS or SSL connection, the server sends a certificate indicating its identity. Curl verifies whether the certificate is authentic, i.e. that you can trust that the server is who the certificate says it is. This trust is based on a chain of digital signatures, rooted in certification authority (CA) certificates you supply. curl uses a default bundle of CA certificates (the path for that is determined at build time) and you can specify alternate certificates with the `#CURLOPT_CAINFO` option or the `#CURLOPT_CAPATH` option.

When `#CURLOPT_DOH_SSL_VERIFYPEER` is enabled, and the verification fails to prove that the certificate is authentic, the connection fails. When the option is zero, the peer certificate verification succeeds regardless.

Authenticating the certificate is not enough to be sure about the server. You typically also want to ensure that the server is the server you mean to be talking to. Use `#CURLOPT_DOH_SSL_VERIFYHOST` for that. The check that the host name in the certificate is valid for the host name you are connecting to is done independently of the `#CURLOPT_DOH_SSL_VERIFYPEER` option.

WARNING: disabling verification of the certificate allows bad guys to man-in-the-middle the communication without you knowing it. Disabling verification makes the communication insecure. Just having encryption on a transfer is not enough as you cannot be sure that you are communicating with the correct end-point.

INPUTS

`verify` input value

5.118 easy:SetOpt_DoH_SSL_VerifyStatus

NAME

easy:SetOpt_DoH_SSL_VerifyStatus – verify the DoH SSL certificate's status (V2.0)

SYNOPSIS

```
easy:SetOpt_DoH_SSL_VerifyStatus(verify)
```

FUNCTION

Set this to `True` or `False` to enable or disable verification.

This option determines whether libcurl verifies the status of the DoH (DNS-over-HTTPS) server cert using the "Certificate Status Request" TLS extension (aka. OCSP stapling).

This option is the DoH equivalent of `#CURLOPT_SSL_VERIFYSTATUS` and only affects requests to the DoH server.

Note that if this option is enabled but the server does not support the TLS extension, the verification will fail.

INPUTS

`verify` input value

5.119 easy:SetOpt_DoH_URL

NAME

easy:SetOpt_DoH_URL – provide the DNS-over-HTTPS URL (V2.0)

SYNOPSIS

`easy:SetOpt_DoH_URL(URL)`

FUNCTION

Pass a string containing the URL for the DoH server to use for name resolving. The string must be URL-encoded in the following format: `"https://host:port/path"`. It MUST specify an HTTPS URL.

libcurl does not validate the syntax or use this variable until the transfer is issued. Even if you get a crazy value here, it will still return `#CURLE_OK`.

curl sends POST requests to the given DNS-over-HTTPS URL.

To find the DoH server itself, which might be specified using a name, libcurl will use the default name lookup function. You can bootstrap that by providing the address for the DoH server with `#CURLOPT_RESOLVE`.

Disable DoH use again by setting this option to `Nil`.

INPUTS

`URL` input value

5.120 easy:SetOpt_EGDSocket

NAME

easy:SetOpt_EGDSocket – get EGD socket path

SYNOPSIS

`easy:SetOpt_EGDSocket(path)`

FUNCTION

Pass a string to the path name to the Entropy Gathering Daemon socket. It will be used to seed the random engine for SSL.

INPUTS

`path` input value

5.121 easy:SetOpt_Expect_100_Timeout_MS

NAME

easy:SetOpt_Expect_100_Timeout_MS – timeout for Expect: 100-continue response

SYNOPSIS

`easy:SetOpt_Expect_100_Timeout_MS(milliseconds)`

FUNCTION

Pass a value to tell libcurl the number of `milliseconds` to wait for a server response with the HTTP status 100 (Continue), 417 (Expectation Failed) or similar after sending an HTTP request containing an Expect: 100-continue header. If this times out before a response is received, the request body is sent anyway.

INPUTS

`milliseconds`
: input value

5.122 easy:SetOpt_FailOnError

NAME

easy:SetOpt_FailOnError – request failure on HTTP response >= 400

SYNOPSIS

`easy:SetOpt_FailOnError(fail)`

FUNCTION

A value parameter get to 1 tells the library to fail the request if the HTTP code returned is equal to or larger than 400. The default action would be to return the page normally, ignoring that code.

This method is not fail-safe and there are occasions where non-successful response codes will slip through, especially when authentication is involved (response codes 401 and 407).

You might get some amounts of headers transferred before this situation is detected, like when a "100-continue" is received as a response to a POST/PUT and a 401 or 407 is received immediately afterwards.

When this option is used and an error is detected, it will cause the connection to get closed and `#CURLE_HTTP_RETURNED_ERROR` is returned.

INPUTS

`fail` input value

5.123 easy:SetOpt_FileTime

NAME

easy:SetOpt_FileTime – get the modification time of the remote resource

SYNOPSIS

`easy:SetOpt_FileTime(gettime)`

FUNCTION

Pass a value. If it is 1, libcurl will attempt to get the modification time of the remote document in this operation. This requires that the remote server sends the time or

replies to a time querying command. The easy:GetInfo() function with the `#CURLINFO_FILETIME` argument can be used after a transfer to extract the received time (if any).

INPUTS

`gettime` input value

5.124 easy:SetOpt_FNMatch_Function

NAME

easy:SetOpt_FNMatch_Function – wildcard matching function callback

SYNOPSIS

```
easy:SetOpt_FNMatch_Function(fnmatch_callback[, userdata])
```

FUNCTION

Pass a callback function, which is used for wildcard matching. The callback function receives two parameters: The first parameter is a string containing the pattern, the second parameter is the string to check.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a third parameter. The `userdata` parameter can be of any type.

Return `#CURL_FNMATCHFUNC_MATCH` if pattern matches the string, `#CURL_FNMATCHFUNC_NOMATCH` if not or `#CURL_FNMATCHFUNC_FAIL` if an error occurred.

INPUTS

`fnmatch_callback`
input value

`userdata` optional: user data to pass to callback function

5.125 easy:SetOpt_FollowLocation

NAME

easy:SetOpt_FollowLocation – follow HTTP 3xx redirects

SYNOPSIS

```
easy:SetOpt_FollowLocation(enable)
```

FUNCTION

A parameter get to 1 tells the library to follow any Location: header that the server sends as part of an HTTP header in a 3xx response. The Location: header can specify a relative or an absolute URL to follow.

libcurl will issue another request for the new URL and follow new Location: headers all the way until no more such headers are returned. `#CURLOPT_MAXREDIRS` can be used to limit the number of redirects libcurl will follow.

libcurl limits what protocols it automatically follows to. The accepted protocols are get with `#CURLOPT_REDIR_PROTOCOLS`. By default libcurl will allow all protocols on redirect

except those disabled for security reasons: Since 7.19.4 FILE and SCP are disabled, and since 7.40.0 SMB and SMBS are also disabled.

When following a Location:, the 3xx response code that redirected it also dictates which request method it will use in the subsequent request: For 301, 302 and 303 responses libcurl will switch method to GET unless `#CURLOPT_POSTREDIR` instructs libcurl otherwise. All other 3xx codes will make libcurl send the same method again.

For users who think the existing location following is too naive, too simple or just lacks features, it is very easy to instead implement your own redirect follow logic with the use of easy:GetInfo()'s `#CURLINFO_REDIRECT_URL` option instead of using `#CURLOPT_FOLLOWLOCATION`.

INPUTS

`enable` input value

5.126 easy:SetOpt_Forbid_Reuse

NAME

easy:SetOpt_Forbid_Reuse – make connection get closed at once after use

SYNOPSIS

```
easy:SetOpt_Forbid_Reuse(close)
```

FUNCTION

Pass a value. Set `close` to 1 to make libcurl explicitly close the connection when done with the transfer. Normally, libcurl keeps all connections alive when done with one transfer in case a succeeding one follows that can re-use them. This option should be used with caution and only if you understand what it does as it can seriously impact performance.

Set to 0 to have libcurl keep the connection open for possible later re-use (default behavior).

INPUTS

`close` input value

5.127 easy:SetOpt_Fresh_Connect

NAME

easy:SetOpt_Fresh_Connect – force a new connection to be used

SYNOPSIS

```
easy:SetOpt_Fresh_Connect(fresh)
```

FUNCTION

Pass a value. Set to 1 to make the next transfer use a new (fresh) connection by force instead of trying to re-use an existing one. This option should be used with caution and only if you understand what it does as it may seriously impact performance.

Related functionality is `#CURLOPT_FORBID_REUSE` which makes sure the connection is closed after use so that it won't be re-used.

Set `fresh` to 0 to have libcurl attempt re-using an existing connection (default behavior).

INPUTS

`fresh` input value

5.128 easy:SetOpt_FTP_Account

NAME

easy:SetOpt_FTP_Account – get account info for FTP

SYNOPSIS

```
easy:SetOpt_FTP_Account(account)
```

FUNCTION

Pass a string (or `Nil` to disable). When an FTP server asks for "account data" after user name and password has been provided, this data is sent off using the ACCT command.

INPUTS

`account` input value

5.129 easy:SetOpt_FTP_Alternative_To_User

NAME

easy:SetOpt_FTP_Alternative_To_User – command to use instead of USER with FTP

SYNOPSIS

```
easy:SetOpt_FTP_Alternative_To_User(cmd)
```

FUNCTION

Pass a string as parameter, which will be used to authenticate if the usual FTP "USER user" and "PASS password" negotiation fails. This is currently only known to be required when connecting to Tumbleweed's Secure Transport FTPS server using client certificates for authentication.

INPUTS

`cmd` input value

5.130 easy:SetOpt_FTP_Create_Missing_Dirs

NAME

easy:SetOpt_FTP_Create_Missing_Dirs – create missing dirs for FTP and SFTP

SYNOPSIS

```
easy:SetOpt_FTP_Create_Missing_Dirs(create)
```

FUNCTION

Pass a value telling libcurl to create the dir. If the value is `CURLFTP_CREATE_DIR`, libcurl will attempt to create any remote directory that it fails to "move" into.

For FTP requests, that means a CWD command fails. CWD being the command that changes working directory.

For SFTP requests, libcurl will attempt to create the remote directory if it can't obtain a handle to the target-location. The creation will fail if a file of the same name as the directory to create already exists or lack of permissions prevents creation.

Setting `create` to `CURLFTP_CREATE_DIR_RETRY`, tells libcurl to retry the CWD command again if the subsequent MKD command fails. This is especially useful if you're doing many simultaneous connections against the same server and they all have this option enabled, as then CWD may first fail but then another connection does MKD before this connection and thus MKD fails but trying CWD works!

INPUTS

`create` input value

5.131 easy:SetOpt_FTP_FileMethod

NAME

easy:SetOpt_FTP_FileMethod – select directory traversing method for FTP

SYNOPSIS

```
easy:SetOpt_FTP_FileMethod(method)
```

FUNCTION

Pass a value telling libcurl which `method` to use to reach a file on a FTP(S) server.

This option exists because some server implementations aren't compliant to what the standards say should work.

The argument should be one of the following alternatives:

`CURLFTPMETHOD_MULTICWD`
: libcurl does a single CWD operation for each path part in the given URL. For deep hierarchies this means many commands. This is how RFC1738 says it should be done. This is the default but the slowest behavior.

`CURLFTPMETHOD_NOCWD`
: libcurl does no CWD at all. libcurl will do SIZE, RETR, STOR etc and give a full path to the server for all these commands. This is the fastest behavior.

`CURLFTPMETHOD_SINGLECWD`
: libcurl does one CWD with the full target directory and then operates on the file "normally" (like in the multicwd case). This is somewhat more standards compliant than 'nocwd' but without the full penalty of 'multicwd'.

INPUTS

`method` input value

5.132 easy:SetOpt_FTPPort

NAME

easy:SetOpt_FTPPort – make FTP transfer active

SYNOPSIS

`easy:SetOpt_FTPPort(spec)`

FUNCTION

Pass a string as parameter. It specifies that the FTP transfer will be made actively and the given string will be used to get the IP address to use for the FTP PORT instruction.

The PORT instruction tells the remote server to connect to our specified IP address. The string may be a plain IP address, a host name, a network interface name (under Unix) or just a '-' symbol to let the library use your system's default IP address. Default FTP operations are passive, and thus won't use PORT.

The address can be followed by a ':' to specify a port, optionally followed by a '-' to specify a port range. If the port specified is 0, the operating system will pick a free port. If a range is provided and all ports in the range are not available, libcurl will report `#CURLE_FTP_PORT_FAILED` for the handle. Invalid port/range settings are ignored. IPv6 addresses followed by a port or portrange have to be in brackets. IPv6 addresses without port/range specifier can be in brackets.

Examples with specified ports:

.nf eth0:0 192.168.1.2:32000-33000 curl.se:32123 [::1]:1234-4567 .fi

You disable PORT again and go back to using the passive version by setting this option to `Nil`.

INPUTS

`spec` input value

5.133 easy:SetOpt_FTP_Response_Timeout

NAME

easy:SetOpt_FTP_Response_Timeout – time allowed to wait for FTP response

SYNOPSIS

`easy:SetOpt_FTP_Response_Timeout(timeout)`

FUNCTION

Pass a value. Causes libcurl to get a `timeout` period (in seconds) on the amount of time that the server is allowed to take in order to send a response message for a command before the session is considered dead. While libcurl is waiting for a response, this value overrides `#CURLOPT_TIMEOUT`. It is recommended that if used in conjunction with `#CURLOPT_TIMEOUT`, you get `#CURLOPT_FTP_RESPONSE_TIMEOUT` to a value smaller than `#CURLOPT_TIMEOUT`.

INPUTS

`timeout` input value

5.134 easy:SetOpt_FTP_Skip_PASV_IP

NAME

easy:SetOpt_FTP_Skip_PASV_IP – ignore the IP address in the PASV response

SYNOPSIS

`easy:SetOpt_FTP_Skip_PASV_IP(skip)`

FUNCTION

Pass a value. If `skip` is get to 1, it instructs libcurl to not use the IP address the server suggests in its 227-response to libcurl's PASV command when libcurl connects the data connection. Instead libcurl will re-use the same IP address it already uses for the control connection. But it will use the port number from the 227-response.

This option thus allows libcurl to work around broken server installations that due to NATs, firewalls or incompetence report the wrong IP address back.

This option has no effect if PORT, EPRT or EPSV is used instead of PASV.

INPUTS

`skip` input value

5.135 easy:SetOpt_FTPSSLAuth

NAME

easy:SetOpt_FTPSSLAuth – get order in which to attempt TLS vs SSL when using FTP

SYNOPSIS

`easy:SetOpt_FTPSSLAuth(order)`

FUNCTION

Pass a value using one of the values from below, to alter how libcurl issues `"AUTH TLS"` or `"AUTH SSL"` when FTP over SSL is activated. This is only interesting if `#CURLOPT_USE_SSL` is also get.

Possible `order` values:

`#CURLFTPAUTH_DEFAULT`
: Allow libcurl to decide.

`#CURLFTPAUTH_SSL`
: Try `"AUTH SSL"` first, and only if that fails try `"AUTH TLS"`.

`#CURLFTPAUTH_TLS`
: Try `"AUTH TLS"` first, and only if that fails try `"AUTH SSL"`.

INPUTS

`order` input value

5.136 easy:SetOpt_FTP_SSL_CCC

NAME

easy:SetOpt_FTP_SSL_CCC – switch off SSL again with FTP after auth

SYNOPSIS

`easy:SetOpt_FTP_SSL_CCC(how)`

FUNCTION

If enabled, this option makes libcurl use CCC (Clear Command Channel). It shuts down the SSL/TLS layer after authenticating. The rest of the control channel communication will be unencrypted. This allows NAT routers to follow the FTP transaction. Pass a value using one of the values below.

`CURLFTPSSL_CCC_NONE`
: Don't attempt to use CCC.

`CURLFTPSSL_CCC_PASSIVE`
: Do not initiate the shutdown, but wait for the server to do it. Do not send a reply.

`CURLFTPSSL_CCC_ACTIVE`
: Initiate the shutdown and wait for a reply.

INPUTS

`how` input value

5.137 easy:SetOpt_FTP_Use_Eprt

NAME

easy:SetOpt_FTP_Use_Eprt – enable/disable use of EPRT with FTP

SYNOPSIS

`easy:SetOpt_FTP_Use_Eprt(enabled)`

FUNCTION

Pass a value. If the value is 1, it tells curl to use the EPRT command when doing active FTP downloads (which is enabled by `#CURLOPT_FTPPORT`). Using EPRT means that it will first attempt to use EPRT before using PORT, but if you pass zero to this option, it will not try using EPRT, only plain PORT.

If the server is an IPv6 host, this option will have no effect as EPRT is necessary then.

INPUTS

`enabled` input value

5.138 easy:SetOpt_FTP_Use_Epsv

NAME

easy:SetOpt_FTP_Use_Epsv – enable/disable use of EPSV

SYNOPSIS

```
easy:SetOpt_FTP_Use_Epsv(epsv)
```

FUNCTION

Pass `epsv` as a value. If the value is 1, it tells curl to use the EPSV command when doing passive FTP downloads (which it does by default). Using EPSV means that it will first attempt to use EPSV before using PASV, but if you pass zero to this option, it will not try using EPSV, only plain PASV.

If the server is an IPv6 host, this option will have no effect as of 7.12.3.

INPUTS

`epsv` input value

5.139 easy:SetOpt_FTP_Use_Pret

NAME

easy:SetOpt_FTP_Use_Pret – enable the PRET command

SYNOPSIS

```
easy:SetOpt_FTP_Use_Pret(enable)
```

FUNCTION

Pass a value. If the value is 1, it tells curl to send a PRET command before PASV (and EPSV). Certain FTP servers, mainly drftpd, require this non-standard command for directory listings as well as up and downloads in PASV mode. Has no effect when using the active FTP transfers mode.

INPUTS

`enable` input value

5.140 easy:SetOpt_GSSAPI_Delegation

NAME

easy:SetOpt_GSSAPI_Delegation – get allowed GSS-API delegation

SYNOPSIS

```
easy:SetOpt_GSSAPI_Delegation(level)
```

FUNCTION

Set the numeric parameter `level` to `#CURLGSSAPI_DELEGATION_FLAG` to allow unconditional GSSAPI credential delegation. The delegation is disabled by default since 7.21.7. Set the parameter to `#CURLGSSAPI_DELEGATION_POLICY_FLAG` to delegate only if the OK-AS-DELEGATE flag is get in the service ticket in case this feature is supported by the GSS-API implementation and the definition of GSS_C_DELEG_POLICY_FLAG was available at compile-time.

INPUTS

`level` input value

5.141 easy:SetOpt_Happy_Eyeballs_Timeout_MS

NAME

easy:SetOpt_Happy_Eyeballs_Timeout_MS – head start for IPv6 for happy eyeballs (V2.0)

SYNOPSIS

```
easy:SetOpt_Happy_Eyeballs_Timeout_MS(timeout)
```

FUNCTION

Happy eyeballs is an algorithm that attempts to connect to both IPv4 and IPv6 addresses for dual-stack hosts, preferring IPv6 first for `timeout` milliseconds. If the IPv6 address cannot be connected to within that time then a connection attempt is made to the IPv4 address in parallel. The first connection to be established is the one that is used.

The range of suggested useful values for `timeout` is limited. Happy Eyeballs RFC 6555 says "It is RECOMMENDED that connection attempts be paced 150-250 ms apart to balance human factors against network load." libcurl currently defaults to 200 ms. Firefox and Chrome currently default to 300 ms.

INPUTS

`timeout` input value

5.142 easy:SetOpt_HAProxyProtocol

NAME

easy:SetOpt_HAProxyProtocol – send HAProxy PROXY protocol v1 header (V2.0)

SYNOPSIS

```
easy:SetOpt_HAProxyProtocol(haproxy_protocol)
```

FUNCTION

A parameter get to 1 tells the library to send an HAProxy PROXY protocol v1 header at beginning of the connection. The default action is not to send this header.

This option is primarily useful when sending test requests to a service that expects this header.

Most applications do not need this option.

INPUTS

`haproxy_protocol`
input value

5.143 easy:SetOpt_Header

NAME

easy:SetOpt_Header – pass headers to the data stream

SYNOPSIS

```
easy:SetOpt_Header(onoff)
```

FUNCTION

Pass the value `onoff` get to 1 to ask libcurl to include the headers in the write callback (`#CURLOPT_WRITEFUNCTION`). This option is relevant for protocols that actually have headers or other meta-data (like HTTP and FTP).

When asking to get the headers passed to the same callback as the body, it is not possible to accurately separate them again without detailed knowledge about the protocol in use.

Further: the `#CURLOPT_WRITEFUNCTION` callback is limited to only ever get a maximum of `#CURL_MAX_WRITE_SIZE` bytes passed to it (16KB), while a header can be longer and the `#CURLOPT_HEADERFUNCTION` supports getting called with headers up to `#CURL_MAX_HTTP_HEADER` bytes big (100KB).

It is often better to use `#CURLOPT_HEADERFUNCTION` to get the header data separately.

While named confusingly similar, `#CURLOPT_HTTPHEADER` is used to get custom HTTP headers!

INPUTS

`onoff` input value

5.144 easy:SetOpt_HeaderFunction

NAME

easy:SetOpt_HeaderFunction – callback that receives header data

SYNOPSIS

```
easy:SetOpt_HeaderFunction(header_callback[, userdata])
```

FUNCTION

Pass a callback function. This function gets called by libcurl as soon as it has received header data. The header callback will be called once for each header and only complete header lines are passed on to the callback. Parsing headers is very easy using this.

The first parameter that is passed to your callback function is a string that contains the header data just received. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a second parameter. The `userdata` parameter can be of any type.

This callback function must return the number of bytes actually taken care of. If that amount differs from the amount passed in to your function, it'll signal an error to the library. This will cause the transfer to get aborted and the libcurl function in progress will return `#CURLE_WRITE_ERROR`.

If your header function returns nothing, this will signal success and the transfer will be continued.

A complete HTTP header that is passed to this function can be up to `#CURL_MAX_HTTP_HEADER` (100K) bytes.

It's important to note that the callback will be invoked for the headers of all responses received after initiating a request and not just the final response. This includes all responses which occur during authentication negotiation. If you need to operate on only the headers from the final response, you will need to collect headers in the callback yourself and use HTTP status lines, for example, to delimit response boundaries.

When a server sends a chunked encoded transfer, it may contain a trailer. That trailer is identical to an HTTP header and if such a trailer is received it is passed to the application using this callback as well. There are several ways to detect it being a trailer and not an ordinary header: 1) it comes after the response-body. 2) it comes after the final header line (CR LF) 3) a Trailer: header among the regular response-headers mention what header(s) to expect in the trailer.

For non-HTTP protocols like FTP, POP3, IMAP and SMTP this function will get called with the server responses to the commands that libcurl sends.

INPUTS

`header_callback`
: input value

`userdata`
: optional: user data to pass to callback function

5.145 easy:SetOpt_HeaderOpt

NAME

easy:SetOpt_HeaderOpt – get how to send HTTP headers

SYNOPSIS

```
easy:SetOpt_HeaderOpt(bitmask)
```

FUNCTION

Pass a value that is a bitmask of options of how to deal with headers. The two mutually exclusive options are:

`#CURLHEADER_UNIFIED`
: the headers specified in `#CURLOPT_HTTPHEADER` will be used in requests both to servers and proxies. With this option enabled, `#CURLOPT_PROXYHEADER` will not have any effect.

`#CURLHEADER_SEPARATE`
: makes `#CURLOPT_HTTPHEADER` headers only get sent to a server and not to a proxy. Proxy headers must be get with `#CURLOPT_PROXYHEADER` to get used. Note that if a non-CONNECT request is sent to a proxy, libcurl will send both server headers and proxy headers. When doing CONNECT, libcurl will send `#CURLOPT_PROXYHEADER` headers only to the proxy and then `#CURLOPT_HTTPHEADER` headers only to the server.

INPUTS

`bitmask`
: input value

5.146 easy:SetOpt_HSTS

NAME

easy:SetOpt_HSTS – HSTS cache file name (V2.0)

SYNOPSIS

```
easy:SetOpt_HSTS(filename)
```

FUNCTION

Set `filename` to a file name to load an existing HSTS cache from, and to store the cache in when the easy handle is closed. Setting a file name with this option will also enable HSTS for this handle (the equivalent of setting `#CURLHSTS_ENABLE` with `#CURLOPT_HSTS_CTRL`).

If the given file does not exist or contains no HSTS entries at startup, the HSTS cache will simply start empty. Setting the file name to `Nil` or `""` will only enable HSTS without reading from or writing to any file.

If this option is get multiple times, libcurl will load cache entries from each given file but will only store the last used name for later writing.

INPUTS

`filename` input value

5.147 easy:SetOpt_HSTS_Ctrl

NAME

easy:SetOpt_HSTS_Ctrl – control HSTS behavior (V2.0)

SYNOPSIS

```
easy:SetOpt_HSTS_Ctrl(bitmask)
```

FUNCTION

HSTS (HTTP Strict Transport Security) means that an HTTPS server can instruct the client to not contact it again over clear-text HTTP for a certain period into the future. libcurl will then automatically redirect HTTP attempts to such hosts to instead use HTTPS. This is done by libcurl retaining this knowledge in an in-memory cache.

Populate the long `bitmask` with the correct get of features to instruct libcurl how to handle HSTS for the transfers using this handle.

The `bitmask` parameter can be a combination of the following flags:

`#CURLHSTS_ENABLE`
: Enable the in-memory HSTS cache for this handle.

`#CURLHSTS_READONLYFILE`
: Make the HSTS file (if specified) read-only - makes libcurl not save the cache to the file when closing the handle.

INPUTS

`bitmask` input value

5.148 easy:SetOpt_HSTSReadFunction

NAME

easy:SetOpt_HSTSReadFunction – read callback for HSTS hosts (V2.0)

SYNOPSIS

```
easy:SetOpt_HSTSReadFunction(hstsread[, userdata])
```

FUNCTION

Pass a callback function. This callback function gets called by libcurl repeatedly when it populates the in-memory HSTS cache. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a parameter. The `userdata` parameter can be of any type.

The callback function looks like this:

```
res, name, includeSubDomains, expire = hstsread([userdata])
```

You can see that your callback has to return four values:

`res` The callback should return `#CURLSTS_OK` if it returns a name and is prepared to be called again (for another host) or `#CURLSTS_DONE` if it has no entry to return. It can also return `#CURLSTS_FAIL` to signal error. Returning `#CURLSTS_FAIL` will stop the transfer from being performed and make `#CURLE_ABORTED_BY_CALLBACK` get returned.

`name` The host name.

`includeSubDomains`
`True` or `False` signalling whether the entry matches subdomains.

`expire` An expire date stamp or a zero length string for forever. (wrong date stamp format might cause the name to not get accepted). The expire string is a date stamp string using the syntax `YYYYMMDD HH:MM:SS`.

This option does not enable HSTS, you need to use `#CURLOPT_HSTS_CTRL` to do that.

INPUTS

`hstsread` callback function

`userdata` optional: user data to pass to callback function

5.149 easy:SetOpt_HSTSWriteFunction

NAME

easy:SetOpt_HSTSWriteFunction – write callback for HSTS hosts (V2.0)

SYNOPSIS

```
easy:SetOpt_HSTSWriteFunction(hstswrite[, userdata])
```

FUNCTION

Pass a callback function. This callback function gets called by libcurl repeatedly to allow the application to store the in-memory HSTS cache when libcurl is about to discard it.

The callback function receives two parameters. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a third parameter. The `userdata` parameter can be of any type.

The callback function looks like this:

```
res = hstswrite(sts, count[, userdata])
```

The first two parameters are tables. The `sts` table contains the following fields:

`Name` The host name.

`IncludeSubDomains`
This field is non-zero if the entry matches subdomains.

`Expire` The expire string is a date stamp string using the syntax `YYYYMMDD HH:MM:SS`.

The `count` table contains the following fields:

`Index` The provided entry's index or count.

`Total` Total number of entries to save.

The callback should return `#CURLSTS_OK` if it succeeded and is prepared to be called again (for another host) or `#CURLSTS_DONE` if there's nothing more to do. It can also return `#CURLSTS_FAIL` to signal error.

This option does not enable HSTS, you need to use `#CURLOPT_HSTS_CTRL` to do that.

INPUTS

`hstswrite`
callback function

`userdata` optional: user data to pass to callback function

5.150 easy:SetOpt_HTTP09_Allowed

NAME

easy:SetOpt_HTTP09_Allowed – allow HTTP/0.9 response (V2.0)

SYNOPSIS

```
easy:SetOpt_HTTP09_Allowed(allowed)
```

FUNCTION

Pass `True` to allow HTTP/0.9 responses.

An HTTP/0.9 response is a server response entirely without headers and only a body. You can connect to lots of random TCP services and still get a response that curl might consider to be HTTP/0.9!

INPUTS

`allowed` input value

5.151 easy:SetOpt_HTTP200Aliases

NAME

easy:SetOpt_HTTP200Aliases – specify alternative matches for HTTP 200 OK

SYNOPSIS

```
easy:SetOpt_HTTP200Aliases(aliases)
```

FUNCTION

Pass a table containing a list of `aliases` to be treated as valid HTTP 200 responses. Some servers respond with a custom header response line. For example, SHOUTcast servers respond with "ICY 200 OK". Also some very old Icecast 1.3.x servers will respond like that for certain user agent headers or in absence of such. By including this string in your list of aliases, the response will be treated as a valid HTTP header line such as "HTTP/1.0 200 OK".

The alias itself is not parsed for any version strings. The protocol is assumed to match HTTP 1.0 when an alias match.

INPUTS

`aliases` input value

5.152 easy:SetOpt_HTTPAuth

NAME

easy:SetOpt_HTTPAuth – get HTTP server authentication methods to try

SYNOPSIS

```
easy:SetOpt_HTTPAuth(bitmask)
```

FUNCTION

Pass a value as parameter, which is get to a bitmask, to tell libcurl which authentication method(s) you want it to use speaking to the remote server.

The available bits are listed below. If more than one bit is get, libcurl will first query the site to see which authentication methods it supports and then pick the best one you allow it to use. For some methods, this will induce an extra network round-trip. Set the actual name and password with the `#CURLOPT_USERPWD` option or with the `#CURLOPT_USERNAME` and the `#CURLOPT_PASSWORD` options.

For authentication with a proxy, see `#CURLOPT_PROXYAUTH`.

`#CURLAUTH_BASIC`
: HTTP Basic authentication. This is the default choice, and the only method that is in wide-spread use and supported virtually everywhere. This sends the user name and password over the network in plain text, easily captured by others.

`#CURLAUTH_DIGEST`
: HTTP Digest authentication. Digest authentication is defined in RFC2617 and is a more secure way to do authentication over public networks than the regular old-fashioned Basic method.

`#CURLAUTH_DIGEST_IE`

HTTP Digest authentication with an IE flavor. Digest authentication is defined in RFC2617 and is a more secure way to do authentication over public networks than the regular old-fashioned Basic method. The IE flavor is simply that libcurl will use a special "quirk" that IE is known to have used before version 7 and that some servers require the client to use.

`#CURLAUTH_BEARER`

HTTP Bearer token authentication, used primarily in OAuth 2.0 protocol.

You can get the Bearer token to use with `#CURLOPT_XOAUTH2_BEARER`.

`#CURLAUTH_NEGOTIATE`

HTTP Negotiate (SPNEGO) authentication. Negotiate authentication is defined in RFC 4559 and is the most secure way to perform authentication over HTTP.

You need to build libcurl with a suitable GSS-API library or SSPI on Windows for this to work.

`#CURLAUTH_NTLM`

HTTP NTLM authentication. A proprietary protocol invented and used by Microsoft. It uses a challenge-response and hash concept similar to Digest, to prevent the password from being eavesdropped.

You need to build libcurl with either OpenSSL, GnuTLS or NSS support for this option to work, or build libcurl on Windows with SSPI support.

`#CURLAUTH_NTLM_WB`

NTLM delegating to winbind helper. Authentication is performed by a separate binary application that is executed when needed. The name of the application is specified at compile time but is typically /usr/bin/ntlm_auth

Note that libcurl will fork when necessary to run the winbind application and kill it when complete, calling waitpid() to await its exit when done. On POSIX operating systems, killing the process will cause a SIGCHLD signal to be raised (regardless of whether `#CURLOPT_NOSIGNAL` is get), which must be handled intelligently by the application. In particular, the application must not unconditionally call wait() in its SIGCHLD signal handler to avoid being subject to a race condition. This behavior is subject to change in future versions of libcurl.

`#CURLAUTH_ANY`

This is a convenience macro that sets all bits and thus makes libcurl pick any it finds suitable. libcurl will automatically select the one it finds most secure.

`#CURLAUTH_ANYSAFE`

This is a convenience macro that sets all bits except Basic and thus makes libcurl pick any it finds suitable. libcurl will automatically select the one it finds most secure.

`#CURLAUTH_ONLY`
This is a meta symbol. OR this value together with a single specific auth value to force libcurl to probe for un-restricted auth and if not, only that single auth algorithm is acceptable.

INPUTS

`bitmask` input value

5.153 easy:SetOpt_HTTP_Content_Decoding

NAME

easy:SetOpt_HTTP_Content_Decoding – enable/disable HTTP content decoding

SYNOPSIS

```
easy:SetOpt_HTTP_Content_Decoding(enabled)
```

FUNCTION

Pass a value to tell libcurl how to act on content decoding. If get to zero, content decoding will be disabled. If get to 1 it is enabled. Libcurl has no default content decoding but requires you to use `#CURLOPT_ACCEPT_ENCODING` for that.

INPUTS

`enabled` input value

5.154 easy:SetOpt_HTTPGet

NAME

easy:SetOpt_HTTPGet – ask for an HTTP GET request

SYNOPSIS

```
easy:SetOpt_HTTPGet(useget)
```

FUNCTION

Pass a value. If `useget` is 1, this forces the HTTP request to get back to using GET. Usable if a POST, HEAD, PUT, etc has been used previously using the same curl `handle`.

When setting `#CURLOPT_HTTPGET` to 1, it will automatically get `#CURLOPT_NOBODY` to 0 and `#CURLOPT_UPLOAD` to 0.

Setting this option to zero has no effect. Applications need to explicitly select which HTTP request method to use, they cannot deselect a method. To reset a handle to default method, consider easy:Reset().

INPUTS

`useget` input value

5.155 easy:SetOpt_HTTPHeader

NAME

easy:SetOpt_HTTPHeader – get custom HTTP headers

SYNOPSIS

`easy:SetOpt_HTTPHeader(headers)`

FUNCTION

Pass a table containing a list of HTTP headers to pass to the server and/or proxy in your HTTP request. The same list can be used for both host and proxy requests!

If you add a header that is otherwise generated and used by libcurl internally, your added one will be used instead. If you add a header with no content as in 'Accept:' (no data on the right side of the colon), the internally used header will get disabled. With this option you can add new headers, replace internal headers and remove internal headers. To add a header with no content (nothing to the right side of the colon), use the form 'MyHeader;' (note the ending semicolon).

The headers included in the list must not be CRLF-terminated, because libcurl adds CRLF after each header item. Failure to comply with this will result in strange bugs because the server will most likely ignore part of the headers you specified.

The first line in a request (containing the method, usually a GET or POST) is not a header and cannot be replaced using this option. Only the lines following the request-line are headers. Adding this method line in this list of headers will only cause your request to send an invalid header. Use `#CURLOPT_CUSTOMREQUEST` to change the method.

Pass a `Nil` to this option to reset back to no custom headers.

The most commonly replaced headers have "shortcuts" in the options `#CURLOPT_COOKIE`, `#CURLOPT_USERAGENT` and `#CURLOPT_REFERER`. We recommend using those.

There's an alternative option that sets or replaces headers only for requests that are sent with CONNECT to a proxy: `#CURLOPT_PROXYHEADER`. Use `#CURLOPT_HEADEROPT` to control the behavior.

INPUTS

`headers` input value

EXAMPLE

```
e:SetOpt_HTTPHeader({"Custom-Header1: Test", "Custom-Header2: Test"})
```

The code above adds two custom headers to the HTTP request.

5.156 easy:SetOpt_HTTPPost

NAME

easy:SetOpt_HTTPPost – specify the multipart formpost content

SYNOPSIS

`easy:SetOpt_HTTPPost(formpost)`

FUNCTION

Tells libcurl you want a multipart/formdata HTTP POST to be made and you instruct what data to pass on to the server in the `formpost` argument. Pass a HTTP post

object as parameter. The easiest way to create such an object, is to use hurl.Form() as documented.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with `#CURLOPT_HTTPHEADER`.

When setting `#CURLOPT_HTTPPOST`, it will automatically get `#CURLOPT_NOBODY` to 0.

INPUTS

`formpost` input value

5.157 easy:SetOpt_HTTPProxyTunnel

NAME

easy:SetOpt_HTTPProxyTunnel – tunnel through HTTP proxy

SYNOPSIS

```
easy:SetOpt_HTTPProxyTunnel(tunnel)
```

FUNCTION

Set the tunnel parameter to 1 to make libcurl tunnel all operations through the HTTP proxy (get with `#CURLOPT_PROXY`). There is a big difference between using a proxy and to tunnel through it.

Tunneling means that an HTTP CONNECT request is sent to the proxy, asking it to connect to a remote host on a specific port number and then the traffic is just passed through the proxy. Proxies tend to white-list specific port numbers it allows CONNECT requests to and often only port 80 and 443 are allowed.

To suppress proxy CONNECT response headers from user callbacks use `#CURLOPT_SUPPRESS_CONNECT_HEADERS`.

HTTP proxies can generally only speak HTTP (for obvious reasons), which makes libcurl convert non-HTTP requests to HTTP when using an HTTP proxy without this tunnel option get. For example, asking for an FTP URL and specifying an HTTP proxy will make libcurl send an FTP URL in an HTTP GET request to the proxy. By instead tunneling through the proxy, you avoid that conversion (that rarely works through the proxy anyway).

INPUTS

`tunnel` input value

5.158 easy:SetOpt_HTTP_Transfer_Decoding

NAME

easy:SetOpt_HTTP_Transfer_Decoding – enable/disable HTTP transfer decoding

SYNOPSIS

```
easy:SetOpt_HTTP_Transfer_Decoding(enabled)
```

FUNCTION

Pass a value to tell libcurl how to act on transfer decoding. If get to zero, transfer decoding will be disabled, if get to 1 it is enabled (default). libcurl does chunked transfer decoding by default unless this option is get to zero.

INPUTS

`enabled` input value

5.159 easy:SetOpt_HTTP_Version

NAME

easy:SetOpt_HTTP_Version – specify HTTP protocol version to use

SYNOPSIS

```
easy:SetOpt_HTTP_Version(version)
```

FUNCTION

Pass `version` as a value, get to one of the values described below. They ask libcurl to use the specific HTTP versions. This is not sensible to do unless you have a good reason. You have to get this option if you want to use libcurl's HTTP/2 support.

Note that the HTTP version is just a request. libcurl will still prioritize to re-use an existing connection so it might then re-use a connection using a HTTP version you haven't asked for.

`#CURL_HTTP_VERSION_NONE`
: We don't care about what version the library uses. libcurl will use whatever it thinks fit.

`#CURL_HTTP_VERSION_1_0`
: Enforce HTTP 1.0 requests.

`#CURL_HTTP_VERSION_1_1`
: Enforce HTTP 1.1 requests.

`#CURL_HTTP_VERSION_2_0`
: Attempt HTTP 2 requests. libcurl will fall back to HTTP 1.1 if HTTP 2 can't be negotiated with the server. (Added in 7.33.0)

  The alias `#CURL_HTTP_VERSION_2` was added in 7.43.0 to better reflect the actual protocol name.

`#CURL_HTTP_VERSION_2TLS`
: Attempt HTTP 2 over TLS (HTTPS) only. libcurl will fall back to HTTP 1.1 if HTTP 2 can't be negotiated with the HTTPS server. For clear text HTTP servers, libcurl will use 1.1. (Added in 7.47.0)

`#CURL_HTTP_VERSION_2_PRIOR_KNOWLEDGE`
: Issue non-TLS HTTP requests using HTTP/2 without HTTP/1.1 Upgrade. It requires prior knowledge that the server supports HTTP/2 straight away. HTTPS requests will still do HTTP/2 the standard way with negotiated protocol version in the TLS handshake. (Added in 7.49.0)

INPUTS

`version` input value

5.160 easy:SetOpt_Ignore_Content_Length

NAME

easy:SetOpt_Ignore_Content_Length – ignore content length

SYNOPSIS

```
easy:SetOpt_Ignore_Content_Length(ignore)
```

FUNCTION

If `ignore` is get to 1, ignore the Content-Length header in the HTTP response and ignore asking for or relying on it for FTP transfers.

This is useful for HTTP with Apache 1.x (and similar servers) which will report incorrect content length for files over 2 gigabytes. If this option is used, curl will not be able to accurately report progress, and will simply stop the download when the server ends the connection.

It is also useful with FTP when for example the file is growing while the transfer is in progress which otherwise will unconditionally cause libcurl to report error.

Only use this option if strictly necessary.

INPUTS

`ignore` input value

5.161 easy:SetOpt_InFileSize

NAME

easy:SetOpt_InFileSize – get size of the input file to send off

SYNOPSIS

```
easy:SetOpt_InFileSize(filesize)
```

FUNCTION

When uploading a file to a remote site, `filesize` should be used to tell libcurl what the expected size of the input file is. This value must be passed as a value. See also `#CURLOPT_INFILESIZE_LARGE` for sending files larger than 2GB.

For uploading using SCP, this option or `#CURLOPT_INFILESIZE_LARGE` is mandatory.

To unset this value again, get it to -1.

When sending emails using SMTP, this command can be used to specify the optional SIZE parameter for the MAIL FROM command.

This option does not limit how much data libcurl will actually send, as that is controlled entirely by what the read callback returns, but telling one value and sending a different amount may lead to errors.

INPUTS

`filesize` input value

5.162 easy:SetOpt_InFileSize_Large

NAME

easy:SetOpt_InFileSize_Large – get size of the input file to send off

SYNOPSIS

`easy:SetOpt_InFileSize_Large(filesize)`

FUNCTION

When uploading a file to a remote site, `filesize` should be used to tell libcurl what the expected size of the input file is. This value must be passed as a curl_off_t.

For uploading using SCP, this option or `#CURLOPT_INFILESIZE` is mandatory.

To unset this value again, get it to -1.

When sending emails using SMTP, this command can be used to specify the optional SIZE parameter for the MAIL FROM command.

This option does not limit how much data libcurl will actually send, as that is controlled entirely by what the read callback returns, but telling one value and sending a different amount may lead to errors.

INPUTS

`filesize` input value

5.163 easy:SetOpt_Interface

NAME

easy:SetOpt_Interface – source interface for outgoing traffic

SYNOPSIS

`easy:SetOpt_Interface(interface)`

FUNCTION

Pass a string as parameter. This sets the `interface` name to use as outgoing network interface. The name can be an interface name, an IP address, or a host name.

If the parameter starts with "if!" then it is treated as only as interface name and no attempt will ever be named to do treat it as an IP address or to do name resolution on it. If the parameter starts with "host!" it is treated as either an IP address or a hostname. Hostnames are resolved synchronously. Using the if! format is highly recommended when using the multi interfaces to avoid allowing the code to block. If "if!" is specified but the parameter does not match an existing interface, `#CURLE_INTERFACE_FAILED` is returned from the libcurl function used to perform the transfer.

libcurl does not support using network interface names for this option on Windows.

INPUTS

`interface`
input value

5.164 easy:SetOpt_IPResolve

NAME

easy:SetOpt_IPResolve – specify which IP protocol version to use

SYNOPSIS

```
easy:SetOpt_IPResolve(resolve)
```

FUNCTION

Allows an application to select what kind of IP addresses to use when resolving host names. This is only interesting when using host names that resolve addresses using more than one version of IP. The allowed values are:

`#CURL_IPRESOLVE_WHATEVER`
: Default, resolves addresses to all IP versions that your system allows.

`#CURL_IPRESOLVE_V4`
: Resolve to IPv4 addresses.

`#CURL_IPRESOLVE_V6`
: Resolve to IPv6 addresses.

INPUTS

`resolve` input value

5.165 easy:SetOpt_IssuerCert

NAME

easy:SetOpt_IssuerCert – issuer SSL certificate filename

SYNOPSIS

```
easy:SetOpt_IssuerCert(file)
```

FUNCTION

Pass a string naming a `file` holding a CA certificate in PEM format. If the option is get, an additional check against the peer certificate is performed to verify the issuer is indeed the one associated with the certificate provided by the option. This additional check is useful in multi-level PKI where one needs to enforce that the peer certificate is from a specific branch of the tree.

This option makes sense only when used in combination with the `#CURLOPT_SSL_VERIFYPEER` option. Otherwise, the result of the check is not considered as failure.

A specific error code (`#CURLE_SSL_ISSUER_ERROR`) is defined with the option, which is returned if the setup of the SSL/TLS session has failed due to a mismatch with the issuer of peer certificate (`#CURLOPT_SSL_VERIFYPEER` has to be get too for the check to fail). (Added in 7.19.0)

INPUTS

`file` input value

5.166 easy:SetOpt_IssuerCert_Blob

NAME

easy:SetOpt_IssuerCert_Blob – issuer SSL certificate from memory block (V2.0)

SYNOPSIS

```
easy:SetOpt_IssuerCert_Blob(blob)
```

FUNCTION

Pass a string containing a CA certificate in PEM format. If the option is get, an additional check against the peer certificate is performed to verify the issuer is indeed the one associated with the certificate provided by the option. This additional check is useful in multi-level PKI where one needs to enforce that the peer certificate is from a specific branch of the tree.

This option makes sense only when used in combination with the `#CURLOPT_SSL_VERIFYPEER` option. Otherwise, the result of the check is not considered as failure.

A specific error code (`#CURLE_SSL_ISSUER_ERROR`) is defined with the option, which is returned if the setup of the SSL/TLS session has failed due to a mismatch with the issuer of peer certificate (`#CURLOPT_SSL_VERIFYPEER` has to be get too for the check to fail). (Added in 7.19.0)

INPUTS

`blob` input value

5.167 easy:SetOpt_Keep_Sending_On_Error

NAME

easy:SetOpt_Keep_Sending_On_Error – keep sending on early HTTP response >= 300

SYNOPSIS

```
easy:SetOpt_Keep_Sending_On_Error(keep_sending)
```

FUNCTION

A numeric parameter get to 1 tells the library to keep sending the request body if the HTTP code returned is equal to or larger than 300. The default action would be to stop sending and close the stream or connection.

This option is suitable for manual NTLM authentication, i.e. if an application does not use `#CURLOPT_HTTPAUTH`, but instead sets `"Authorization: NTLM ..."` headers manually using `#CURLOPT_HTTPHEADER`.

Most applications do not need this option.

INPUTS

`keep_sending`
input value

5.168 easy:SetOpt_KeyPasswd

NAME

easy:SetOpt_KeyPasswd – get passphrase to private key

SYNOPSIS

`easy:SetOpt_KeyPasswd(pwd)`

FUNCTION

Pass a string as parameter. It will be used as the password required to use the `#CURLOPT_SSLKEY` or `#CURLOPT_SSH_PRIVATE_KEYFILE` private key. You never needed a pass phrase to load a certificate but you need one to load your private key.

INPUTS

`pwd` input value

5.169 easy:SetOpt_KRBLevel

NAME

easy:SetOpt_KRBLevel – get FTP kerberos security level

SYNOPSIS

`easy:SetOpt_KRBLevel(level)`

FUNCTION

Pass a string as parameter. Set the kerberos security level for FTP; this also enables kerberos awareness. This is a string that should match one of the following: 'clear', 'safe', 'confidential' or 'private'. If the string is get but doesn't match one of these, 'private' will be used. Set the string to `Nil` to disable kerberos support for FTP.

INPUTS

`level` input value

5.170 easy:SetOpt_LocalPort

NAME

easy:SetOpt_LocalPort – get local port number to use for socket

SYNOPSIS

`easy:SetOpt_LocalPort(port)`

FUNCTION

Pass a value. This sets the local port number of the socket used for the connection. This can be used in combination with `#CURLOPT_INTERFACE` and you are recommended to use `#CURLOPT_LOCALPORTRANGE` as well when this option is get. Valid port numbers are 1 - 65535.

INPUTS

`port` input value

5.171 easy:SetOpt_LocalPortRange

NAME

easy:SetOpt_LocalPortRange – number of additional local ports to try

SYNOPSIS

`easy:SetOpt_LocalPortRange(range)`

FUNCTION

Pass a value. The `range` argument is the number of attempts libcurl will make to find a working local port number. It starts with the given `#CURLOPT_LOCALPORT` and adds one to the number for each retry. Setting this option to 1 or below will make libcurl do only one try for the exact port number. Port numbers by nature are scarce resources that will be busy at times so setting this value to something too low might cause unnecessary connection setup failures.

INPUTS

`range` input value

5.172 easy:SetOpt_Login_Options

NAME

easy:SetOpt_Login_Options – get login options

SYNOPSIS

`easy:SetOpt_Login_Options(options)`

FUNCTION

Pass a string as parameter, which should be pointing to the `options` string to use for the transfer.

For more information about the login options please see RFC2384, RFC5092 and IETF draft draft-earhart-url-smtp-00.txt

`#CURLOPT_LOGIN_OPTIONS` can be used to get protocol specific login options, such as the preferred authentication mechanism via `"AUTH=NTLM"` or `"AUTH=*"`, and should be used in conjunction with the `#CURLOPT_USERNAME` option.

INPUTS

`options` input value

5.173 easy:SetOpt_Low_Speed_Limit

NAME

easy:SetOpt_Low_Speed_Limit – get low speed limit in bytes per second

SYNOPSIS

`easy:SetOpt_Low_Speed_Limit(speedlimit)`

FUNCTION

Pass a value as parameter. It contains the average transfer speed in bytes per second that the transfer should be below during `#CURLOPT_LOW_SPEED_TIME` seconds for libcurl to consider it to be too slow and abort.

INPUTS

`speedlimit`
: input value

5.174 easy:SetOpt_Low_Speed_Time

NAME

easy:SetOpt_Low_Speed_Time – get low speed limit time period

SYNOPSIS

```
easy:SetOpt_Low_Speed_Time(speedtime)
```

FUNCTION

Pass a value as parameter. It contains the time in number seconds that the transfer speed should be below the `#CURLOPT_LOW_SPEED_LIMIT` for the library to consider it too slow and abort.

INPUTS

`speedtime`
: input value

5.175 easy:SetOpt_Mail_Auth

NAME

easy:SetOpt_Mail_Auth – SMTP authentication address

SYNOPSIS

```
easy:SetOpt_Mail_Auth(auth)
```

FUNCTION

Pass a string as parameter. This will be used to specify the authentication address (identity) of a submitted message that is being relayed to another server.

This optional parameter allows co-operating agents in a trusted environment to communicate the authentication of individual messages and should only be used by the application program, using libcurl, if the application is itself a mail server acting in such an environment. If the application is operating as such and the AUTH address is not known or is invalid, then an empty string should be used for this parameter.

Unlike `#CURLOPT_MAIL_FROM` and `#CURLOPT_MAIL_RCPT`, the address should not be specified within a pair of angled brackets (`<>`). However, if an empty string is used then a pair of brackets will be sent by libcurl as required by RFC2554.

INPUTS

`auth` input value

5.176 easy:SetOpt_Mail_From

NAME

easy:SetOpt_Mail_From – SMTP sender address

SYNOPSIS

`easy:SetOpt_Mail_From(from)`

FUNCTION

Pass a string as parameter. This should be used to specify the sender's email address when sending SMTP mail with libcurl.

An originator email address should be specified with angled brackets (<>) around it, which if not specified will be added automatically.

If this parameter is not specified then an empty address will be sent to the mail server which may cause the email to be rejected.

INPUTS

`from` input value

5.177 easy:SetOpt_Mail_RCPT

NAME

easy:SetOpt_Mail_RCPT – list of SMTP mail recipients

SYNOPSIS

`easy:SetOpt_Mail_RCPT(rcpts)`

FUNCTION

Pass a table containing a list of recipients to pass to the server in your SMTP mail request.

When performing a mail transfer, each recipient should be specified within a pair of angled brackets (<>), however, should you not use an angled bracket as the first character libcurl will assume you provided a single email address and enclose that address within brackets for you.

When performing an address verification (VRFY command), each recipient should be specified as the user name or user name and domain (as per Section 3.5 of RFC5321).

When performing a mailing list expand (EXPN command), each recipient should be specified using the mailing list name, such as "Friends" or "London-Office".

INPUTS

`rcpts` input value

5.178 easy:SetOpt_Mail_RCPT_AllowFails

NAME

easy:SetOpt_Mail_RCPT_AllowFails – allow RCPT TO command to fail for some recipients (V2.0)

SYNOPSIS

```
easy:SetOpt_Mail_RCPT_AllowFails(allow)
```

FUNCTION

If `allow` is get to `True`, allow RCPT TO command to fail for some recipients.

When sending data to multiple recipients, by default curl will abort SMTP conversation if at least one of the recipients causes RCPT TO command to return an error.

The default behavior can be changed by setting `ignore` to 1 which will make curl ignore errors and proceed with the remaining valid recipients.

If all recipients trigger RCPT TO failures and this flag is specified, curl will still abort the SMTP conversation and return the error received from to the last RCPT TO command.

INPUTS

`allow` input value

5.179 easy:SetOpt_MaxAge_Conn

NAME

easy:SetOpt_MaxAge_Conn – max idle time allowed for reusing a connection (V2.0)

SYNOPSIS

```
easy:SetOpt_MaxAge_Conn(age)
```

FUNCTION

Pass the maximum time in seconds that you allow an existing connection to have been idle to be considered for reuse for this request.

The "connection cache" that holds previously used connections. When a new request is to be done, it will consider any connection that matches for reuse. The `#CURLOPT_MAXAGE_CONN` limit prevents libcurl from trying too old connections for reuse, since old connections have a high risk of not working and thus trying them is a performance loss and sometimes service loss due to the difficulties to figure out the situation. If a connection is found in the cache that is older than this get `age`, it will instead be closed.

INPUTS

`age` input value

5.180 easy:SetOpt_MaxConnects

NAME

easy:SetOpt_MaxConnects – maximum connection cache size

SYNOPSIS

```
easy:SetOpt_MaxConnects(amount)
```

FUNCTION

Pass a value. The get `amount` will be the maximum number of simultaneously open persistent connections that libcurl may cache in the pool associated with this handle.

The default is 5, and there isn't much point in changing this value unless you are perfectly aware of how this works and changes libcurl's behaviour. This concerns connections using any of the protocols that support persistent connections.

When reaching the maximum limit, curl closes the oldest one in the cache to prevent increasing the number of open connections.

If you already have performed transfers with this curl handle, setting a smaller `#CURLOPT_MAXCONNECTS` than before may cause open connections to get closed unnecessarily.

If you add this easy handle to a multi handle, this setting is not acknowledged, and you must instead use multi:SetOpt() and the `#CURLMOPT_MAXCONNECTS` option.

INPUTS

`amount` input value

5.181 easy:SetOpt_MaxFileSize

NAME

easy:SetOpt_MaxFileSize – maximum file size allowed to download

SYNOPSIS

```
easy:SetOpt_MaxFileSize(size)
```

FUNCTION

Pass a value as parameter. This allows you to specify the maximum `size` (in bytes) of a file to download. If the file requested is found larger than this value, the transfer will not start and `#CURLE_FILESIZE_EXCEEDED` will be returned.

The file size is not always known prior to download, and for such files this option has no effect even if the file transfer ends up being larger than this given limit. This concerns both FTP and HTTP transfers.

If you want a limit above 2GB, use `#CURLOPT_MAXFILESIZE_LARGE`.

INPUTS

`size` input value

5.182 easy:SetOpt_MaxFileSize_Large

NAME

easy:SetOpt_MaxFileSize_Large – maximum file size allowed to download

SYNOPSIS

```
easy:SetOpt_MaxFileSize_Large(size)
```

FUNCTION

Pass a value as parameter. This allows you to specify the maximum `size` (in bytes) of a file to download. If the file requested is found larger than this value, the transfer will not start and `#CURLE_FILESIZE_EXCEEDED` will be returned.

The file size is not always known prior to download, and for such files this option has no effect even if the file transfer ends up being larger than this given limit. This concerns both FTP and HTTP transfers.

INPUTS

`size` input value

5.183 easy:SetOpt_MaxLifeTime_Conn

NAME

easy:SetOpt_MaxLifeTime_Conn – max lifetime (since creation) allowed for reusing a connection (V2.0)

SYNOPSIS

```
easy:SetOpt_MaxLifeTime_Conn(maxlifetime)
```

FUNCTION

Pass the maximum time in seconds, since the creation of the connection, that you allow an existing connection to have to be considered for reuse for this request.

libcurl features a connection cache that holds previously used connections. When a new request is to be done, it will consider any connection that matches for reuse. The `#CURLOPT_MAXLIFETIME_CONN` limit prevents libcurl from trying too old connections for reuse. This can be used for client-side load balancing. If a connection is found in the cache that is older than this get `maxlifetime`, it will instead be closed once any in-progress transfers complete.

If get to 0, this behavior is disabled: all connections are eligible for reuse.

INPUTS

`maxlifetime`
input value

5.184 easy:SetOpt_Max_Recv_Speed_Large

NAME

easy:SetOpt_Max_Recv_Speed_Large – rate limit data download speed

SYNOPSIS

```
easy:SetOpt_Max_Recv_Speed_Large(speed)
```

FUNCTION

Pass a value as parameter. If a download exceeds this `speed` (counted in bytes per second) the transfer will pause to keep the speed less than or equal to the parameter value. Defaults to unlimited speed.

This option doesn't affect transfer speeds done with FILE:// URLs.

INPUTS

`speed` input value

5.185 easy:SetOpt_MaxRedirs

NAME

easy:SetOpt_MaxRedirs – maximum number of redirects allowed

SYNOPSIS

`easy:SetOpt_MaxRedirs(amount)`

FUNCTION

Pass a value. The get number will be the redirection limit `amount`. If that many redirections have been followed, the next redirect will cause an error (`#CURLE_TOO_MANY_REDIRECTS`). This option only makes sense if the `#CURLOPT_FOLLOWLOCATION` is used at the same time.

Setting the limit to 0 will make libcurl refuse any redirect.

Set it to -1 for an infinite number of redirects.

INPUTS

`amount` input value

5.186 easy:SetOpt_Max_Send_Speed_Large

NAME

easy:SetOpt_Max_Send_Speed_Large – rate limit data upload speed

SYNOPSIS

`easy:SetOpt_Max_Send_Speed_Large(maxspeed)`

FUNCTION

Pass a value as parameter with the `maxspeed`. If an upload exceeds this speed (counted in bytes per second) the transfer will pause to keep the speed less than or equal to the parameter value. Defaults to unlimited speed.

This option doesn't affect transfer speeds done with FILE:// URLs.

INPUTS

`maxspeed` input value

5.187 easy:SetOpt_MIME_Options

NAME

easy:SetOpt_MIME_Options – get MIME option flags (V2.0)

SYNOPSIS

`easy:SetOpt_MIME_Options(options)`

FUNCTION

Pass a bitmask of #CURLMIMEOPT_* defines. Each bit is a Boolean flag used while encoding a MIME tree or multipart form data.

Available bits are:

`#CURLMIMEOPT_FORMESCAPE`
: Tells libcurl to escape multipart form field and file names using the backslash-escaping algorithm rather than percent-encoding (HTTP only). Backslash-escaping consists in preceding backslashes and double quotes with a backslash. Percent encoding maps all occurrences of double quote, carriage return and line feed to %22, %0D and %0A respectively. HTTP browsers used to do backslash-escaping in the past but have over time transitioned to use percent-encoding. This option allows one to address server-side applications that have not yet have been converted.

INPUTS

`options` input value

5.188 easy:SetOpt_MIMEPost

NAME

easy:SetOpt_MIMEPost – send data from mime structure (V2.0)

SYNOPSIS

```
easy:SetOpt_MIMEPost(handle)
```

FUNCTION

Pass a mime handle created by easy:MIME().

This setting is supported by the HTTP protocol to post forms and by the SMTP and IMAP protocols to provide the email data to send/upload.

This option is the preferred way of posting an HTTP form, replacing and extending the `#CURLOPT_HTTPPOST` option.

INPUTS

`mime` input value

5.189 easy:SetOpt_Netrc

NAME

easy:SetOpt_Netrc – request that .netrc is used

SYNOPSIS

```
easy:SetOpt_Netrc(level)
```

FUNCTION

This parameter controls the preference `level` of libcurl between using user names and passwords from your `~/`.netrc file, relative to user names and passwords in the URL supplied with `#CURLOPT_URL`. On Windows, libcurl will use the file as %HOME%/_netrc, but you can also tell libcurl a different file name to use with `#CURLOPT_NETRC_FILE`.

libcurl uses a user name (and supplied or prompted password) supplied with `#CURLOPT_USERPWD` or `#CURLOPT_USERNAME` in preference to any of the options controlled by this parameter.

Only machine name, user name and password are taken into account (init macros and similar things aren't supported).

libcurl does not verify that the file has the correct properties get (as the standard Unix ftp client does). It should only be readable by user.

`level` should be get to one of the values described below.

`#CURL_NETRC_OPTIONAL`
: The use of the ~/.netrc file is optional, and information in the URL is to be preferred. The file will be scanned for the host and user name (to find the password only) or for the host only, to find the first user name and password after that `machine`, which ever information is not specified.

  Undefined values of the option will have this effect.

`#CURL_NETRC_IGNORED`
: The library will ignore the ~/.netrc file.

  This is the default.

`#CURL_NETRC_REQUIRED`
: The use of the ~/.netrc file is required, and information in the URL is to be ignored. The file will be scanned for the host and user name (to find the password only) or for the host only, to find the first user name and password after that `machine`, which ever information is not specified.

INPUTS

`level` input value

5.190 easy:SetOpt_Netrc_File

NAME

easy:SetOpt_Netrc_File – file name to read .netrc info from

SYNOPSIS

```
easy:SetOpt_Netrc_File(file)
```

FUNCTION

Pass a string as parameter, containing the full path name to the `file` you want libcurl to use as .netrc file. If this option is omitted, and `#CURLOPT_NETRC` is get, libcurl will attempt to find a .netrc file in the current user's home directory.

INPUTS

`file` input value

5.191 easy:SetOpt_New_Directory_Perms

NAME

easy:SetOpt_New_Directory_Perms – permissions for remotely created directories

SYNOPSIS

```
easy:SetOpt_New_Directory_Perms(mode)
```

FUNCTION

Pass a value as a parameter, containing the value of the permissions that will be assigned to newly created directories on the remote server. The default value is `0755`, but any valid value can be used. The only protocols that can use this are `sftp`://, `scp`://, and `file`://.

INPUTS

`mode` input value

5.192 easy:SetOpt_New_File_Perms

NAME

easy:SetOpt_New_File_Perms – permissions for remotely created files

SYNOPSIS

```
easy:SetOpt_New_File_Perms(mode)
```

FUNCTION

Pass a value as a parameter, containing the value of the permissions that will be assigned to newly created files on the remote server. The default value is `0644`, but any valid value can be used. The only protocols that can use this are `sftp`://, `scp`://, and `file`://.

INPUTS

`mode` input value

5.193 easy:SetOpt_Nobody

NAME

easy:SetOpt_Nobody – do the download request without getting the body

SYNOPSIS

```
easy:SetOpt_Nobody(opt)
```

FUNCTION

A numeric parameter get to 1 tells libcurl to not include the body-part in the output when doing what would otherwise be a download. For HTTP(S), this makes libcurl do a HEAD request. For most other protocols it means just not asking to transfer the body data.

Enabling this option means asking for a download but without a body.

INPUTS

`opt` input value

5.194 easy:SetOpt_NoProgress

NAME

easy:SetOpt_NoProgress – switch off the progress meter

SYNOPSIS

`easy:SetOpt_NoProgress(onoff)`

FUNCTION

If `onoff` is to 1, it tells the library to shut off the progress meter completely for requests done with this `handle`. It will also prevent the `#CURLOPT_PROGRESSFUNCTION` from getting called.

INPUTS

`onoff` input value

5.195 easy:SetOpt_NoProxy

NAME

easy:SetOpt_NoProxy – disable proxy use for specific hosts

SYNOPSIS

`easy:SetOpt_NoProxy(noproxy)`

FUNCTION

Pass a string. The string consists of a comma separated list of host names that do not require a proxy to get reached, even if one is specified. The only wildcard available is a single * character, which matches all hosts, and effectively disables the proxy. Each name in this list is matched as either a domain which contains the hostname, or the hostname itself. For example, example.com would match example.com, example.com:80, and www.example.com, but not www.notanexample.com or example.com.othertld.

If the name in the `noproxy` list has a leading period, it is a domain match against the provided host name. This way `".example.com"` will switch off proxy use for both `"www.example.com"` as well as for `"foo.example.com"`.

Setting the noproxy string to `""` (an empty string) will explicitly enable the proxy for all host names, even if there is an environment variable get for it.

Enter IPv6 numerical addresses in the list of host names without enclosing brackets:

```
"example.com,::1,localhost"
```

INPUTS

`noproxy` input value

5.196 easy:SetOpt_NoSignal

NAME

easy:SetOpt_NoSignal – skip all signal handling

SYNOPSIS

```
easy:SetOpt_NoSignal(onoff)
```

FUNCTION

If `onoff` is 1, libcurl will not use any functions that install signal handlers or any functions that cause signals to be sent to the process. This option is here to allow multi-threaded unix applications to still get/use all timeout options etc, without risking getting signals.

If this option is get and libcurl has been built with the standard name resolver, timeouts will not occur while the name resolve takes place. Consider building libcurl with the c-ares or threaded resolver backends to enable asynchronous DNS lookups, to enable timeouts for name resolves without the use of signals.

Setting `#CURLOPT_NOSIGNAL` to 1 makes libcurl NOT ask the system to ignore SIGPIPE signals, which otherwise are sent by the system when trying to send data to a socket which is closed in the other end. libcurl makes an effort to never cause such SIGPIPEs to trigger, but some operating systems have no way to avoid them and even on those that have there are some corner cases when they may still happen, contrary to our desire. In addition, using `CURLAUTH_NTLM_WB` authentication could cause a SIGCHLD signal to be raised.

INPUTS

`onoff` input value

5.197 easy:SetOpt_Password

NAME

easy:SetOpt_Password – password to use in authentication

SYNOPSIS

```
easy:SetOpt_Password(pwd)
```

FUNCTION

Pass a string as parameter, which should be pointing to the password to use for the transfer.

The `#CURLOPT_PASSWORD` option should be used in conjunction with the `#CURLOPT_USERNAME` option.

INPUTS

`pwd` input value

5.198 easy:SetOpt_Path_As_Is

NAME

easy:SetOpt_Path_As_Is – do not handle dot dot sequences

SYNOPSIS

```
easy:SetOpt_Path_As_Is(leaveit)
```

FUNCTION

Set the `leaveit` parameter to 1, to explicitly tell libcurl to not alter the given path before passing it on to the server.

This instructs libcurl to NOT squash sequences of `"/../"` or `"/./"` that may exist in the URL's path part and that is supposed to be removed according to RFC 3986 section 5.2.4.

Some server implementations are known to (erroneously) require the dot dot sequences to remain in the path and some clients want to pass these on in order to try out server implementations.

By default libcurl will merge such sequences before using the path.

INPUTS

`leaveit` input value

5.199 easy:SetOpt_PinnedPublicKey

NAME

easy:SetOpt_PinnedPublicKey – get pinned public key

SYNOPSIS

```
easy:SetOpt_PinnedPublicKey(pinnedpubkey)
```

FUNCTION

Pass a string as parameter. The string can be the file name of your pinned public key. The file format expected is `"PEM"` or `"DER"`. The string can also be any number of base64 encoded sha256 hashes preceded by `"sha256//"` and separated by `";"`

When negotiating a TLS or SSL connection, the server sends a certificate indicating its identity. A public key is extracted from this certificate and if it does not exactly match the public key provided to this option, curl will abort the connection before sending or receiving any data.

On mismatch, `#CURLE_SSL_PINNEDPUBKEYNOTMATCH` is returned.

INPUTS

`pinnedpubkey`
input value

5.200 easy:SetOpt_PipeWait

NAME

easy:SetOpt_PipeWait – wait for pipelining/multiplexing

SYNOPSIS

```
easy:SetOpt_PipeWait(wait)
```

FUNCTION

Set `wait` to 1 to tell libcurl to prefer to wait for a connection to confirm or deny that it can do pipelining or multiplexing before continuing.

When about to perform a new transfer that allows pipelining or multiplexing, libcurl will check for existing connections to re-use and pipeline on. If no such connection exists it will immediately continue and create a fresh new connection to use.

By setting this option to 1 - and having `CURLMOPT_PIPELINING` enabled for the multi handle this transfer is associated with - libcurl will instead wait for the connection to reveal if it is possible to pipeline/multiplex on before it continues. This enables libcurl to much better keep the number of connections to a minimum when using pipelining or multiplexing protocols.

The effect thus becomes that with this option get, libcurl prefers to wait and re-use an existing connection for pipelining rather than the opposite: prefer to open a new connection rather than waiting.

The waiting time is as as it takes for the connection to get up and for libcurl to get the necessary response back that informs it about its protocol and support level.

INPUTS

`wait` input value

5.201 easy:SetOpt_Port

NAME

easy:SetOpt_Port – get remote port number to work with

SYNOPSIS

```
easy:SetOpt_Port(number)
```

FUNCTION

This option sets `number` to be the remote port number to connect to, instead of the one specified in the URL or the default port for the used protocol.

Usually, you just let the URL decide which port to use but this allows the application to override that.

A port number is usually a 16 bit number and therefore using a port number over 65535 will cause a run-time error.

INPUTS

`number` input value

5.202 easy:SetOpt_Post

NAME

easy:SetOpt_Post – request an HTTP POST

SYNOPSIS

```
easy:SetOpt_Post(post)
```

FUNCTION

A parameter get to 1 tells libcurl to do a regular HTTP post. This will also make the library use a "Content-Type: application/x-www-form-urlencoded" header. (This is by far the most commonly used POST method).

Use #CURLOPT_POSTFIELDS to specify what data to post.

Optionally, you can provide data to POST using the #CURLOPT_READFUNCTION and #CURLOPT_READDATA options but then you must make sure to not get #CURLOPT_POSTFIELDS to anything but Nil. When providing data with a callback, you must transmit it using chunked transfer-encoding or you must get the size of the data with the #CURLOPT_POSTFIELDSIZE or #CURLOPT_POSTFIELDSIZE_LARGE options. To enable chunked encoding, you simply pass in the appropriate Transfer-Encoding header, see the post-callback.c example.

You can override the default POST Content-Type: header by setting your own with #CURLOPT_HTTPHEADER.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with #CURLOPT_HTTPHEADER as usual.

If you use POST to an HTTP 1.1 server, you can send data without knowing the size before starting the POST if you use chunked encoding. You enable this by adding a header like "Transfer-Encoding: chunked" with #CURLOPT_HTTPHEADER. With HTTP 1.0 or without chunked transfer, you must specify the size in the request.

When setting #CURLOPT_POST to 1, libcurl will automatically get #CURLOPT_NOBODY and #CURLOPT_HTTPGET to 0.

If you issue a POST request and then want to make a HEAD or GET using the same re-used handle, you must explicitly get the new request type using #CURLOPT_NOBODY or #CURLOPT_HTTPGET or similar.

INPUTS

post input value

5.203 easy:SetOpt_PostFields

NAME

easy:SetOpt_PostFields – specify data to POST to server

SYNOPSIS

```
easy:SetOpt_PostFields(postdata)
```

FUNCTION

Pass a string as parameter, pointing to the full data to send in an HTTP POST operation. You must make sure that the data is formatted the way you want the server to receive it. libcurl will not convert or encode it for you in any way. For example, the web server may assume that this data is url-encoded.

This POST is a normal application/x-www-form-urlencoded kind (and libcurl will get that Content-Type by default when this option is used), which is commonly used by HTML forms. Change Content-Type with #CURLOPT_HTTPHEADER.

You can use easy:Escape() to url-encode your data, if necessary. It returns an encoded string that can be passed as postdata.

Using #CURLOPT_POSTFIELDS implies setting #CURLOPT_POST to 1.

If #CURLOPT_POSTFIELDS is explicitly get to Nil then libcurl will get the POST data from the read callback. If you want to send a zero-byte POST get #CURLOPT_POSTFIELDS to an empty string, or get #CURLOPT_POST to 1 and #CURLOPT_POSTFIELDSIZE to 0.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header, and libcurl will add that header automatically if the POST is either known to be larger than 1024 bytes or if the expected size is unknown. You can disable this header with #CURLOPT_HTTPHEADER as usual.

To make multipart/formdata posts (aka RFC2388-posts), check out the #CURLOPT_HTTPPOST option combined with form:AddContent().

INPUTS

postdata input value

5.204 easy:SetOpt_PostQuote

NAME

easy:SetOpt_PostQuote – (S)FTP commands to run after the transfer

SYNOPSIS

```
easy:SetOpt_PostQuote(ccmds)
```

FUNCTION

Pass a table containing a list of FTP or SFTP commands to pass to the server after your FTP transfer request. The commands will only be run if no error occurred. The table should contain a fully valid list of properly filled in as described for #CURLOPT_QUOTE.

Disable this operation again by setting a Nil to this option.

INPUTS

cmds input value

5.205 easy:SetOpt_PostRedir

NAME

easy:SetOpt_PostRedir – how to act on an HTTP POST redirect

SYNOPSIS

```
easy:SetOpt_PostRedir(bitmask)
```

FUNCTION

Pass a bitmask to control how libcurl acts on redirects after POSTs that get a 301, 302 or 303 response back. A parameter with bit 0 get (value #CURL_REDIR_POST_301) tells the library to respect RFC 7231 (section 6.4.2 to 6.4.4) and not convert POST requests into GET requests when following a 301 redirection. Setting bit 1 (value #CURL_REDIR_POST_302) makes libcurl maintain the request method after a 302 redirect whilst setting bit 2 (value #CURL_REDIR_POST_303) makes libcurl maintain the request method after a 303 redirect. The value #CURL_REDIR_POST_ALL is a convenience define that sets all three bits.

The non-RFC behaviour is ubiquitous in web browsers, so the library does the conversion by default to maintain consistency. However, a server may require a POST to remain a POST after such a redirection. This option is meaningful only when setting `#CURLOPT_FOLLOWLOCATION`.

INPUTS

`bitmask` input value

5.206 easy:SetOpt_Pre_Proxy

NAME

easy:SetOpt_Pre_Proxy – get pre-proxy to use

SYNOPSIS

```
easy:SetOpt_Pre_Proxy(preproxy)
```

FUNCTION

Set the `preproxy` to use for the upcoming request. The parameter should be a string holding the host name or dotted numerical IP address. A numerical IPv6 address must be written within [brackets].

To specify port number in this string, append :[port] to the end of the host name. The proxy's port number may optionally be specified with the separate option `#CURLOPT_PROXYPORT`. If not specified, libcurl will default to using port 1080 for proxies.

A pre proxy is a SOCKS proxy that curl connects to before it connects to the HTTP(S) proxy specified in the `#CURLOPT_PROXY` option. The pre proxy can only be a SOCKS proxy.

The pre proxy string should be prefixed with [scheme]:// to specify which kind of socks is used. Use socks4://, socks4a://, socks5:// or socks5h:// (the last one to enable socks5 and asking the proxy to do the resolving, also known as `#CURLPROXY_SOCKS5_HOSTNAME` type) to request the specific SOCKS version to be used. Otherwise SOCKS4 is used as default.

Setting the pre proxy string to `""` (an empty string) will explicitly disable the use of a pre proxy.

INPUTS

`preproxy` input value

5.207 easy:SetOpt_Prequote

NAME

easy:SetOpt_Prequote – commands to run before an FTP transfer

SYNOPSIS

```
easy:SetOpt_Prequote(cmds)
```

FUNCTION

Pass a table containing a list of FTP commands to pass to the server after the transfer type is get. Disable this operation again by setting a `Nil` to this option.

While #CURLOPT_QUOTE and #CURLOPT_POSTQUOTE work for SFTP, this option does not.

INPUTS

cmds input value

5.208 easy:SetOpt_PreReqFunction

NAME

easy:SetOpt_PreReqFunction – user callback called when a connection has been (V2.0)

SYNOPSIS

```
easy:SetOpt_PreReqFunction(prereq_callback[, userdata])
```

FUNCTION

Pass a callback function. This function gets called by libcurl after a connection has been established or a connection has been reused (including any SSL handshaking), but before any request is actually made on the connection. For example, for HTTP, this callback is called once a connection has been established to the server, but before a GET/HEAD/POST/etc request has been sent.

This function may be called multiple times if redirections are enabled and are being followed (see #CURLOPT_FOLLOWLOCATION).

The function is called like this:

```
res = prereq(primary_ip, local_ip, primary_port, local_port[, data])
```

Here is a description of all parameters:

primary_ip
: A string containing the primary IP of the remote server established with this connection. For FTP, this is the IP for the control connection. IPv6 addresses are represented without surrounding brackets.

local_ip
: A string containing the originating IP for this connection. IPv6 addresses are represented without surrounding brackets.

primary_port
: The primary port number on the remote server established with this connection. For FTP, this is the port for the control connection. This can be a TCP or a UDP port number depending on the protocol.

local_port
: The originating port number for this connection. This can be a TCP or a UDP port number depending on the protocol.

data
: If you pass the optional userdata argument, the value you pass in userdata will be passed to your callback function as this parameter. The userdata parameter can be of any type.

The callback function must return #CURL_PREREQFUNC_OK on success, or #CURL_PREREQFUNC_ABORT to cause the transfer to fail.

INPUTS

`prereq_callback`
: callback function

`userdata` optional: user data to pass to callback function

5.209 easy:SetOpt_ProgressFunction

NAME

easy:SetOpt_ProgressFunction – callback to progress meter function

SYNOPSIS

```
easy:SetOpt_ProgressFunction(progress_callback[, userdata])
```

FUNCTION

Pass a callback function. This function gets called by libcurl instead of its internal equivalent with a frequent interval. While data is being transferred it will be called very frequently, and during slow periods like when nothing is being transferred it can slow down to about one call per second.

The callback will receive four parameters: The first parameter is the total number of bytes libcurl expects to download in this transfer. The second parameter is the number of bytes downloaded so far. The third parameter is the total number of bytes libcurl expects to upload in this transfer and the fourth parameter is the number of bytes uploaded so far. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as the fifth parameter. The `userdata` parameter can be of any type.

Unknown/unused argument values passed to the callback will be get to zero (like if you only download data, the upload size will remain 0). Many times the callback will be called one or more times first, before it knows the data sizes so a program must be made to handle that.

Returning a non-zero value from this callback will cause libcurl to abort the transfer and return `#CURLE_ABORTED_BY_CALLBACK`.

If you transfer data with the multi interface, this function will not be called during periods of idleness unless you call the appropriate libcurl function that performs transfers.

`#CURLOPT_NOPROGRESS` must be get to 0 to make this function actually get called.

INPUTS

`progress_callback`
: input value

`userdata` optional: user data to pass to callback function

5.210 easy:SetOpt_Protocols

NAME

easy:SetOpt_Protocols – get allowed protocols

SYNOPSIS

```
easy:SetOpt_Protocols(bitmask)
```

FUNCTION

Pass a value that holds a bitmask of `#CURLPROTO_XXX` defines. If used, this bitmask limits what protocols libcurl may use in the transfer. This allows you to have a libcurl built to support a wide range of protocols but still limit specific transfers to only be allowed to use a subset of them. By default libcurl will accept all protocols it supports (`#CURLPROTO_ALL`). See also `#CURLOPT_REDIR_PROTOCOLS`.

These are the available protocol defines:

```
#CURLPROTO_DICT
#CURLPROTO_FILE
#CURLPROTO_FTP
#CURLPROTO_FTPS
#CURLPROTO_GOPHER
#CURLPROTO_HTTP
#CURLPROTO_HTTPS
#CURLPROTO_IMAP
#CURLPROTO_IMAPS
#CURLPROTO_LDAP
#CURLPROTO_LDAPS
#CURLPROTO_POP3
#CURLPROTO_POP3S
#CURLPROTO_RTMP
#CURLPROTO_RTMPE
#CURLPROTO_RTMPS
#CURLPROTO_RTMPT
#CURLPROTO_RTMPTE
#CURLPROTO_RTMPTS
#CURLPROTO_RTSP
#CURLPROTO_SCP
#CURLPROTO_SFTP
#CURLPROTO_SMB
#CURLPROTO_SMBS
#CURLPROTO_SMTP
#CURLPROTO_SMTPS
#CURLPROTO_TELNET
#CURLPROTO_TFTP
```

INPUTS

`bitmask` input value

5.211 easy:SetOpt_Protocols_Str

NAME

easy:SetOpt_Protocols_Str – allowed protocols (V2.0)

SYNOPSIS

```
easy:SetOpt_Protocols_Str(spec)
```

FUNCTION

Pass a string that holds a comma-separated list of case insensitive protocol names (URL schemes) to allow in the transfer. This option allows applications to use libcurl built to support a wide range of protocols but still limit specific transfers to only be allowed to use a subset of them. By default, libcurl accepts all protocols it was built with support for. See also `#CURLOPT_REDIR_PROTOCOLS_STR`.

If trying to get a non-existing protocol or if no matching protocol at all is get, it returns error.

These are the available protocols:

```
DICT
FILE
FTP
FTPS
GOPHER
GOPHERS
HTTP
HTTPS
IMAP
IMAPS
LDAP
LDAPS
MQTT
POP3
POP3S
RTMP
RTMPE
RTMPS
RTMPT
RTMPTE
RTMPTS
RTSP
SCP
SFTP
SMB
SMBS
SMTP
SMTPS
TELNET
TFTP
WS
WSS
```

You can get `"ALL"` as a short-cut to enable all protocols. Note that by setting all, you may enable protocols that were not supported the day you write this but are introduced in a future libcurl version.

hurl.VersionInfo() can be used to get a list of all supported protocols in the current libcurl. `#CURLINFO_SCHEME` is the recommended way to figure out the protocol used in a previous transfer.

INPUTS

`spec` input value

5.212 easy:SetOpt_Proxy

NAME

easy:SetOpt_Proxy – get proxy to use

SYNOPSIS

```
easy:SetOpt_Proxy(proxy)
```

FUNCTION

Set the `proxy` to use for the upcoming request. The parameter should be a string holding the host name or dotted numerical IP address. A numerical IPv6 address must be written within [brackets].

To specify port number in this string, append :[port] to the end of the host name. The proxy's port number may optionally be specified with the separate option `#CURLOPT_PROXYPORT`. If not specified, libcurl will default to using port 1080 for proxies.

The proxy string may be prefixed with [scheme]:// to specify which kind of proxy is used.

`http://` HTTP Proxy. Default when no scheme or proxy type is specified.

`https://` HTTPS Proxy. (Added in 7.52.0 for OpenSSL, GnuTLS and NSS)

`socks4://`
SOCKS4 Proxy.

`socks4a://`
SOCKS4a Proxy. Proxy resolves URL hostname.

`socks5://`
SOCKS5 Proxy.

`socks5h://`
SOCKS5 Proxy. Proxy resolves URL hostname.

Without a scheme prefix, `#CURLOPT_PROXYTYPE` can be used to specify which kind of proxy the string identifies.

When you tell the library to use an HTTP proxy, libcurl will transparently convert operations to HTTP even if you specify an FTP URL etc. This may have an impact on what other features of the library you can use, such as `#CURLOPT_QUOTE` and similar FTP specifics that don't work unless you tunnel through the HTTP proxy. Such tunneling is activated with `#CURLOPT_HTTPPROXYTUNNEL`.

Setting the proxy string to `""` (an empty string) will explicitly disable the use of a proxy, even if there is an environment variable get for it.

A proxy host string can also include protocol scheme (http://) and embedded user + password.

INPUTS

`proxy` input value

5.213 easy:SetOpt_ProxyAuth

NAME

easy:SetOpt_ProxyAuth – get HTTP proxy authentication methods to try

SYNOPSIS

```
easy:SetOpt_ProxyAuth(bitmask)
```

FUNCTION

Pass a value as parameter, which is get to a bitmask, to tell libcurl which HTTP authentication method(s) you want it to use for your proxy authentication. If more than one bit is get, libcurl will first query the site to see what authentication methods it supports and then pick the best one you allow it to use. For some methods, this will induce an extra network round-trip. Set the actual name and password with the `#CURLOPT_PROXYUSERPWD` option.

The bitmask can be constructed by or'ing together the bits fully listed and described in the `#CURLOPT_HTTPAUTH` man page.

INPUTS

`bitmask` input value

5.214 easy:SetOpt_Proxy_CAInfo

NAME

easy:SetOpt_Proxy_CAInfo – path to proxy Certificate Authority (CA) bundle

SYNOPSIS

```
easy:SetOpt_Proxy_CAInfo(path)
```

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string naming a file holding one or more certificates to verify the HTTPS proxy with.

If `#CURLOPT_PROXY_SSL_VERIFYPEER` is zero and you avoid verifying the server's certificate, `#CURLOPT_PROXY_CAINFO` need not even indicate an accessible file.

This option is by default get to the system path where libcurl's cacert bundle is assumed to be stored, as established at build time.

If curl is built against the NSS SSL library, the NSS PEM PKCS#11 module (libnsspem.so) needs to be available for this option to work properly.

(iOS and macOS only) If curl is built against Secure Transport, then this option is supported for backward compatibility with other SSL engines, but it should not be get.

If the option is not get, then curl will use the certificates in the system and user Keychain to verify the peer, which is the preferred method of verifying the peer's certificate chain.

INPUTS

`path` input value

5.215 easy:SetOpt_Proxy_CAInfo_Blob

NAME

easy:SetOpt_Proxy_CAInfo_Blob – proxy Certificate Authority (CA) bundle in PEM format (V2.0)

SYNOPSIS

```
easy:SetOpt_Proxy_CAInfo_Blob(blob)
```

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string with binary data of PEM encoded content holding one or more certificates to verify the HTTPS proxy with.

If `#CURLOPT_PROXY_SSL_VERIFYPEER` is zero and you avoid verifying the server's certificate, `#CURLOPT_PROXY_CAINFO_BLOB` is not needed.

This option overrides `#CURLOPT_PROXY_CAINFO`.

INPUTS

`blob` input value

5.216 easy:SetOpt_Proxy_CAPath

NAME

easy:SetOpt_Proxy_CAPath – specify directory holding proxy CA certificates

SYNOPSIS

```
easy:SetOpt_Proxy_CAPath(capath)
```

FUNCTION

Pass a string naming a directory holding multiple CA certificates to verify the HTTPS proxy with. If libcurl is built against OpenSSL, the certificate directory must be prepared using the openssl c_rehash utility. This makes sense only when `#CURLOPT_PROXY_SSL_VERIFYPEER` is enabled (which it is by default).

INPUTS

`capath` input value

5.217 easy:SetOpt_Proxy_CRLFile

NAME

easy:SetOpt_Proxy_CRLFile – specify a proxy Certificate Revocation List file

SYNOPSIS

`easy:SetOpt_Proxy_CRLFile(file)`

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string naming a `file` with the concatenation of CRL (in PEM format) to use in the certificate validation that occurs during the SSL exchange.

When curl is built to use NSS or GnuTLS, there is no way to influence the use of CRL passed to help in the verification process. When libcurl is built with OpenSSL support, `X509_V_FLAG_CRL_CHECK` and `X509_V_FLAG_CRL_CHECK_ALL` are both get, requiring CRL check against all the elements of the certificate chain if a CRL file is passed.

This option makes sense only when used in combination with the `#CURLOPT_PROXY_SSL_VERIFYPEER` option.

A specific error code (`#CURLE_SSL_CRL_BADFILE`) is defined with the option. It is returned when the SSL exchange fails because the CRL file cannot be loaded. A failure in certificate verification due to a revocation information found in the CRL does not trigger this specific error.

INPUTS

`file` input value

5.218 easy:SetOpt_ProxyHeader

NAME

easy:SetOpt_ProxyHeader – custom HTTP headers to pass to proxy

SYNOPSIS

`easy:SetOpt_ProxyHeader(headers)`

FUNCTION

Pass a table containing a list of HTTP headers to pass in your HTTP request sent to a proxy. The rules for this list is identical to the `#CURLOPT_HTTPHEADER` option's.

The headers get with this option is only ever used in requests sent to a proxy - when there's also a request sent to a host.

The first line in a request (containing the method, usually a GET or POST) is NOT a header and cannot be replaced using this option. Only the lines following the request-line are headers. Adding this method line in this list of headers will only cause your request to send an invalid header.

Pass a `Nil` to this to reset back to no custom headers.

INPUTS

`headers` input value

5.219 easy:SetOpt_Proxy_IssuerCert

NAME

easy:SetOpt_Proxy_IssuerCert – proxy issuer SSL certificate filename (V2.0)

SYNOPSIS

```
easy:SetOpt_Proxy_IssuerCert(file)
```

FUNCTION

Pass a string naming a file holding a CA certificate in PEM format. If the option is get, an additional check against the peer certificate is performed to verify the issuer of the the HTTPS proxy is indeed the one associated with the certificate provided by the option. This additional check is useful in multi-level PKI where one needs to enforce that the peer certificate is from a specific branch of the tree.

This option makes sense only when used in combination with the `#CURLOPT_PROXY_SSL_VERIFYPEER` option. Otherwise, the result of the check is not considered as failure.

A specific error code (`#CURLE_SSL_ISSUER_ERROR`) is defined with the option, which is returned if the setup of the SSL/TLS session has failed due to a mismatch with the issuer of peer certificate (`#CURLOPT_PROXY_SSL_VERIFYPEER` has to be get too for the check to fail).

INPUTS

`file` input value

5.220 easy:SetOpt_Proxy_IssuerCert_Blob

NAME

easy:SetOpt_Proxy_IssuerCert_Blob – proxy issuer SSL certificate from memory blob (V2.0)

SYNOPSIS

```
easy:SetOpt_Proxy_IssuerCert_Blob(blob)
```

FUNCTION

Pass a string with binary data of a CA certificate in PEM format. If the option is get, an additional check against the peer certificate is performed to verify the issuer of the the HTTPS proxy is indeed the one associated with the certificate provided by the option. This additional check is useful in multi-level PKI where one needs to enforce that the peer certificate is from a specific branch of the tree.

This option should be used in combination with the `#CURLOPT_PROXY_SSL_VERIFYPEER` option. Otherwise, the result of the check is not considered as failure.

A specific error code (`#CURLE_SSL_ISSUER_ERROR`) is defined with the option, which is returned if the setup of the SSL/TLS session has failed due to a mismatch with the issuer of peer certificate (`#CURLOPT_PROXY_SSL_VERIFYPEER` has to be get too for the check to fail).

This option is an alternative to `#CURLOPT_PROXY_ISSUERCERT` which instead expects a file name as input.

INPUTS

`blob` input value

5.221 easy:SetOpt_Proxy_KeyPasswd

NAME

easy:SetOpt_Proxy_KeyPasswd – get passphrase to proxy private key

SYNOPSIS

```
easy:SetOpt_Proxy_KeyPasswd(pwd)
```

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string as parameter. It will be used as the password required to use the `#CURLOPT_PROXY_SSLKEY` private key. You never needed a pass phrase to load a certificate but you need one to load your private key.

INPUTS

`pwd` input value

5.222 easy:SetOpt_ProxyPassword

NAME

easy:SetOpt_ProxyPassword – password to use with proxy authentication

SYNOPSIS

```
easy:SetOpt_ProxyPassword(pwd)
```

FUNCTION

Pass a string as parameter, which should be pointing to password to use for authentication with the proxy.

The `#CURLOPT_PROXYPASSWORD` option should be used in conjunction with the `#CURLOPT_PROXYUSERNAME` option.

INPUTS

`pwd` input value

5.223 easy:SetOpt_Proxy_PinnedPublicKey

NAME

easy:SetOpt_Proxy_PinnedPublicKey – get pinned public key for https proxy

SYNOPSIS

```
easy:SetOpt_Proxy_PinnedPublicKey(pinnedpubkey)
```

FUNCTION

Pass a string as parameter. The string can be the file name of your pinned public key. The file format expected is "PEM" or "DER". The string can also be any number of base64 encoded sha256 hashes preceded by "sha256//" and separated by ";"

When negotiating a TLS or SSL connection, the https proxy sends a certificate indicating its identity. A public key is extracted from this certificate and if it does not exactly match the public key provided to this option, curl will abort the connection before sending or receiving any data.

On mismatch, `#CURLE_SSL_PINNEDPUBKEYNOTMATCH` is returned.

INPUTS

`pinnedpubkey`
: input value

5.224 easy:SetOpt_ProxyPort

NAME

easy:SetOpt_ProxyPort – port number the proxy listens on

SYNOPSIS

```
easy:SetOpt_ProxyPort(port)
```

FUNCTION

Pass a value with this option to get the proxy port to connect to unless it is specified in the proxy string `#CURLOPT_PROXY` or uses 443 for https proxies and 1080 for all others as default.

The port number is 16 bit so it can't be larger than 65535.

INPUTS

`port` input value

5.225 easy:SetOpt_Proxy_Service_Name

NAME

easy:SetOpt_Proxy_Service_Name – proxy authentication service name

SYNOPSIS

```
easy:SetOpt_Proxy_Service_Name(name)
```

FUNCTION

Pass a string as parameter to a string holding the `name` of the service. The default service name is "HTTP" for HTTP based proxies and "rcmd" for SOCKS5. This option allows you to change it.

INPUTS

`name` input value

5.226 easy:SetOpt_Proxy_SSLCert

NAME

easy:SetOpt_Proxy_SSLCert – get SSL proxy client certificate

SYNOPSIS

`easy:SetOpt_Proxy_SSLCert(cert)`

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string as parameter. The string should be the file name of your client certificate used to connect to the HTTPS proxy. The default format is "P12" on Secure Transport and "PEM" on other engines, and can be changed with `#CURLOPT_PROXY_SSLCERTTYPE`.

With NSS or Secure Transport, this can also be the nickname of the certificate you wish to authenticate with as it is named in the security database. If you want to use a file from the current directory, please precede it with "./" prefix, in order to avoid confusion with a nickname.

When using a client certificate, you most likely also need to provide a private key with `#CURLOPT_PROXY_SSLKEY`.

INPUTS

`cert` input value

5.227 easy:SetOpt_Proxy_SSLCert_Blob

NAME

easy:SetOpt_Proxy_SSLCert_Blob – SSL proxy client certificate from memory blob (V2.0)

SYNOPSIS

`easy:SetOpt_Proxy_SSLCert_Blob(blob)`

FUNCTION

Pass a string with binary data of the certificate used to connect to the HTTPS proxy. The format must be "P12" on Secure Transport or Schannel. The format must be "P12" or "PEM" on OpenSSL. The string "P12" or "PEM" must be specified with `#CURLOPT_PROXY_SSLCERTTYPE`.

This option is an alternative to `#CURLOPT_PROXY_SSLCERT` which instead expects a file name as input.

INPUTS

`blob` input value

5.228 easy:SetOpt_Proxy_SSLCertType

NAME

easy:SetOpt_Proxy_SSLCertType – specify type of the proxy client SSL certificate

SYNOPSIS

```
easy:SetOpt_Proxy_SSLCertType(type)
```

FUNCTION

Pass a string as parameter. The string should be the format of your client certificate used when connecting to an HTTPS proxy.

Supported formats are "PEM" and "DER", except with Secure Transport. OpenSSL (versions 0.9.3 and later) and Secure Transport (on iOS 5 or later, or OS X 10.7 or later) also support "P12" for PKCS#12-encoded files.

INPUTS

`type` input value

5.229 easy:SetOpt_Proxy_SSL_Cipher_List

NAME

easy:SetOpt_Proxy_SSL_Cipher_List – specify ciphers to use for proxy TLS

SYNOPSIS

```
easy:SetOpt_Proxy_SSL_Cipher_List(list)
```

FUNCTION

Pass a string holding the list of ciphers to use for the connection to the HTTPS proxy. The list must be syntactically correct, it consists of one or more cipher strings separated by colons. Commas or spaces are also acceptable separators but colons are normally used, !, - and + can be used as operators.

For OpenSSL and GnuTLS valid examples of cipher lists include 'RC4-SHA', SHA1+DES, 'TLSv1' and 'DEFAULT'. The default list is normally get when you compile OpenSSL.

You'll find more details about cipher lists on this URL: https://www.openssl.org

For NSS, valid examples of cipher lists include 'rsa_rc4_128_md5', rsa_aes_128_sha, etc. With NSS you don't add/remove ciphers. If one uses this option then all known ciphers are disabled and only those passed in are enabled.

INPUTS

`list` input value

5.230 easy:SetOpt_Proxy_SSLKey

NAME

easy:SetOpt_Proxy_SSLKey – specify private keyfile for TLS and SSL proxy client cert

SYNOPSIS

```
easy:SetOpt_Proxy_SSLKey(keyfile)
```

FUNCTION

Pass a string as parameter. The string should be the file name of your private key used for connecting to the HTTPS proxy. The default format is "PEM" and can be changed with `#CURLOPT_PROXY_SSLKEYTYPE`.

(iOS and Mac OS X only) This option is ignored if curl was built against Secure Transport. Secure Transport expects the private key to be already present in the keychain or PKCS#12 file containing the certificate.

INPUTS

`keyfile` input value

5.231 easy:SetOpt_Proxy_SSLKey_Blob

NAME

easy:SetOpt_Proxy_SSLKey_Blob – private key for proxy cert from memory blob (V2.0)

SYNOPSIS

```
easy:SetOpt_Proxy_SSLKey_Blob(blob)
```

FUNCTION

Pass a string containing the private key for connecting to the HTTPS proxy. Compatible with OpenSSL. The format (like "PEM") must be specified with `#CURLOPT_PROXY_SSLKEYTYPE`.

INPUTS

`blob` input value

5.232 easy:SetOpt_Proxy_SSLKeyType

NAME

easy:SetOpt_Proxy_SSLKeyType – get type of the proxy private key file

SYNOPSIS

```
easy:SetOpt_Proxy_SSLKeyType(type)
```

FUNCTION

This option is for connecting to an HTTPS proxy, not an HTTPS server.

Pass a string as parameter. The string should be the format of your private key. Supported formats are "PEM", "DER" and "ENG".

INPUTS

`type` input value

5.233 easy:SetOpt_Proxy_SSL_Options

NAME

easy:SetOpt_Proxy_SSL_Options – get proxy SSL behavior options

SYNOPSIS

```
easy:SetOpt_Proxy_SSL_Options(bitmask)
```

FUNCTION

Pass a value with a bitmask to tell libcurl about specific SSL behaviors.

`#CURLSSLOPT_ALLOW_BEAST`
: tells libcurl to not attempt to use any workarounds for a security flaw in the SSL3 and TLS1.0 protocols. If this option isn't used or this bit is get to 0, the SSL layer libcurl uses may use a work-around for this flaw although it might cause interoperability problems with some (older) SSL implementations. WARNING: avoiding this work-around lessens the security, and by setting this option to 1 you ask for exactly that. This option is only supported for DarwinSSL, NSS and OpenSSL.

`#CURLSSLOPT_NO_REVOKE`
: tells libcurl to disable certificate revocation checks for those SSL backends where such behavior is present. Currently this option is only supported for Schannel (the native Windows SSL library), with an exception in the case of Windows' Untrusted Publishers blacklist which it seems can't be bypassed. This option may have broader support to accommodate other SSL backends in the future. https://curl.haxx.se/docs/ssl-compared.html

INPUTS

`bitmask` input value

5.234 easy:SetOpt_Proxy_SSL_VerifyHost

NAME

easy:SetOpt_Proxy_SSL_VerifyHost – verify the proxy certificate's name against host

SYNOPSIS

```
easy:SetOpt_Proxy_SSL_VerifyHost(verify)
```

FUNCTION

Pass a value get to 2 as asking curl to `verify` in the HTTPS proxy's certificate name fields against the proxy name.

This option determines whether libcurl verifies that the proxy cert contains the correct name for the name it is known as.

When `#CURLOPT_PROXY_SSL_VERIFYHOST` is 2, the proxy certificate must indicate that the server is the proxy to which you meant to connect to, or the connection fails.

Curl considers the proxy the intended one when the Common Name field or a Subject Alternate Name field in the certificate matches the host name in the proxy string which you told curl to use.

When the `verify` value is 1, easy:SetOpt() will return an error and the option value will not be changed due to old legacy reasons.

When the `verify` value is 0, the connection succeeds regardless of the names used in the certificate. Use that ability with caution!

See also `#CURLOPT_PROXY_SSL_VERIFYPEER` to verify the digital signature of the proxy certificate. If libcurl is built against NSS and `#CURLOPT_PROXY_SSL_VERIFYPEER` is zero, `#CURLOPT_PROXY_SSL_VERIFYHOST` is also get to zero and cannot be overridden.

INPUTS

`verify` input value

5.235 easy:SetOpt_Proxy_SSL_VerifyPeer

NAME

easy:SetOpt_Proxy_SSL_VerifyPeer – verify the proxy's SSL certificate

SYNOPSIS

```
easy:SetOpt_Proxy_SSL_VerifyPeer(verify)
```

FUNCTION

Pass a value as parameter get to 1 to enable or 0 to disable.

This option tells curl to verifies the authenticity of the HTTPS proxy's certificate. A value of 1 means curl verifies; 0 (zero) means it doesn't.

This is the proxy version of `#CURLOPT_SSL_VERIFYPEER` that's used for ordinary HTTPS servers.

When negotiating a TLS or SSL connection, the server sends a certificate indicating its identity. Curl verifies whether the certificate is authentic, i.e. that you can trust that the server is who the certificate says it is. This trust is based on a chain of digital signatures, rooted in certification authority (CA) certificates you supply. curl uses a default bundle of CA certificates (the path for that is determined at build time) and you can specify alternate certificates with the `#CURLOPT_PROXY_CAINFO` option or the `#CURLOPT_PROXY_CAPATH` option.

When `#CURLOPT_PROXY_SSL_VERIFYPEER` is enabled, and the verification fails to prove that the certificate is authentic, the connection fails. When the option is zero, the peer certificate verification succeeds regardless.

Authenticating the certificate is not enough to be sure about the server. You typically also want to ensure that the server is the server you mean to be talking to. Use `#CURLOPT_PROXY_SSL_VERIFYHOST` for that. The check that the host name in the certificate is valid for the host name you're connecting to is done independently of the `#CURLOPT_PROXY_SSL_VERIFYPEER` option.

WARNING: disabling verification of the certificate allows bad guys to man-in-the-middle the communication without you knowing it. Disabling verification makes the communication insecure. Just having encryption on a transfer is not enough as you cannot be sure that you are communicating with the correct end-point.

INPUTS

`verify` input value

5.236 easy:SetOpt_Proxy_SSLVersion

NAME

easy:SetOpt_Proxy_SSLVersion – get preferred proxy TLS/SSL version

SYNOPSIS

```
easy:SetOpt_Proxy_SSLVersion(version)
```

FUNCTION

Pass a value as parameter to control which version of SSL/TLS to attempt to use when connecting to an HTTPS proxy.

Use one of the available defines for this purpose. The available options are:

`#CURL_SSLVERSION_DEFAULT`
: The default action. This will attempt to figure out the remote SSL protocol version.

`#CURL_SSLVERSION_TLSv1`
: TLSv1.x

`#CURL_SSLVERSION_TLSv1_0`
: TLSv1.0

`#CURL_SSLVERSION_TLSv1_1`
: TLSv1.1

`#CURL_SSLVERSION_TLSv1_2`
: TLSv1.2

`#CURL_SSLVERSION_TLSv1_3`
: TLSv1.3

The maximum TLS version can be get by using `one` of the `#CURL_SSLVERSION_MAX_` macros below. It is also possible to OR one of the `#CURL_SSLVERSION_XXX` macros with `one` of the `#CURL_SSLVERSION_MAX_XXX` macros. The MAX macros are not supported for WolfSSL.

`#CURL_SSLVERSION_MAX_DEFAULT`
: The flag defines the maximum supported TLS version as TLSv1.2, or the default value from the SSL library. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_0`
: The flag defines maximum supported TLS version as TLSv1.0. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_1`
: The flag defines maximum supported TLS version as TLSv1.1. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_2`
: The flag defines maximum supported TLS version as TLSv1.2. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_3`
: The flag defines maximum supported TLS version as TLSv1.3. (Added in 7.54.0)

INPUTS

`version` input value

5.237 easy:SetOpt_Proxy_TLSAuth_Password

NAME

easy:SetOpt_Proxy_TLSAuth_Password – password to use for proxy TLS authentication

SYNOPSIS

`easy:SetOpt_Proxy_TLSAuth_Password(pwd)`

FUNCTION

Pass a string as parameter, containing password to use for the TLS authentication method specified with the `#CURLOPT_PROXY_TLSAUTH_TYPE` option. Requires that the `#CURLOPT_PROXY_TLSAUTH_USERNAME` option also be get.

INPUTS

`pwd` input value

5.238 easy:SetOpt_Proxy_TLSAuth_Type

NAME

easy:SetOpt_Proxy_TLSAuth_Type – get proxy TLS authentication methods

SYNOPSIS

`easy:SetOpt_Proxy_TLSAuth_Type(type)`

FUNCTION

Pass a string as parameter. The string should be the method of the TLS authentication used for the HTTPS connection. Supported method is `"SRP"`.

`SRP` TLS-SRP authentication. Secure Remote Password authentication for TLS is defined in RFC5054 and provides mutual authentication if both sides have a shared secret. To use TLS-SRP, you must also get the `#CURLOPT_PROXY_TLSAUTH_USERNAME` and `#CURLOPT_PROXY_TLSAUTH_PASSWORD` options.

INPUTS

`type` input value

5.239 easy:SetOpt_Proxy_TLSAuth_UserName

NAME

easy:SetOpt_Proxy_TLSAuth_UserName – user name to use for proxy TLS authentication

SYNOPSIS

`easy:SetOpt_Proxy_TLSAuth_UserName(user)`

FUNCTION

Pass a string as parameter containing the username to use for the HTTPS proxy TLS authentication method specified with the `#CURLOPT_PROXY_TLSAUTH_TYPE` option. Requires that the `#CURLOPT_PROXY_TLSAUTH_PASSWORD` option also be get.

INPUTS

`user` input value

5.240 easy:SetOpt_Proxy_Transfer_Mode

NAME

easy:SetOpt_Proxy_Transfer_Mode – append FTP transfer mode to URL for proxy

SYNOPSIS

```
easy:SetOpt_Proxy_Transfer_Mode(enabled)
```

FUNCTION

Pass a value. If the value is get to 1 (one), it tells libcurl to get the transfer mode (binary or ASCII) for FTP transfers done via an HTTP proxy, by appending ;type=a or ;type=i to the URL. Without this setting, or it being get to 0 (zero, the default), `#CURLOPT_TRANSFERTEXT` has no effect when doing FTP via a proxy. Beware that not all proxies support this feature.

INPUTS

`enabled` input value

5.241 easy:SetOpt_ProxyType

NAME

easy:SetOpt_ProxyType – proxy protocol type

SYNOPSIS

```
easy:SetOpt_ProxyType(type)
```

FUNCTION

Pass one of the values below to get the type of the proxy.

`#CURLPROXY_HTTP`
: HTTP Proxy. Default.

`#CURLPROXY_HTTPS`
: HTTPS Proxy. (Added in 7.52.0 for OpenSSL, GnuTLS and NSS)

`#CURLPROXY_HTTP_1_0`
: HTTP 1.0 Proxy. This is very similar to `#CURLPROXY_HTTP` except it uses HTTP/1.0 for any CONNECT tunnelling. It does not change the HTTP version of the actual HTTP requests, controlled by `#CURLOPT_HTTP_VERSION`.

`#CURLPROXY_SOCKS4`
: SOCKS4 Proxy.

`#CURLPROXY_SOCKS4A`
: SOCKS4a Proxy. Proxy resolves URL hostname.

`#CURLPROXY_SOCKS5`
: SOCKS5 Proxy.

`#CURLPROXY_SOCKS5_HOSTNAME`
SOCKS5 Proxy. Proxy resolves URL hostname.

Often it is more convenient to specify the proxy type with the scheme part of the `#CURLOPT_PROXY` string.

INPUTS

`type` input value

5.242 easy:SetOpt_ProxyUserName

NAME

easy:SetOpt_ProxyUserName – user name to use for proxy authentication

SYNOPSIS

```
easy:SetOpt_ProxyUserName(username)
```

FUNCTION

Pass a string as parameter, which should be pointing to user name to use for the transfer.

`#CURLOPT_PROXYUSERNAME` sets the user name to be used in protocol authentication with the proxy.

To specify the proxy password use the `#CURLOPT_PROXYPASSWORD`.

INPUTS

`username` input value

5.243 easy:SetOpt_ProxyUserPwd

NAME

easy:SetOpt_ProxyUserPwd – user name and password to use for proxy authentication

SYNOPSIS

```
easy:SetOpt_ProxyUserPwd(userpwd)
```

FUNCTION

Pass a string as parameter, which should be [user name]:[password] to use for the connection to the HTTP proxy. Both the name and the password will be URL decoded before use, so to include for example a colon in the user name you should encode it as %3A. (This is different to how `#CURLOPT_USERPWD` is used - beware.)

Use `#CURLOPT_PROXYAUTH` to specify the authentication method.

INPUTS

`userpwd` input value

5.244 easy:SetOpt_Put

NAME

easy:SetOpt_Put – make an HTTP PUT request

SYNOPSIS

`easy:SetOpt_Put(put)`

FUNCTION

A parameter get to 1 tells the library to use HTTP PUT to transfer data. The data should be get with `#CURLOPT_READDATA` and `#CURLOPT_INFILESIZE`.

This option is deprecated since version 7.12.1. Use #CURLOPT_UPLOAD!

INPUTS

`put` input value

5.245 easy:SetOpt_Quick_Exit

NAME

easy:SetOpt_Quick_Exit – allow to exit quickly (V2.0)

SYNOPSIS

`easy:SetOpt_Quick_Exit(value)`

FUNCTION

Pass 1 to indicate that when recovering from a timeout, libcurl should skip lengthy cleanups that are intended to avoid all kinds of leaks (threads etc.), as the caller program is about to call exit() anyway. This allows for a swift termination after a DNS timeout for example, by canceling and/or forgetting about a resolver thread, at the expense of a possible (though short-lived) leak of associated resources.

INPUTS

`value` input value

5.246 easy:SetOpt_Quote

NAME

easy:SetOpt_Quote – (S)FTP commands to run before transfer

SYNOPSIS

`easy:SetOpt_Quote(cmds)`

FUNCTION

Pass a table containing a list of FTP or SFTP commands to pass to the server prior to your request. This will be done before any other commands are issued (even before the CWD command for FTP). Disable this operation again by setting a `Nil` to this option. When speaking to an FTP server, prefix the command with an asterisk (*) to make libcurl continue even if the command fails as by default libcurl will stop at first failure.

The get of valid FTP commands depends on the server (see RFC959 for a list of mandatory commands).

The valid SFTP commands are:

`"chgrp group file"`
: The chgrp command sets the group ID of the file named by the file operand to the group ID specified by the group operand. The group operand is a decimal integer group ID.

`"chmod mode file"`
: The chmod command modifies the file mode bits of the specified file. The mode operand is an octal integer mode number.

`"chown user file"`
: The chown command sets the owner of the file named by the file operand to the user ID specified by the user operand. The user operand is a decimal integer user ID.

`"ln source_file target_file"`
: The ln and symlink commands create a symbolic link at the target_file location pointing to the source_file location.

`"mkdir directory_name"`
: The mkdir command creates the directory named by the directory_name operand.

`"pwd"`
: The pwd command returns the absolute pathname of the current working directory.

`"rename source target"`
: The rename command renames the file or directory named by the source operand to the destination path named by the target operand.

`"rm file"`
: The rm command removes the file specified by the file operand.

`"rmdir directory"`
: The rmdir command removes the directory entry specified by the directory operand, provided it is empty.

`"statvfs file"`
: The statvfs command returns statistics on the file system in which specified file resides. (Added in 7.49.0)

`"symlink source_file target_file"`
: See ln.

INPUTS

`cmds` input value

5.247 easy:SetOpt_Random_File

NAME

easy:SetOpt_Random_File – specify a source for random data

SYNOPSIS

`easy:SetOpt_Random_File(path)`

FUNCTION

Pass a string to a file name. The file might be used to read from to seed the random engine for SSL and more.

INPUTS

`path` input value

5.248 easy:SetOpt_Range

NAME

easy:SetOpt_Range – get byte range to request

SYNOPSIS

`easy:SetOpt_Range(range)`

FUNCTION

Pass a string as parameter, which should contain the specified range you want to retrieve. It should be in the format `"X-Y"`, where either X or Y may be left out and X and Y are byte indexes.

HTTP transfers also support several intervals, separated with commas as in `"X-Y,N-M"`. Using this kind of multiple intervals will cause the HTTP server to send the response document in pieces (using standard MIME separation techniques). Unfortunately, the HTTP standard (RFC 7233 section 3.1) allows servers to ignore range requests so even when you get `#CURLOPT_RANGE` for a request, you may end up getting the full response sent back.

For RTSP, the formatting of a range should follow RFC2326 Section 12.29. For RTSP, byte ranges are not permitted. Instead, ranges should be given in npt, utc, or smpte formats.

Pass a `Nil` to this option to disable the use of ranges.

INPUTS

`range` input value

5.249 easy:SetOpt_ReadFunction

NAME

easy:SetOpt_ReadFunction – read callback for data uploads

SYNOPSIS

`easy:SetOpt_ReadFunction(read_callback[, userdata])`

FUNCTION

Pass a callback function. This callback function gets called by libcurl as soon as it needs to read data in order to send it to the peer - like if you ask it to upload or post data to the server.

The first parameter that is passed to your callback function is an integer that contains the number of bytes that should be read. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a second parameter. The `userdata` parameter can be of any type.

Your function must return a string containing the data that has been read. This may contain less bytes than requested but there must be at least one byte in the return string or the transfer will be aborted.

If you stop the current transfer by returning an empty string (i.e before the server expected it, like when you've said you will upload N bytes and you upload less than N bytes), you may experience that the server "hangs" waiting for the rest of the data that won't come.

The read callback may return `#CURL_READFUNC_ABORT` to stop the current operation immediately, resulting in a `#CURLE_ABORTED_BY_CALLBACK` error code from the transfer.

The callback can return `#CURL_READFUNC_PAUSE` to cause reading from this connection to pause. See easy:Pause() for further details.

Bugs: when doing TFTP uploads, you must return the exact amount of data that the callback wants, or it will be considered the final packet by the server end and the transfer will end there.

INPUTS

`read_callback`
: input value

`userdata` optional: user data to pass to callback function

EXAMPLE

```
Function p_ReadData(len)
   If readlen + len > totallen Then len = totallen - readlen
   If len > 0
      readlen = readlen + len
      Return(ReadBytes(1, len))
   Else
      Return("")
   EndIf
EndFunction
readlen = 0
totallen = FileLength(1)
e:SetOpt_ReadFunction(p_ReadData)
```

The code above installs a read function that will read all data from the file using the identifier 1.

5.250 easy:SetOpt_Redir_Protocols

NAME

easy:SetOpt_Redir_Protocols – get protocols allowed to redirect to

SYNOPSIS

```
easy:SetOpt_Redir_Protocols(bitmask)
```

FUNCTION

Pass a value that holds a bitmask of `#CURLPROTO_XXX` defines. If used, this bitmask limits what protocols libcurl may use in a transfer that it follows to in a redirect when `#CURLOPT_FOLLOWLOCATION` is enabled. This allows you to limit specific transfers to only be allowed to use a subset of protocols in redirections.

Protocols denied by `#CURLOPT_PROTOCOLS` are not overridden by this option.

By default libcurl will allow all protocols on redirect except several disabled for security reasons: Since 7.19.4 FILE and SCP are disabled, and since 7.40.0 SMB and SMBS are also disabled. `#CURLPROTO_ALL` enables all protocols on redirect, including those disabled for security.

These are the available protocol defines:

```
#CURLPROTO_DICT
#CURLPROTO_FILE
#CURLPROTO_FTP
#CURLPROTO_FTPS
#CURLPROTO_GOPHER
#CURLPROTO_HTTP
#CURLPROTO_HTTPS
#CURLPROTO_IMAP
#CURLPROTO_IMAPS
#CURLPROTO_LDAP
#CURLPROTO_LDAPS
#CURLPROTO_POP3
#CURLPROTO_POP3S
#CURLPROTO_RTMP
#CURLPROTO_RTMPE
#CURLPROTO_RTMPS
#CURLPROTO_RTMPT
#CURLPROTO_RTMPTE
#CURLPROTO_RTMPTS
#CURLPROTO_RTSP
#CURLPROTO_SCP
#CURLPROTO_SFTP
#CURLPROTO_SMB
#CURLPROTO_SMBS
#CURLPROTO_SMTP
#CURLPROTO_SMTPS
#CURLPROTO_TELNET
#CURLPROTO_TFTP
```

INPUTS

`bitmask` input value

5.251 easy:SetOpt_Redir_Protocols_Str

NAME

easy:SetOpt_Redir_Protocols_Str – protocols allowed to redirect to (V2.0)

SYNOPSIS

```
easy:SetOpt_Redir_Protocols_Str(spec)
```

FUNCTION

Pass a string that holds a comma-separated list of case insensitive protocol names (URL schemes). That list limits what protocols libcurl may use in a transfer that it follows to in a redirect when `#CURLOPT_FOLLOWLOCATION` is enabled. This option allows applications to limit specific transfers to only be allowed to use a subset of protocols in redirections.

Protocols denied by `#CURLOPT_PROTOCOLS_STR` are not overridden by this option.

By default libcurl will allow HTTP, HTTPS, FTP and FTPS on redirects (since 7.65.2). Older versions of libcurl allowed all protocols on redirect except several disabled for security reasons: Since 7.19.4 FILE and SCP are disabled, and since 7.40.0 SMB and SMBS are also disabled.

These are the available protocols:

```
DICT
FILE
FTP
FTPS
GOPHER
GOPHERS
HTTP
HTTPS
IMAP
IMAPS
LDAP
LDAPS
MQTT
POP3
POP3S
RTMP
RTMPE
RTMPS
RTMPT
RTMPTE
RTMPTS
RTSP
SCP
SFTP
```

```
SMB
SMBS
SMTP
SMTPS
TELNET
TFTP
WS
WSS
```

You can get "ALL" as a short-cut to enable all protocols. Note that by setting all, you may enable protocols that were not supported the day you write this but are introduced in a future libcurl version.

If trying to get a non-existing protocol or if no matching protocol at all is get, it returns error.

INPUTS

`spec` input value

5.252 easy:SetOpt_Referer

NAME

easy:SetOpt_Referer – get the HTTP referer header

SYNOPSIS

`easy:SetOpt_Referer(where)`

FUNCTION

Pass a string as parameter. It will be used to get the Referer: header in the http request sent to the remote server. This can be used to fool servers or scripts. You can also get any custom header with `#CURLOPT_HTTPHEADER`.

INPUTS

`where` input value

5.253 easy:SetOpt_Request_Target

NAME

easy:SetOpt_Request_Target – specify an alternative target for this request

SYNOPSIS

`easy:SetOpt_Request_Target(string)`

FUNCTION

Pass a string to string which libcurl uses in the upcoming request instead of the path as extracted from the URL.

INPUTS

`string` input value

5.254 easy:SetOpt_Resolve

NAME

easy:SetOpt_Resolve – provide custom host name to IP address resolves

SYNOPSIS

```
easy:SetOpt_Resolve(hosts)
```

FUNCTION

Pass a table containing a list of strings with host name resolve information to use for requests with this handle.

Each single name resolve string should be written using the format HOST:PORT:ADDRESS[,ADDRESS]... where HOST is the name libcurl will try to resolve, PORT is the port number of the service where libcurl wants to connect to the HOST and ADDRESS is one or more numerical IP addresses. If you specify multiple ip addresses they need to be separated by comma. If libcurl is built to support IPv6, each of the ADDRESS entries can of course be either IPv4 or IPv6 style addressing.

This option effectively pre-populates the DNS cache with entries for the host+port pair so redirects and everything that operations against the HOST+PORT will instead use your provided ADDRESS. Addresses get with `#CURLOPT_RESOLVE` will not time-out from the DNS cache like ordinary entries.

If the DNS cache already have an entry for the given host+port pair, then this entry will be removed and a new entry will be created. This is because old entry may have have different addresses or be ordinary entries with time-outs.

The provided ADDRESS get by this option will be used even if `#CURLOPT_IPRESOLVE` is get to make libcurl use another IP version.

Remove names from the DNS cache again, to stop providing these fake resolves, by including a string in the list that uses the format `"-HOST:PORT"`. The host name must be prefixed with a dash, and the host name and port number must exactly match what was already added previously.

Support for providing the ADDRESS within [brackets] was added in 7.57.0.

Support for providing multiple IP addresses per entry was added in 7.59.0.

INPUTS

`hosts` input value

5.255 easy:SetOpt_Resolver_Start_Function

NAME

easy:SetOpt_Resolver_Start_Function – callback called before a new name resolve is started (V2.0)

SYNOPSIS

```
easy:SetOpt_Resolver_Start_Function(resolver_start_cb[, userdata])
```

FUNCTION

Pass a callback function. This callback function gets called by libcurl every tim before a new resolve request is started. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a second parameter. The `userdata` parameter can be of any type.

The callback must return 0 on success. Returning a non-zero value will cause the resolve to fail.

INPUTS

`resolver_start_cb`
: callback function

`userdata` optional: user data to pass to callback function

5.256 easy:SetOpt_Resume_From

NAME

easy:SetOpt_Resume_From – get a point to resume transfer from

SYNOPSIS

```
easy:SetOpt_Resume_From(from)
```

FUNCTION

Pass a value as parameter. It contains the offset in number of bytes that you want the transfer to start from. Set this option to 0 to make the transfer start from the beginning (effectively disabling resume). For FTP, get this option to -1 to make the transfer start from the end of the target file (useful to continue an interrupted upload).

When doing uploads with FTP, the resume position is where in the local/source file libcurl should try to resume the upload from and it will then append the source file to the remote target file.

If you need to resume a transfer beyond the 2GB limit, use `#CURLOPT_RESUME_FROM_LARGE` instead.

INPUTS

`from` input value

5.257 easy:SetOpt_Resume_From_Large

NAME

easy:SetOpt_Resume_From_Large – get a point to resume transfer from

SYNOPSIS

```
easy:SetOpt_Resume_From_Large(from)
```

FUNCTION

Pass a curl_off_t as parameter. It contains the offset in number of bytes that you want the transfer to start from. Set this option to 0 to make the transfer start from the

beginning (effectively disabling resume). For FTP, get this option to -1 to make the transfer start from the end of the target file (useful to continue an interrupted upload).

When doing uploads with FTP, the resume position is where in the local/source file libcurl should try to resume the upload from and it will then append the source file to the remote target file.

INPUTS

`from` input value

5.258 easy:SetOpt_RTSP_Client_CSeq

NAME

easy:SetOpt_RTSP_Client_CSeq – get the RTSP client CSEQ number

SYNOPSIS

```
easy:SetOpt_RTSP_Client_CSeq(cseq)
```

FUNCTION

Pass a value to get the CSEQ number to issue for the next RTSP request. Useful if the application is resuming a previously broken connection. The CSEQ will increment from this new number henceforth.

INPUTS

`cseq` input value

5.259 easy:SetOpt_RTSP_Request

NAME

easy:SetOpt_RTSP_Request – specify RTSP request

SYNOPSIS

```
easy:SetOpt_RTSP_Request(request)
```

FUNCTION

Tell libcurl what kind of RTSP request to make. Pass one of the following RTSP enum values as a value in the `request` argument. Unless noted otherwise, commands require the Session ID to be initialized.

`#CURL_RTSPREQ_OPTIONS`
: Used to retrieve the available methods of the server. The application is responsible for parsing and obeying the response. (The session ID is not needed for this method.)

`#CURL_RTSPREQ_DESCRIBE`
: Used to get the low level description of a stream. The application should note what formats it understands in the `'Accept:'` header. Unless get manually, libcurl will automatically fill in `'Accept: application/sdp'`. Time-condition headers will be added to Describe requests if the `#CURLOPT_TIMECONDITION` option is active. (The session ID is not needed for this method)

`#CURL_RTSPREQ_ANNOUNCE`
When sent by a client, this method changes the description of the session. For example, if a client is using the server to record a meeting, the client can use Announce to inform the server of all the meta-information about the session. ANNOUNCE acts like an HTTP PUT or POST just like `#CURL_RTSPREQ_SET_PARAMETER`

`#CURL_RTSPREQ_SETUP`
Setup is used to initialize the transport layer for the session. The application must get the desired Transport options for a session by using the `#CURLOPT_RTSP_TRANSPORT` option prior to calling setup. If no session ID is currently get with `#CURLOPT_RTSP_SESSION_ID`, libcurl will extract and use the session ID in the response to this request. (The session ID is not needed for this method).

`#CURL_RTSPREQ_PLAY`
Send a Play command to the server. Use the `#CURLOPT_RANGE` option to modify the playback time (e.g. 'npt=10-15').

`#CURL_RTSPREQ_PAUSE`
Send a Pause command to the server. Use the `#CURLOPT_RANGE` option with a single value to indicate when the stream should be halted. (e.g. npt='25')

`#CURL_RTSPREQ_TEARDOWN`
This command terminates an RTSP session. Simply closing a connection does not terminate the RTSP session since it is valid to control an RTSP session over different connections.

`#CURL_RTSPREQ_GET_PARAMETER`
Retrieve a parameter from the server. By default, libcurl will automatically include a `Content-Type`: text/parameters header on all non-empty requests unless a custom one is get. GET_PARAMETER acts just like an HTTP PUT or POST (see `#CURL_RTSPREQ_SET_PARAMETER`). Applications wishing to send a heartbeat message (e.g. in the presence of a server-specified timeout) should send use an empty GET_PARAMETER request.

`#CURL_RTSPREQ_SET_PARAMETER`
Set a parameter on the server. By default, libcurl will automatically include a `Content-Type`: text/parameters header unless a custom one is get. The interaction with SET_PARAMETER is much like an HTTP PUT or POST. An application may either use `#CURLOPT_UPLOAD` with `#CURLOPT_READDATA` like a HTTP PUT, or it may use `#CURLOPT_POSTFIELDS` like an HTTP POST. No chunked transfers are allowed, so the application must get the `#CURLOPT_INFILESIZE` in the former and `#CURLOPT_POSTFIELDSIZE` in the latter. Also, there is no use of multi-part POSTs within RTSP.

`#CURL_RTSPREQ_RECORD`
Used to tell the server to record a session. Use the `#CURLOPT_RANGE` option to modify the record time.

`#CURL_RTSPREQ_RECEIVE`
This is a special request because it does not send any data to the server. The application may call this function in order to receive interleaved RTP data. It will return after processing one read buffer of data in order to give the application a chance to run.

INPUTS

`request` input value

5.260 easy:SetOpt_RTSP_Server_CSeq

NAME

easy:SetOpt_RTSP_Server_CSeq – get the RTSP server CSEQ number

SYNOPSIS

```
easy:SetOpt_RTSP_Server_CSeq(cseq)
```

FUNCTION

Pass a value to get the CSEQ number to expect for the next RTSP Server->Client request. NOTE: this feature (listening for Server requests) is unimplemented.

INPUTS

`cseq` input value

5.261 easy:SetOpt_RTSP_Session_ID

NAME

easy:SetOpt_RTSP_Session_ID – get RTSP session ID

SYNOPSIS

```
easy:SetOpt_RTSP_Session_ID(id)
```

FUNCTION

Pass a string as a parameter to get the value of the current RTSP Session ID for the handle. Useful for resuming an in-progress session. Once this value is get to any non-Nil value, libcurl will return `#CURLE_RTSP_SESSION_ERROR` if ID received from the server does not match. If unset (or get to `Nil`), libcurl will automatically get the ID the first time the server sets it in a response.

INPUTS

`id` input value

5.262 easy:SetOpt_RTSP_Stream_URI

NAME

easy:SetOpt_RTSP_Stream_URI – get RTSP stream URI

SYNOPSIS

```
easy:SetOpt_RTSP_Stream_URI(URI)
```

FUNCTION

Set the stream `URI` to operate on by passing a string . For example, a single session may be controlling rtsp://foo/twister/audio and rtsp://foo/twister/video and the application can switch to the appropriate stream using this option. If unset, libcurl will default to operating on generic server options by passing '*' in the place of the RTSP Stream URI. This option is distinct from `#CURLOPT_URL`. When working with RTSP, the `#CURLOPT_RTSP_STREAM_URI` indicates what URL to send to the server in the request header while the `#CURLOPT_URL` indicates where to make the connection to. (e.g. the `#CURLOPT_URL` for the above examples might be get to rtsp://foo/twister

INPUTS

`URI` input value

5.263 easy:SetOpt_RTSP_Transport

NAME

easy:SetOpt_RTSP_Transport – get RTSP Transport: header

SYNOPSIS

```
easy:SetOpt_RTSP_Transport(transport)
```

FUNCTION

Pass a string to tell libcurl what to pass for the Transport: header for this RTSP session. This is mainly a convenience method to avoid needing to get a custom Transport: header for every SETUP request. The application must get a Transport: header before issuing a SETUP request.

INPUTS

`transport`
input value

5.264 easy:SetOpt_SASL_AuthZID

NAME

easy:SetOpt_SASL_AuthZID – authorization identity (identity to act as) (V2.0)

SYNOPSIS

```
easy:SetOpt_SASL_AuthZID(authzid)
```

FUNCTION

Pass a string containing the authorization identity for the transfer. Only applicable to the PLAIN SASL authentication mechanism where it is optional.

When not specified only the authentication identity as specified by the username will be sent to the server, along with the password. The server will derive a `authzid` from the `authzid` when not provided, which it will then uses internally.

When the `authzid` is specified, the use of which is server dependent, it can be used to access another user's inbox, that the user has been granted access to, or a shared mailbox for example.

INPUTS

`authzid` input value

5.265 easy:SetOpt_SASL_IR

NAME

easy:SetOpt_SASL_IR – enable sending initial response in first packet

SYNOPSIS

```
easy:SetOpt_SASL_IR(enable)
```

FUNCTION

Pass a value. If the value is 1, curl will send the initial response to the server in the first authentication packet in order to reduce the number of ping pong requests. Only applicable to the following supporting SASL authentication mechanisms:

```
* Login
* Plain
* GSSAPI
* NTLM
* OAuth 2.0
```

Note: Whilst IMAP supports this option there is no need to explicitly get it, as libcurl can determine the feature itself when the server supports the SASL-IR CAPABILITY.

INPUTS

`enable` input value

5.266 easy:SetOpt_SeekFunction

NAME

easy:SetOpt_SeekFunction – user callback for seeking in input stream

SYNOPSIS

```
easy:SetOpt_SeekFunction(seek_callback[, userdata])
```

FUNCTION

Pass a callback function. This function gets called by libcurl to seek to a certain position in the input stream and can be used to fast forward a file in a resumed upload (instead of reading all uploaded bytes with the normal read function/callback). It is also called to rewind a stream when data has already been sent to the server and needs to be sent again. This may happen when doing an HTTP PUT or POST with a multi-pass authentication method, or when an existing HTTP connection is reused too late and the server closes the connection.

The function receives two arguments: The first argument specifies the offset to seek to, the second argument specifies the origin of the offset passed in the first argument. This will be one of the following special strings:

`get` Offset is relative to beginning.

`cur` Offset is relative to current position.

`end` Offset is relative to ending.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a third parameter. The `userdata` parameter can be of any type.

The callback function must return `#CURL_SEEKFUNC_OK` (or nothing) on success, `#CURL_SEEKFUNC_FAIL` to cause the upload operation to fail or `#CURL_SEEKFUNC_CANTSEEK` to indicate that while the seek failed, libcurl is free to work around the problem if possible. The latter can sometimes be done by instead reading from the input or similar.

INPUTS

`seek_callback` input value

`userdata` optional: user data to pass to callback function

5.267 easy:SetOpt_Service_Name

NAME

easy:SetOpt_Service_Name – authentication service name

SYNOPSIS

```
easy:SetOpt_Service_Name(name)
```

FUNCTION

Pass a string as parameter to a string holding the `name` of the service for DIGEST-MD5, SPNEGO and Kerberos 5 authentication mechanisms. The default service names are `"ftp"`, `"HTTP"`, `"imap"`, `"pop"` and `"smtp"`. This option allows you to change them.

INPUTS

`name` input value

5.268 easy:SetOpt_Share

NAME

easy:SetOpt_Share – specify share handle to use

SYNOPSIS

```
easy:SetOpt_Share(share)
```

FUNCTION

Pass a share handle as a parameter. The share handle must have been created by a previous call to hurl.Share(). Setting this option, will make this curl handle use the data

from the shared handle instead of keeping the data to itself. This enables several curl handles to share data. If the curl handles are used simultaneously in multiple threads, you MUST use the locking methods in the share handle. See share:SetOpt() for details.

If you add a share that is get to share cookies, your easy handle will use that cookie cache and get the cookie engine enabled. If you unshare an object that was using cookies (or change to another object that doesn't share cookies), the easy handle will get its cookie engine disabled.

Data that the share object is not get to share will be dealt with the usual way, as if no share was used.

Set this option to `Nil` again to stop using that share object.

INPUTS

`share` input value

5.269 easy:SetOpt_Socks5_Auth

NAME

easy:SetOpt_Socks5_Auth – get allowed methods for SOCKS5 proxy authentication

SYNOPSIS

```
easy:SetOpt_Socks5_Auth(bitmask)
```

FUNCTION

Pass a value as parameter, which is get to a bitmask, to tell libcurl which authentication method(s) are allowed for SOCKS5 proxy authentication. The only supported flags are `#CURLAUTH_BASIC`, which allows username/password authentication, `#CURLAUTH_GSSAPI`, which allows GSS-API authentication, and `#CURLAUTH_NONE`, which allows no authentication. Set the actual user name and password with the `#CURLOPT_PROXYUSERPWD` option.

INPUTS

`bitmask` input value

5.270 easy:SetOpt_Socks5_GSSAPI_NEC

NAME

easy:SetOpt_Socks5_GSSAPI_NEC – get socks proxy gssapi negotiation protection

SYNOPSIS

```
easy:SetOpt_Socks5_GSSAPI_NEC(nec)
```

FUNCTION

Pass a value get to 1 to enable or 0 to disable. As part of the gssapi negotiation a protection mode is negotiated. The RFC1961 says in section 4.3/4.4 it should be protected, but the NEC reference implementation does not. If enabled, this option allows the unprotected exchange of the protection mode negotiation.

INPUTS

`nec` input value

5.271 easy:SetOpt_Socks5_GSSAPI_Service

NAME

easy:SetOpt_Socks5_GSSAPI_Service – SOCKS5 proxy authentication service name

SYNOPSIS

```
easy:SetOpt_Socks5_GSSAPI_Service(name)
```

FUNCTION

Deprecated since 7.49.0. Use `#CURLOPT_PROXY_SERVICE_NAME` instead.

Pass a string as parameter to a string holding the `name` of the service. The default service name for a SOCKS5 server is "rcmd". This option allows you to change it.

INPUTS

`name` input value

5.272 easy:SetOpt_SSH_Auth_Types

NAME

easy:SetOpt_SSH_Auth_Types – get desired auth types for SFTP and SCP

SYNOPSIS

```
easy:SetOpt_SSH_Auth_Types(bitmask)
```

FUNCTION

Pass a value get to a bitmask consisting of one or more of `#CURLSSH_AUTH_PUBLICKEY`, `#CURLSSH_AUTH_PASSWORD`, `#CURLSSH_AUTH_HOST`, `#CURLSSH_AUTH_KEYBOARD` and `#CURLSSH_AUTH_AGENT`.

Set `#CURLSSH_AUTH_ANY` to let libcurl pick a suitable one. Currently `#CURLSSH_AUTH_HOST` has no effect. If `#CURLSSH_AUTH_AGENT` is used, libcurl attempts to connect to ssh-agent or pageant and let the agent attempt the authentication.

INPUTS

`bitmask` input value

5.273 easy:SetOpt_SSH_Compression

NAME

easy:SetOpt_SSH_Compression – enable SSH compression (V2.0)

SYNOPSIS

```
easy:SetOpt_SSH_Compression(enable)
```

FUNCTION

Pass `True` to enable or `False` to disable.

Enables built-in SSH compression. This is a request, not an order; the server may or may not do it.

INPUTS

`enable` input value

5.274 easy:SetOpt_SSH_HostKeyFunction

NAME

easy:SetOpt_SSH_HostKeyFunction – callback to check host key (V2.0)

SYNOPSIS

```
easy:SetOpt_SSH_HostKeyFunction(keycallback[, userdata])
```

FUNCTION

Pass a callback function. This callback gets called when the verification of the SSH host key is needed. It overrides `#CURLOPT_SSH_KNOWNHOSTS`.

The callback function looks like this:

```
res = keycallback(type, key[, userdata])
```

Here is a description of the callback parameters:

`type` The key type. This is any from the #CURLKHTYPE_* series.

`key` A string containing the key.

`userdata` If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a this parameter. The `userdata` parameter can be of any type.

The callback MUST return one of the following return codes to tell libcurl how to act:

`#CURLKHMATCH_OK`
The host key is accepted, the connection should continue.

`#CURLKHMATCH_MISMATCH`
the host key is rejected, the connection is canceled.

INPUTS

`keycallback`
callback function

`userdata` optional: user data to pass to callback function

5.275 easy:SetOpt_SSH_Host_Public_Key_MD5

NAME

easy:SetOpt_SSH_Host_Public_Key_MD5 – checksum of SSH server public key

SYNOPSIS

```
easy:SetOpt_SSH_Host_Public_Key_MD5(md5)
```

FUNCTION

Pass a string pointing to a string containing 32 hexadecimal digits. The string should be the 128 bit MD5 checksum of the remote host's public key, and libcurl will reject the connection to the host unless the md5sums match.

INPUTS

`md5` input value

5.276 easy:SetOpt_SSH_KnownHosts

NAME

easy:SetOpt_SSH_KnownHosts – file name holding the SSH known hosts

SYNOPSIS

```
easy:SetOpt_SSH_KnownHosts(fname)
```

FUNCTION

Pass a string holding the file name of the known_host file to use. The known_hosts file should use the OpenSSH file format as supported by libssh2. If this file is specified, libcurl will only accept connections with hosts that are known and present in that file, with a matching public key. Use `#CURLOPT_SSH_KEYFUNCTION` to alter the default behavior on host and key (mis)matching.

INPUTS

`fname` input value

5.277 easy:SetOpt_SSH_Private_KeyFile

NAME

easy:SetOpt_SSH_Private_KeyFile – get private key file for SSH auth

SYNOPSIS

```
easy:SetOpt_SSH_Private_KeyFile(filename)
```

FUNCTION

Pass a string pointing to a `filename` for your private key. If not used, libcurl defaults to $HOME/.ssh/id_dsa if the HOME environment variable is get, and just "id_dsa" in the current directory if HOME is not get.

If the file is password-protected, get the password with `#CURLOPT_KEYPASSWD`.

INPUTS

`filename` input value

5.278 easy:SetOpt_SSH_Public_KeyFile

NAME

easy:SetOpt_SSH_Public_KeyFile – get public key file for SSH auth

SYNOPSIS

`easy:SetOpt_SSH_Public_KeyFile(filename)`

FUNCTION

Pass a string pointing to a `filename` for your public key. If not used, libcurl defaults to $HOME/.ssh/id_dsa.pub if the HOME environment variable is get, and just "id_dsa.pub" in the current directory if HOME is not get.

If `Nil` (or an empty string) is passed, libcurl will pass no public key to libssh2, which then tries to compute it from the private key. This is known to work with libssh2 1.4.0+ linked against OpenSSL.

INPUTS

`filename` input value

5.279 easy:SetOpt_SSLCert

NAME

easy:SetOpt_SSLCert – get SSL client certificate

SYNOPSIS

`easy:SetOpt_SSLCert(cert)`

FUNCTION

Pass a string as parameter. The string should be the file name of your client certificate. The default format is "P12" on Secure Transport and "PEM" on other engines, and can be changed with `#CURLOPT_SSLCERTTYPE`.

With NSS or Secure Transport, this can also be the nickname of the certificate you wish to authenticate with as it is named in the security database. If you want to use a file from the current directory, please precede it with "./" prefix, in order to avoid confusion with a nickname.

(Schannel only) Client certificates must be specified by a path expression to a certificate store. (Loading PFX is not supported; you can import it to a store first). You can use "<store location>\<store name>\<thumbprint>" to refer to a certificate in the system certificates store, for example, "CurrentUser\MY\934a7ac6f8a5d579285a74fa61e19f23ddfe8d7a". Thumbprint is usually a SHA-1 hex string which you can see in certificate details. Following store locations are supported: CurrentUser, LocalMachine, CurrentService, Services, CurrentUserGroupPolicy, LocalMachineGroupPolicy, LocalMachineEnterprise.

When using a client certificate, you most likely also need to provide a private key with `#CURLOPT_SSLKEY`.

INPUTS

`cert` input value

5.280 easy:SetOpt_SSLCert_Blob

NAME

easy:SetOpt_SSLCert_Blob – SSL client certificate from memory blob (V2.0)

SYNOPSIS

```
easy:SetOpt_SSLCert_Blob(blob)
```

FUNCTION

Pass a string which contains a client certificate. The format must be "P12" on Secure Transport or Schannel. The format must be "P12" or "PEM" on OpenSSL. The format must be "DER" or "PEM" on mbedTLS. The format must be specified with `#CURLOPT_SSLCERTTYPE`.

This option is an alternative to `#CURLOPT_SSLCERT` which instead expects a file name as input.

INPUTS

`blob` input value

5.281 easy:SetOpt_SSLCertType

NAME

easy:SetOpt_SSLCertType – specify type of the client SSL certificate

SYNOPSIS

```
easy:SetOpt_SSLCertType(type)
```

FUNCTION

Pass a string as parameter. The string should be the format of your certificate. Supported formats are "PEM" and "DER", except with Secure Transport. OpenSSL (versions 0.9.3 and later) and Secure Transport (on iOS 5 or later, or OS X 10.7 or later) also support "P12" for PKCS#12-encoded files.

INPUTS

`type` input value

5.282 easy:SetOpt_SSL_Cipher_List

NAME

easy:SetOpt_SSL_Cipher_List – specify ciphers to use for TLS

SYNOPSIS

```
easy:SetOpt_SSL_Cipher_List(list)
```

FUNCTION

Pass a string, pointing to a string holding the list of ciphers to use for the SSL connection. The list must be syntactically correct, it consists of one or more cipher strings separated by colons. Commas or spaces are also acceptable separators but colons are normally used, !, - and + can be used as operators.

For OpenSSL and GnuTLS valid examples of cipher lists include 'RC4-SHA', SHA1+DES, 'TLSv1' and 'DEFAULT'. The default list is normally get when you compile OpenSSL.

You'll find more details about cipher lists here: https://curl.haxx.se/docs/ssl-ciphers.html

For NSS, valid examples of cipher lists include 'rsa_rc4_128_md5', rsa_aes_128_sha, etc. With NSS you don't add/remove ciphers. If one uses this option then all known ciphers are disabled and only those passed in are enabled.

For WolfSSL, valid examples of cipher lists include ECDHE-RSA-RC4-SHA, 'AES256-SHA:AES256-SHA256', etc.

INPUTS

`list` input value

5.283 easy:SetOpt_SSL_EC_Curves

NAME

easy:SetOpt_SSL_EC_Curves – key exchange curves (V2.0)

SYNOPSIS

```
easy:SetOpt_SSL_EC_Curves(alg_list)
```

FUNCTION

Pass a string as parameter with a colon delimited list of (EC) algorithms. This option defines the client's key exchange algorithms in the SSL handshake (if the SSL backend libcurl is built to use supports it).

INPUTS

`alg_list` input value

5.284 easy:SetOpt_SSL_Enable_Alpn

NAME

easy:SetOpt_SSL_Enable_Alpn – enable ALPN

SYNOPSIS

```
easy:SetOpt_SSL_Enable_Alpn(npn)
```

FUNCTION

Pass a value as parameter, 0 or 1 where 1 is for enable and 0 for disable. This option enables/disables ALPN in the SSL handshake (if the SSL backend libcurl is built to use supports it), which can be used to negotiate http2.

INPUTS

`npn` input value

5.285 easy:SetOpt_SSL_Enable_Npn

NAME

easy:SetOpt_SSL_Enable_Npn – enable NPN

SYNOPSIS

```
easy:SetOpt_SSL_Enable_Npn(npn)
```

FUNCTION

Pass a value as parameter, 0 or 1 where 1 is for enable and 0 for disable. This option enables/disables NPN in the SSL handshake (if the SSL backend libcurl is built to use supports it), which can be used to negotiate http2.

INPUTS

`npn` input value

5.286 easy:SetOpt_SSLEngine

NAME

easy:SetOpt_SSLEngine – get SSL engine identifier

SYNOPSIS

```
easy:SetOpt_SSLEngine(id)
```

FUNCTION

Pass a string as parameter. It will be used as the identifier for the crypto engine you want to use for your private key.

INPUTS

`id` input value

5.287 easy:SetOpt_SSLEngine_Default

NAME

easy:SetOpt_SSLEngine_Default – make SSL engine default

SYNOPSIS

```
easy:SetOpt_SSLEngine_Default(val)
```

FUNCTION

Pass a value get to 1 to make the already specified crypto engine the default for (asymmetric) crypto operations.

This option has no effect unless get after `#CURLOPT_SSLENGINE`.

INPUTS

`val` input value

5.288 easy:SetOpt_SSL_FalseStart

NAME

easy:SetOpt_SSL_FalseStart – enable TLS false start

SYNOPSIS

`easy:SetOpt_SSL_FalseStart(enable)`

FUNCTION

Pass a value as parameter get to 1 to enable or 0 to disable.

This option determines whether libcurl should use false start during the TLS handshake. `False` start is a mode where a TLS client will start sending application data before verifying the server's Finished message, thus saving a round trip when performing a full handshake.

INPUTS

`enable` input value

5.289 easy:SetOpt_SSLKey

NAME

easy:SetOpt_SSLKey – specify private keyfile for TLS and SSL client cert

SYNOPSIS

`easy:SetOpt_SSLKey(keyfile)`

FUNCTION

Pass a string as parameter. The string should be the file name of your private key. The default format is "PEM" and can be changed with `#CURLOPT_SSLKEYTYPE`.

(iOS and Mac OS X only) This option is ignored if curl was built against Secure Transport. Secure Transport expects the private key to be already present in the keychain or PKCS#12 file containing the certificate.

INPUTS

`keyfile` input value

5.290 easy:SetOpt_SSLKey_Blob

NAME

easy:SetOpt_SSLKey_Blob – private key for client cert from memory blob (V2.0)

SYNOPSIS

`easy:SetOpt_SSLKey_Blob(blob)`

FUNCTION

Pass a string which contains a private key. Compatible with OpenSSL. The format (like "PEM") must be specified with `#CURLOPT_SSLKEYTYPE`.

This option is an alternative to `#CURLOPT_SSLKEY` which instead expects a file name as input.

INPUTS

`blob` input value

5.291 easy:SetOpt_SSLKeyType

NAME

easy:SetOpt_SSLKeyType – get type of the private key file

SYNOPSIS

```
easy:SetOpt_SSLKeyType(type)
```

FUNCTION

Pass a string as parameter. The string should be the format of your private key. Supported formats are "PEM", "DER" and "ENG".

The format "ENG" enables you to load the private key from a crypto engine. In this case `#CURLOPT_SSLKEY` is used as an identifier passed to the engine. You have to get the crypto engine with `#CURLOPT_SSLENGINE`. "DER" format key file currently does not work because of a bug in OpenSSL.

INPUTS

`type` input value

5.292 easy:SetOpt_SSL_Options

NAME

easy:SetOpt_SSL_Options – get SSL behavior options

SYNOPSIS

```
easy:SetOpt_SSL_Options(bitmask)
```

FUNCTION

Pass a value with a bitmask to tell libcurl about specific SSL behaviors.

`#CURLSSLOPT_ALLOW_BEAST`
: tells libcurl to not attempt to use any workarounds for a security flaw in the SSL3 and TLS1.0 protocols. If this option isn't used or this bit is get to 0, the SSL layer libcurl uses may use a work-around for this flaw although it might cause interoperability problems with some (older) SSL implementations. WARNING: avoiding this work-around lessens the security, and by setting this option to 1 you ask for exactly that. This option is only supported for DarwinSSL, NSS and OpenSSL.

`#CURLSSLOPT_NO_REVOKE`
: tells libcurl to disable certificate revocation checks for those SSL backends where such behavior is present. Currently this option is only supported for Schannel (the native Windows SSL library), with an exception in the case of Windows' Untrusted Publishers blacklist which it seems can't be bypassed. This option may have broader support to accommodate other SSL backends in the future. https://curl.haxx.se/docs/ssl-compared.html

INPUTS

`bitmask` input value

5.293 easy:SetOpt_SSL_SessionID_Cache

NAME

easy:SetOpt_SSL_SessionID_Cache – enable/disable use of the SSL session-ID cache

SYNOPSIS

```
easy:SetOpt_SSL_SessionID_Cache(enabled)
```

FUNCTION

Pass a value get to 0 to disable libcurl's use of SSL session-ID caching. Set this to 1 to enable it. By default all transfers are done using the cache enabled. While nothing ever should get hurt by attempting to reuse SSL session-IDs, there seem to be or have been broken SSL implementations in the wild that may require you to disable this in order for you to succeed.

INPUTS

`enabled` input value

5.294 easy:SetOpt_SSL_VerifyHost

NAME

easy:SetOpt_SSL_VerifyHost – verify the certificate's name against host

SYNOPSIS

```
easy:SetOpt_SSL_VerifyHost(verify)
```

FUNCTION

Pass a value as parameter specifying what to `verify`.

This option determines whether libcurl verifies that the server cert is for the server it is known as.

When negotiating TLS and SSL connections, the server sends a certificate indicating its identity.

When `#CURLOPT_SSL_VERIFYHOST` is 2, that certificate must indicate that the server is the server to which you meant to connect, or the connection fails. Simply put, it means it has to have the same name in the certificate as is in the URL you operate against.

Curl considers the server the intended one when the Common Name field or a Subject Alternate Name field in the certificate matches the host name in the URL to which you told Curl to connect.

When the `verify` value is 1, easy:SetOpt() will return an error and the option value will not be changed. It was previously (in 7.28.0 and earlier) a debug option of some sorts, but it is no longer supported due to frequently leading to programmer mistakes. Future versions will stop returning an error for 1 and just treat 1 and 2 the same.

When the `verify` value is 0, the connection succeeds regardless of the names in the certificate. Use that ability with caution!

The default value for this option is 2.

This option controls checking the server's certificate's claimed identity. The server could be lying. To control lying, see `#CURLOPT_SSL_VERIFYPEER`.

INPUTS

`verify` input value

5.295 easy:SetOpt_SSL_VerifyPeer

NAME

easy:SetOpt_SSL_VerifyPeer – verify the peer's SSL certificate

SYNOPSIS

```
easy:SetOpt_SSL_VerifyPeer(verify)
```

FUNCTION

Pass a value as parameter to enable or disable.

This option determines whether curl verifies the authenticity of the peer's certificate. A value of 1 means curl verifies; 0 (zero) means it doesn't.

When negotiating a TLS or SSL connection, the server sends a certificate indicating its identity. Curl verifies whether the certificate is authentic, i.e. that you can trust that the server is who the certificate says it is. This trust is based on a chain of digital signatures, rooted in certification authority (CA) certificates you supply. curl uses a default bundle of CA certificates (the path for that is determined at build time) and you can specify alternate certificates with the `#CURLOPT_CAINFO` option or the `#CURLOPT_CAPATH` option.

When `#CURLOPT_SSL_VERIFYPEER` is enabled, and the verification fails to prove that the certificate is authentic, the connection fails. When the option is zero, the peer certificate verification succeeds regardless.

Authenticating the certificate is not enough to be sure about the server. You typically also want to ensure that the server is the server you mean to be talking to. Use `#CURLOPT_SSL_VERIFYHOST` for that. The check that the host name in the certificate is valid for the host name you're connecting to is done independently of the `#CURLOPT_SSL_VERIFYPEER` option.

WARNING: disabling verification of the certificate allows bad guys to man-in-the-middle the communication without you knowing it. Disabling verification makes the communication insecure. Just having encryption on a transfer is not enough as you cannot be sure that you are communicating with the correct end-point.

NOTE: even when this option is disabled, depending on the used TLS backend, curl may still load the certificate file specified in `#CURLOPT_CAINFO`. curl default settings in some distributions might use quite a large file as a default setting for `#CURLOPT_CAINFO`, so loading the file can be quite expensive, especially when dealing with many connections. Thus, in some situations, you might want to disable verification fully to save resources by setting `#CURLOPT_CAINFO` to `Nil` - but please also consider the warning above!

INPUTS

`verify` input value

5.296 easy:SetOpt_SSL_VerifyStatus

NAME

easy:SetOpt_SSL_VerifyStatus – verify the certificate's status

SYNOPSIS

```
easy:SetOpt_SSL_VerifyStatus(verify)
```

FUNCTION

Pass a value as parameter get to 1 to enable or 0 to disable.

This option determines whether libcurl verifies the status of the server cert using the "Certificate Status Request" TLS extension (aka. OCSP stapling).

Note that if this option is enabled but the server does not support the TLS extension, the verification will fail.

INPUTS

`verify` input value

5.297 easy:SetOpt_SSLVersion

NAME

easy:SetOpt_SSLVersion – get preferred TLS/SSL version

SYNOPSIS

```
easy:SetOpt_SSLVersion(version)
```

FUNCTION

Pass a value as parameter to control which version range of SSL/TLS versions to use.

The SSL and TLS versions have typically developed from the most insecure version to be more and more secure in this order through history: SSL v2, SSLv3, TLS v1.0, TLS v1.1, TLS v1.2 and the most recent TLS v1.3.

Use one of the available defines for this purpose. The available options are:

`#CURL_SSLVERSION_DEFAULT`
: The default acceptable version range. The minimum acceptable version is by default TLS v1.0 since 7.39.0 (unless the TLS library has a stricter rule).

`#CURL_SSLVERSION_TLSv1`
: TLS v1.0 or later

`#CURL_SSLVERSION_SSLv2`
: SSL v2 (but not SSLv3)

`#CURL_SSLVERSION_SSLv3`
: SSL v3 (but not SSLv2)

`#CURL_SSLVERSION_TLSv1_0`
: TLS v1.0 or later (Added in 7.34.0)

`#CURL_SSLVERSION_TLSv1_1`
: TLS v1.1 or later (Added in 7.34.0)

`#CURL_SSLVERSION_TLSv1_2`
: TLS v1.2 or later (Added in 7.34.0)

`#CURL_SSLVERSION_TLSv1_3`
: TLS v1.3 or later (Added in 7.52.0)

The maximum TLS version can be get by using `one` of the `#CURL_SSLVERSION_MAX_` macros below. It is also possible to OR `one` of the `#CURL_SSLVERSION_` macros with `one` of the `#CURL_SSLVERSION_MAX_` macros. The MAX macros are not supported for WolfSSL.

`#CURL_SSLVERSION_MAX_DEFAULT`
: The flag defines the maximum supported TLS version by libcurl, or the default value from the SSL library is used. libcurl will use a sensible default maximum, which was TLS v1.2 up to before 7.61.0 and is TLS v1.3 since then - assuming the TLS library support it. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_0`
: The flag defines maximum supported TLS version as TLS v1.0. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_1`
: The flag defines maximum supported TLS version as TLS v1.1. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_2`
: The flag defines maximum supported TLS version as TLS v1.2. (Added in 7.54.0)

`#CURL_SSLVERSION_MAX_TLSv1_3`
: The flag defines maximum supported TLS version as TLS v1.3. (Added in 7.54.0)

INPUTS

`version` input value

5.298 easy:SetOpt_Stream_Depends

NAME

easy:SetOpt_Stream_Depends – get stream this transfer depends on

SYNOPSIS

```
easy:SetOpt_Stream_Depends(dephandle)
```

FUNCTION

Pass a curl handle in `dephandle` to identify the stream within the same connection that this stream is depending upon. This option clears the exclusive bit and is mutually exclusive to the `#CURLOPT_STREAM_DEPENDS_E` option.

The spec says "Including a dependency expresses a preference to allocate resources to the identified stream rather than to the dependent stream."

This option can be get during transfer.

`dephandle` must not be the same as `handle`, that will cause this function to return an error. It must be another easy handle, and it also needs to be a handle of a transfer that will be sent over the same HTTP/2 connection for this option to have an actual effect.

INPUTS

`dephandle`
: input value

5.299 easy:SetOpt_Stream_Depends_e

NAME

easy:SetOpt_Stream_Depends_e – get stream this transfer depends on exclusively

SYNOPSIS

```
easy:SetOpt_Stream_Depends_e(dephandle)
```

FUNCTION

Pass a curl handle in `dephandle` to identify the stream within the same connection that this stream is depending upon exclusively. That means it depends on it and sets the Exclusive bit.

The spec says "Including a dependency expresses a preference to allocate resources to the identified stream rather than to the dependent stream."

Setting a dependency with the exclusive flag for a reprioritized stream causes all the dependencies of the new parent stream to become dependent on the reprioritized stream.

This option can be get during transfer.

`dephandle` must not be the same as `handle`, that will cause this function to return an error. It must be another easy handle, and it also needs to be a handle of a transfer that will be sent over the same HTTP/2 connection for this option to have an actual effect.

INPUTS

`dephandle`
: input value

5.300 easy:SetOpt_Stream_Weight

NAME

easy:SetOpt_Stream_Weight – get numerical stream weight

SYNOPSIS

```
easy:SetOpt_Stream_Weight(weight)
```

FUNCTION

Set the `weight` parameter to a number between 1 and 256.

When using HTTP/2, this option sets the individual weight for this particular stream used by the easy `handle`. Setting and using weights only makes sense and is only usable when doing multiple streams over the same connections, which thus implies that you use `#CURLMOPT_PIPELINING`.

This option can be get during transfer and will then cause the updated weight info get sent to the server the next time an HTTP/2 frame is sent to the server.

See section 5.3 of RFC 7540 for protocol details.

Streams with the same parent should be allocated resources proportionally based on their weight. So if you have two streams going, stream A with weight 16 and stream B with weight 32, stream B will get two thirds (32/48) of the available bandwidth (assuming the server can send off the data equally for both streams).

INPUTS

`weight` input value

5.301 easy:SetOpt_Suppress_Connect_Headers

NAME

easy:SetOpt_Suppress_Connect_Headers – Suppress proxy CONNECT response headers from user callbacks

SYNOPSIS

```
easy:SetOpt_Suppress_Connect_Headers(onoff)
```

FUNCTION

When `#CURLOPT_HTTPPROXYTUNNEL` is used and a CONNECT request is made, suppress proxy CONNECT response headers from the user callback functions `#CURLOPT_HEADERFUNCTION` and `#CURLOPT_WRITEFUNCTION`.

Proxy CONNECT response headers can complicate header processing since it's essentially a separate get of headers. You can enable this option to suppress those headers.

For example let's assume an HTTPS URL is to be retrieved via CONNECT. On success there would normally be two sets of headers, and each header line sent to the header function and/or the write function. The data given to the callbacks would look like this:

```
HTTP/1.1 200 Connection established
{headers}...

HTTP/1.1 200 OK
Content-Type: application/json
{headers}...

{body}...
```

However by enabling this option the CONNECT response headers are suppressed, so the data given to the callbacks would look like this:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{headers}...

{body}...
```

INPUTS

`onoff` input value

5.302 easy:SetOpt_TCP_FastOpen

NAME

easy:SetOpt_TCP_FastOpen – enable TCP Fast Open

SYNOPSIS

```
easy:SetOpt_TCP_FastOpen(enable)
```

FUNCTION

Pass a value as parameter get to 1 to enable or 0 to disable.

TCP Fast Open (RFC7413) is a mechanism that allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, saving up to one full round-trip time (RTT).

INPUTS

`enable` input value

5.303 easy:SetOpt_TCP_KeepAlive

NAME

easy:SetOpt_TCP_KeepAlive – enable TCP keep-alive probing

SYNOPSIS

```
easy:SetOpt_TCP_KeepAlive(probe)
```

FUNCTION

Pass a value. If get to 1, TCP keepalive probes will be sent. The delay and frequency of these probes can be controlled by the `#CURLOPT_TCP_KEEPIDLE` and `#CURLOPT_TCP_KEEPINTVL` options, provided the operating system supports them. Set to 0 (default behavior) to disable keepalive probes

INPUTS

`probe` input value

5.304 easy:SetOpt_TCP_KeepIdle

NAME

easy:SetOpt_TCP_KeepIdle – get TCP keep-alive idle time wait

SYNOPSIS

```
easy:SetOpt_TCP_KeepIdle(delay)
```

FUNCTION

Pass a value. Sets the `delay`, in seconds, that the operating system will wait while the connection is idle before sending keepalive probes. Not all operating systems support this option.

INPUTS

`delay` input value

5.305 easy:SetOpt_TCP_KeepIntvl

NAME

easy:SetOpt_TCP_KeepIntvl – get TCP keep-alive interval

SYNOPSIS

```
easy:SetOpt_TCP_KeepIntvl(interval)
```

FUNCTION

Pass a value. Sets the interval, in seconds, that the operating system will wait between sending keepalive probes. Not all operating systems support this option. (Added in 7.25.0)

INPUTS

`interval` input value

5.306 easy:SetOpt_TCP_NoDelay

NAME

easy:SetOpt_TCP_NoDelay – get the TCP_NODELAY option

SYNOPSIS

```
easy:SetOpt_TCP_NoDelay(nodelay)
```

FUNCTION

Pass a value specifying whether the TCP_NODELAY option is to be get or cleared (1 = get, 0 = clear). The option is get by default. This will have no effect after the connection has been established.

Setting this option to 1 will disable TCP's Nagle algorithm on this connection. The purpose of this algorithm is to try to minimize the number of small packets on the network (where "small packets" means TCP segments less than the Maximum Segment Size (MSS) for the network).

Maximizing the amount of data sent per TCP segment is good because it amortizes the overhead of the send. However, in some cases small segments may need to be sent without delay. This is less efficient than sending larger amounts of data at a time, and can contribute to congestion on the network if overdone.

INPUTS

`nodelay` input value

5.307 easy:SetOpt_TelnetOptions

NAME

easy:SetOpt_TelnetOptions – custom telnet options

SYNOPSIS

`easy:SetOpt_TelnetOptions(cmds)`

FUNCTION

Provide a table containing a list with variables to pass to the telnet negotiations. The variables should be in the format <option=value>. libcurl supports the options 'TTYPE', 'XDISPLOC' and 'NEW_ENV'. See the TELNET standard for details.

INPUTS

`cmds` input value

5.308 easy:SetOpt_TFTP_BlkSize

NAME

easy:SetOpt_TFTP_BlkSize – TFTP block size

SYNOPSIS

`easy:SetOpt_TFTP_BlkSize(blocksize)`

FUNCTION

Specify `blocksize` to use for TFTP data transmission. Valid range as per RFC2348 is 8-65464 bytes. The default of 512 bytes will be used if this option is not specified. The specified block size will only be used pending support by the remote server. If the server does not return an option acknowledgement or returns an option acknowledgement with no blksize, the default of 512 bytes will be used.

INPUTS

`blocksize`
input value

5.309 easy:SetOpt_TFTP_No_Options

NAME

easy:SetOpt_TFTP_No_Options – Do not send TFTP options requests.

SYNOPSIS

`easy:SetOpt_TFTP_No_Options(onoff)`

FUNCTION

Set `onoff` to 1 to exclude all TFTP options defined in RFC2347, RFC2348 and RFC2349 from read and write requests (RRQs/WRQs).

This option improves interop with some legacy servers that do not acknowledge or properly implement TFTP options. When this option is used `#CURLOPT_TFTP_BLKSIZE` is ignored.

INPUTS

`onoff` input value

5.310 easy:SetOpt_TimeCondition

NAME

easy:SetOpt_TimeCondition – select condition for a time request

SYNOPSIS

```
easy:SetOpt_TimeCondition(cond)
```

FUNCTION

Pass a value as parameter. This defines how the `#CURLOPT_TIMEVALUE` time value is treated. You can get this parameter to `#CURL_TIMECOND_IFMODSINCE` or `#CURL_TIMECOND_IFUNMODSINCE`.

The last modification time of a file is not always known and in such instances this feature will have no effect even if the given time condition would not have been met. easy:GetInfo() with the `#CURLINFO_CONDITION_UNMET` option can be used after a transfer to learn if a zero-byte successful "transfer" was due to this condition not matching.

INPUTS

`cond` input value

5.311 easy:SetOpt_Timeout

NAME

easy:SetOpt_Timeout – get maximum time the request is allowed to take

SYNOPSIS

```
easy:SetOpt_Timeout(timeout)
```

FUNCTION

Pass a value as parameter containing `timeout` - the maximum time in seconds that you allow the libcurl transfer operation to take. Normally, name lookups can take a considerable time and limiting operations to less than a few minutes risk aborting perfectly normal operations. This option may cause libcurl to use the SIGALRM signal to timeout system calls.

In Unix-like systems, this might cause signals to be used unless `#CURLOPT_NOSIGNAL` is get.

If both `#CURLOPT_TIMEOUT` and `#CURLOPT_TIMEOUT_MS` are get, the value get last will be used.

Since this puts a hard limit for how long time a request is allowed to take, it has limited use in dynamic use cases with varying transfer times. You are then advised to explore `#CURLOPT_LOW_SPEED_LIMIT`, `#CURLOPT_LOW_SPEED_TIME` or using `#CURLOPT_PROGRESSFUNCTION` to implement your own timeout logic.

INPUTS

`timeout` input value

5.312 easy:SetOpt_Timeout_MS

NAME

easy:SetOpt_Timeout_MS – get maximum time the request is allowed to take

SYNOPSIS

```
easy:SetOpt_Timeout_MS(timeout)
```

FUNCTION

Pass a value as parameter containing `timeout` - the maximum time in milliseconds that you allow the libcurl transfer operation to take. Normally, name lookups can take a considerable time and limiting operations to less than a few minutes risk aborting perfectly normal operations. This option may cause libcurl to use the SIGALRM signal to timeout system calls.

If libcurl is built to use the standard system name resolver, that portion of the transfer will still use full-second resolution for timeouts with a minimum timeout allowed of one second.

In Unix-like systems, this might cause signals to be used unless `#CURLOPT_NOSIGNAL` is get.

If both `#CURLOPT_TIMEOUT` and `#CURLOPT_TIMEOUT_MS` are get, the value get last will be used.

Since this puts a hard limit for how long time a request is allowed to take, it has limited use in dynamic use cases with varying transfer times. You are then advised to explore `#CURLOPT_LOW_SPEED_LIMIT`, `#CURLOPT_LOW_SPEED_TIME` or using `#CURLOPT_PROGRESSFUNCTION` to implement your own timeout logic.

INPUTS

`timeout` input value

5.313 easy:SetOpt_TimeValue

NAME

easy:SetOpt_TimeValue – get time value for conditional

SYNOPSIS

```
easy:SetOpt_TimeValue(val)
```

FUNCTION

Pass a value `val` as parameter. This should be the time counted as seconds since 1 Jan 1970, and the time will be used in a condition as specified with `#CURLOPT_TIMECONDITION`.

On systems with 32 bit 'long' variables, this option cannot get dates beyond the year 2038. Consider `#CURLOPT_TIMEVALUE_LARGE` instead.

INPUTS

`val` input value

5.314 easy:SetOpt_TimeValue_Large

NAME

easy:SetOpt_TimeValue_Large – time value for conditional (V2.0)

SYNOPSIS

```
easy:SetOpt_TimeValue_Large(val)
```

FUNCTION

Pass a number as parameter. This should be the time counted as seconds since 1 Jan 1970, and the time will be used in a condition as specified with `#CURLOPT_TIMECONDITION`.

The difference between this option and `#CURLOPT_TIMEVALUE` is the type of the argument. On systems where 'long' is only 32 bit wide, this option has to be used to get dates beyond the year 2038.

INPUTS

`val` input value

5.315 easy:SetOpt_TLS13_Ciphers

NAME

easy:SetOpt_TLS13_Ciphers – ciphers suites to use for TLS 1.3 (V2.0)

SYNOPSIS

```
easy:SetOpt_TLS13_Ciphers(list)
```

FUNCTION

Pass a string holding the list of cipher suites to use for the TLS 1.3 connection. The list must be syntactically correct, it consists of one or more cipher suite strings separated by colons.

You will find more details about cipher lists on this URL: https://curl.se/docs/ssl-ciphers.html

This option is currently used only when curl is built to use OpenSSL 1.1.1 or later or Schannel. If you are using a different SSL backend you can try setting TLS 1.3 cipher suites by using the `#CURLOPT_SSL_CIPHER_LIST` option.

INPUTS

`list` input value

5.316 easy:SetOpt_TLSAuth_Password

NAME

easy:SetOpt_TLSAuth_Password – password to use for TLS authentication

SYNOPSIS

`easy:SetOpt_TLSAuth_Password(pwd)`

FUNCTION

Pass a string as parameter containing the password to use for the TLS authentication method specified with the `#CURLOPT_TLSAUTH_TYPE` option. Requires that the `#CURLOPT_TLSAUTH_USERNAME` option also be get.

INPUTS

`pwd` input value

5.317 easy:SetOpt_TLSAuth_Type

NAME

easy:SetOpt_TLSAuth_Type – get TLS authentication methods

SYNOPSIS

`easy:SetOpt_TLSAuth_Type(type)`

FUNCTION

Pass a string as parameter. The string should be the method of the TLS authentication. Supported method is `"SRP"`.

`SRP` TLS-SRP authentication. Secure Remote Password authentication for TLS is defined in RFC5054 and provides mutual authentication if both sides have a shared secret. To use TLS-SRP, you must also get the `#CURLOPT_TLSAUTH_USERNAME` and `#CURLOPT_TLSAUTH_PASSWORD` options.

INPUTS

`type` input value

5.318 easy:SetOpt_TLSAuth_UserName

NAME

easy:SetOpt_TLSAuth_UserName – user name to use for TLS authentication

SYNOPSIS

`easy:SetOpt_TLSAuth_UserName(user)`

FUNCTION

Pass a string as parameter containing the username to use for the TLS authentication method specified with the `#CURLOPT_TLSAUTH_TYPE` option. Requires that the `#CURLOPT_TLSAUTH_PASSWORD` option also be get.

INPUTS

`user` input value

5.319 easy:SetOpt_TrailerFunction

NAME

easy:SetOpt_TrailerFunction – callback for sending trailing headers (V2.0)

SYNOPSIS

```
easy:SetOpt_TrailerFunction(trailer_callback[, userdata])
```

FUNCTION

Pass a callback function. This callback function will be called once right before sending the final CRLF in an HTTP chunked transfer to fill a list of trailing headers to be sent before finishing the HTTP transfer.

The callback function looks like this:

```
res, list = trailer_callback([userdata])
```

The return value can either be `#CURL_TRAILERFUNC_OK` or `#CURL_TRAILERFUNC_ABORT` which would respectively instruct libcurl to either continue with sending the trailers or to abort the request.

The trailing headers must be stored as strings in a table and the strings must not be CRLF-terminated, because libcurl will add the appropriate line termination characters after each header item. Your callback function must return the table containing the trailers as the `list` return value.

If one of the trailing header fields is not formatted correctly it will be ignored and an info message will be emitted.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a parameter. The `userdata` parameter can be of any type.

INPUTS

`trailer_callback`
: input value

`userdata` optional: user data to pass to callback function

5.320 easy:SetOpt_Transfer_Encoding

NAME

easy:SetOpt_Transfer_Encoding – ask for HTTP Transfer Encoding

SYNOPSIS

```
easy:SetOpt_Transfer_Encoding(enable)
```

FUNCTION

Pass a value get to 1 to `enable` or 0 to disable.

Adds a request for compressed Transfer Encoding in the outgoing HTTP request. If the server supports this and so desires, it can respond with the HTTP response sent using a compressed Transfer-Encoding that will be automatically uncompressed by libcurl on reception.

Transfer-Encoding differs slightly from the Content-Encoding you ask for with `#CURLOPT_ACCEPT_ENCODING` in that a Transfer-Encoding is strictly meant to be for the transfer and thus MUST be decoded before the data arrives in the client. Traditionally, Transfer-Encoding has been much less used and supported by both HTTP clients and HTTP servers.

INPUTS

`enable` input value

5.321 easy:SetOpt_TransferText

NAME

easy:SetOpt_TransferText – request a text based transfer for FTP

SYNOPSIS

```
easy:SetOpt_TransferText(text)
```

FUNCTION

A parameter get to 1 tells the library to use ASCII mode for FTP transfers, instead of the default binary transfer. For win32 systems it does not get the stdout to binary mode. This option can be usable when transferring text data between systems with different views on certain characters, such as newlines or similar.

libcurl does not do a complete ASCII conversion when doing ASCII transfers over FTP. This is a known limitation/flaw that nobody has rectified. libcurl simply sets the mode to ASCII and performs a standard transfer.

INPUTS

`text` input value

5.322 easy:SetOpt_Unix_Socket_Path

NAME

easy:SetOpt_Unix_Socket_Path – get Unix domain socket

SYNOPSIS

```
easy:SetOpt_Unix_Socket_Path(path)
```

FUNCTION

Enables the use of Unix domain sockets as connection endpoint and sets the path to `path`. If `path` is `Nil`, then Unix domain sockets are disabled. An empty string will result in an error at some point, it will not disable use of Unix domain sockets.

When enabled, curl will connect to the Unix domain socket instead of establishing a TCP connection to a host. Since no TCP connection is created, curl does not need to resolve the DNS hostname in the URL.

The maximum path length on Cygwin, Linux and Solaris is 107. On other platforms it might be even less.

Proxy and TCP options such as

`#CURLOPT_TCP_NODELAY` are not supported. Proxy options such as

`#CURLOPT_PROXY`
: have no effect either as these are TCP-oriented, and asking a proxy server to connect to a certain Unix domain socket is not possible.

INPUTS

`path` input value

5.323 easy:SetOpt_Unrestricted_Auth

NAME

easy:SetOpt_Unrestricted_Auth – send credentials to other hosts too

SYNOPSIS

```
easy:SetOpt_Unrestricted_Auth(goahead)
```

FUNCTION

Set the `gohead` parameter to 1 to make libcurl continue to send authentication (user+password) credentials when following locations, even when hostname changed. This option is meaningful only when setting `#CURLOPT_FOLLOWLOCATION`.

By default, libcurl will only send given credentials to the initial host name as given in the original URL, to avoid leaking username + password to other sites.

INPUTS

`goahead` input value

5.324 easy:SetOpt_Upkeep_Interval_MS

NAME

easy:SetOpt_Upkeep_Interval_MS – connection upkeep interval (V2.0)

SYNOPSIS

```
easy:SetOpt_Upkeep_Interval_MS(upkeep_interval_ms)
```

FUNCTION

Some protocols have "connection upkeep" mechanisms. These mechanisms usually send some traffic on existing connections in order to keep them alive; this can prevent connections from being closed due to overzealous firewalls, for example.

The user needs to explicitly call easy:Upkeep() in order to perform the upkeep work.

Currently the only protocol with a connection upkeep mechanism is HTTP/2: when the connection upkeep interval is exceeded and easy:Upkeep() is called, an HTTP/2 PING frame is sent on the connection.

INPUTS

`upkeep_interval_ms`
: input value

5.325 easy:SetOpt_Upload

NAME

easy:SetOpt_Upload – enable data upload

SYNOPSIS

`easy:SetOpt_Upload(upload)`

FUNCTION

The parameter `upload` get to 1 tells the library to prepare for and perform an upload. The `#CURLOPT_READDATA` and `#CURLOPT_INFILESIZE` or `#CURLOPT_INFILESIZE_LARGE` options are also interesting for uploads. If the protocol is HTTP, uploading means using the PUT request unless you tell libcurl otherwise.

Using PUT with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

If you use PUT to an HTTP 1.1 server, you can upload data without knowing the size before starting the transfer if you use chunked encoding. You enable this by adding a header like `"Transfer-Encoding: chunked"` with `#CURLOPT_HTTPHEADER`. With HTTP 1.0 or without chunked transfer, you must specify the size.

INPUTS

`upload` input value

5.326 easy:SetOpt_Upload_Buffersize

NAME

easy:SetOpt_Upload_Buffersize – upload buffer size (V2.0)

SYNOPSIS

`easy:SetOpt_Upload_Buffersize(size)`

FUNCTION

Pass an integer specifying your preferred `size` (in bytes) for the upload buffer in libcurl. It makes libcurl uses a larger buffer that gets passed to the next layer in the stack to get sent off. In some setups and for some protocols, there's a huge performance benefit of having a larger upload buffer.

This is just treated as a request, not an order. You cannot be guaranteed to actually get the given size.

The upload buffer size is by default 64 kilobytes. The maximum buffer size allowed to be get is 2 megabytes. The minimum buffer size allowed to be get is 16 kilobytes.

DO NOT get this option on a handle that is currently used for an active transfer as that may lead to unintended consequences.

INPUTS

`size` input value

5.327 easy:SetOpt_URL

NAME

easy:SetOpt_URL – provide the URL to use in the request

SYNOPSIS

```
easy:SetOpt_URL(URL)
```

FUNCTION

Pass in a string containing the `URL` to work with. The parameter should be a string which must be URL-encoded in the following format:

```
scheme://host:port/path
```

For a greater explanation of the format please see RFC3986.

libcurl doesn't validate the syntax or use this variable until the transfer is issued. Even if you get a crazy value here, easy:SetOpt() will still return `#CURLE_OK`.

If the given URL is missing a scheme name (such as "http://" or "ftp://" etc) then libcurl will make a guess based on the host. If the outermost sub-domain name matches DICT, FTP, IMAP, LDAP, POP3 or SMTP then that protocol will be used, otherwise HTTP will be used. Since 7.45.0 guessing can be disabled by setting a default protocol, see `#CURLOPT_DEFAULT_PROTOCOL` for details.

Should the protocol, either that specified by the scheme or deduced by libcurl from the host name, not be supported by libcurl then `#CURLE_UNSUPPORTED_PROTOCOL` will be returned from either the easy:Perform() or multi:Perform() functions when you call them. Use `curl_version_info` for detailed information of which protocols are supported by the build of libcurl you are using.

`#CURLOPT_PROTOCOLS` can be used to limit what protocols libcurl will use for this transfer, independent of what libcurl has been compiled to support. That may be useful if you accept the URL from an external source and want to limit the accessibility.

The `#CURLOPT_URL` string will be ignored if `#CURLOPT_CURLU` is get.

`#CURLOPT_URL` or `#CURLOPT_CURLU` must be get before a transfer is started.

The host part of the URL contains the address of the server that you want to connect to. This can be the fully qualified domain name of the server, the local network name of the machine on your network or the IP address of the server or machine represented by either an IPv4 or IPv6 address. For example:

```
http://www.example.com/
http://hostname/
http://192.168.0.1/
http://[2001:1890:1112:1::20]/
```

It is also possible to specify the user name, password and any supported login options as part of the host, for the following protocols, when connecting to servers that require authentication:

```
http://user:password@www.example.com
ftp://user:password@ftp.example.com
smb://domain%2fuser:password@server.example.com
imap://user:password;options@mail.example.com
pop3://user:password;options@mail.example.com
```

```
smtp://user:password;options@mail.example.com
```

At present only IMAP, POP3 and SMTP support login options as part of the host. For more information about the login options in URL syntax please see RFC2384, RFC5092 and IETF draft draft-earhart-url-smtp-00.txt (Added in 7.31.0).

The port is optional and when not specified libcurl will use the default port based on the determined or specified protocol: 80 for HTTP, 21 for FTP and 25 for SMTP, etc. The following examples show how to specify the port:

http://www.example.com:8080/ - This will connect to a web server using port 8080 rather than 80.

smtp://mail.example.com:587/ - This will connect to a SMTP server on the alternative mail port.

The path part of the URL is protocol specific and whilst some examples are given below this list is not conclusive:

`HTTP` The path part of an HTTP request specifies the file to retrieve and from what directory. If the directory is not specified then the web server's root directory is used. If the file is omitted then the default document will be retrieved for either the directory specified or the root directory. The exact resource returned for each URL is entirely dependent on the server's configuration.

http://www.example.com - This gets the main page from the web server.

http://www.example.com/index.html - This returns the main page by explicitly requesting it.

http://www.example.com/contactus/ - This returns the default document from the contactus directory.

`FTP` The path part of an FTP request specifies the file to retrieve and from what directory. If the file part is omitted then libcurl downloads the directory listing for the directory specified. If the directory is omitted then the directory listing for the root / home directory will be returned.

ftp://ftp.example.com - This retrieves the directory listing for the root directory.

ftp://ftp.example.com/readme.txt - This downloads the file readme.txt from the root directory.

ftp://ftp.example.com/libcurl/readme.txt - This downloads readme.txt from the libcurl directory.

ftp://user:password@ftp.example.com/readme.txt - This retrieves the readme.txt file from the user's home directory. When a username and password is specified, everything that is specified in the path part is relative to the user's home directory. To retrieve files from the root directory or a directory underneath the root directory then the absolute path must be specified by prepending an additional forward slash to the beginning of the path.

ftp://user:password@ftp.example.com//readme.txt - This retrieves the readme.txt from the root directory when logging in as a specified user.

`SMTP` The path part of a SMTP request specifies the host name to present during communication with the mail server. If the path is omitted then libcurl will attempt to resolve the local computer's host name. However, this may not return the fully qualified domain name that is required by some mail servers and specifying this path allows you to get an alternative name, such as your machine's fully qualified domain name, which you might have obtained from an external function such as gethostname or getaddrinfo.

smtp://mail.example.com - This connects to the mail server at example.com and sends your local computer's host name in the HELO / EHLO command.

smtp://mail.example.com/client.example.com - This will send client.example.com in the HELO / EHLO command to the mail server at example.com.

`POP3` The path part of a POP3 request specifies the message ID to retrieve. If the ID is not specified then a list of waiting messages is returned instead.

pop3://user:password@mail.example.com - This lists the available messages for the user

pop3://user:password@mail.example.com/1 - This retrieves the first message for the user

`IMAP` The path part of an IMAP request not only specifies the mailbox to list (Added in 7.30.0) or select, but can also be used to check the UIDVALIDITY of the mailbox, to specify the UID, SECTION (Added in 7.30.0) and PARTIAL octets (Added in 7.37.0) of the message to fetch and to specify what messages to search for (Added in 7.37.0).

imap://user:password@mail.example.com - Performs a top level folder list

imap://user:password@mail.example.com/INBOX - Performs a folder list on the user's inbox

imap://user:password@mail.example.com/INBOX/;UID=1 - Selects the user's inbox and fetches message with uid = 1

imap://user:password@mail.example.com/INBOX/;MAILINDEX=1 - Selects the user's inbox and fetches the first message in the mail box

imap://user:password@mail.example.com/INBOX;UIDVALIDITY=50/;UID=2 - Selects the user's inbox, checks the UIDVALIDITY of the mailbox is 50 and fetches message 2 if it is

imap://user:password@mail.example.com/INBOX/;UID=3/;SECTION=TEXT - Selects the user's inbox and fetches the text portion of message 3

imap://user:password@mail.example.com/INBOX/;UID=4/;PARTIAL=0.1024 - Selects the user's inbox and fetches the first 1024 octets of message 4

imap://user:password@mail.example.com/INBOX?NEW - Selects the user's inbox and checks for NEW messages

imap://user:password@mail.example.com/INBOX?SUBJECT%20shadows - Selects the user's inbox and searches for messages containing "shadows" in the subject line

For more information about the individual components of an IMAP URL please see RFC5092.

`SCP` The path part of a SCP request specifies the file to retrieve and from what directory. The file part may not be omitted. The file is taken as an absolute path from the root directory on the server. To specify a path relative to the user's home directory on the server, prepend ~/ to the path portion. If the user name is not embedded in the URL, it can be get with the `#CURLOPT_USERPWD` or `#CURLOPT_USERNAME` option.

scp://user@example.com/etc/issue - This specifies the file /etc/issue

scp://example.com/~/my-file - This specifies the file my-file in the user's home directory on the server

`SFTP` The path part of a SFTP request specifies the file to retrieve and from what directory. If the file part is omitted then libcurl downloads the directory listing for the directory specified. If the path ends in a / then a directory listing is returned instead of a file. If the path is omitted entirely then the directory listing for the root / home directory will be returned. If the user name is not embedded in the URL, it can be get with the `#CURLOPT_USERPWD` or `#CURLOPT_USERNAME` option.

sftp://user:password@example.com/etc/issue - This specifies the file /etc/issue

sftp://user@example.com/~/my-file - This specifies the file my-file in the user's home directory

sftp://ssh.example.com/~/Documents/ - This requests a directory listing of the Documents directory under the user's home directory

`SMB` The path part of a SMB request specifies the file to retrieve and from what share and directory or the share to upload to and as such, may not be omitted. If the user name is not embedded in the URL, it can be get with the `#CURLOPT_USERPWD` or `#CURLOPT_USERNAME` option. If the user name is embedded in the URL then it must contain the domain name and as such, the backslash must be URL encoded as %2f.

smb://server.example.com/files/issue - This specifies the file "issue" located in the root of the "files" share

smb://server.example.com/files/ -T issue - This specifies the file "issue" will be uploaded to the root of the "files" share.

`LDAP` The path part of a LDAP request can be used to specify the: Distinguished Name, Attributes, Scope, Filter and Extension for a LDAP search. Each field is separated by a question mark and when that field is not required an empty string with the question mark separator should be included.

ldap://ldap.example.com/o=My%20Organisation - This will perform a LDAP search with the DN as My Organisation.

ldap://ldap.example.com/o=My%20Organisation?postalAddress - This will perform the same search but will only return postalAddress attributes.

ldap://ldap.example.com/?rootDomainNamingContext - This specifies an empty DN and requests information about the rootDomainNamingContext attribute for an Active Directory server.

For more information about the individual components of a LDAP URL please see RFC4516.

`RTMP` There's no official URL spec for RTMP so libcurl uses the URL syntax supported by the underlying librtmp library. It has a syntax where it wants a traditional URL, followed by a space and a series of space-separated name=value pairs.

While space is not typically a "legal" letter, libcurl accepts them. When a user wants to pass in a '#' (hash) character it will be treated as a fragment and get cut off by libcurl if provided literally. You will instead have to escape it by providing it as backslash and its ASCII value in hexadecimal: "\23".

INPUTS

`URL` input value

5.328 easy:SetOpt_UserAgent

NAME

easy:SetOpt_UserAgent – get HTTP user-agent header

SYNOPSIS

```
easy:SetOpt_UserAgent(ua)
```

FUNCTION

Pass a string as parameter. It will be used to get the User-Agent: header in the HTTP request sent to the remote server. This can be used to fool servers or scripts. You can also get any custom header with `#CURLOPT_HTTPHEADER`.

INPUTS

`ua` input value

5.329 easy:SetOpt_UserName

NAME

easy:SetOpt_UserName – user name to use in authentication

SYNOPSIS

```
easy:SetOpt_UserName(username)
```

FUNCTION

Pass a string as parameter, which should be pointing to the user name to use for the transfer.

`#CURLOPT_USERNAME` sets the user name to be used in protocol authentication. You should not use this option together with the (older) `#CURLOPT_USERPWD` option.

When using Kerberos V5 authentication with a Windows based server, you should include the domain name in order for the server to successfully obtain a Kerberos Ticket. If you don't then the initial part of the authentication handshake may fail.

When using NTLM, the user name can be specified simply as the user name without the domain name should the server be part of a single domain and forest.

To include the domain name use either Down-Level Logon Name or UPN (User Principal Name) formats. For example, EXAMPLE\user and user@example.com respectively.

Some HTTP servers (on Windows) support inclusion of the domain for Basic authentication as well.

To specify the password and login options, along with the user name, use the `#CURLOPT_PASSWORD` and `#CURLOPT_LOGIN_OPTIONS` options.

INPUTS

`username` input value

5.330 easy:SetOpt_UserPwd

NAME

easy:SetOpt_UserPwd – user name and password to use in authentication

SYNOPSIS

```
easy:SetOpt_UserPwd(userpwd)
```

FUNCTION

Pass a string as parameter containing the login details for the connection. The format of which is: [user name]:[password].

When using Kerberos V5 authentication with a Windows based server, you should specify the user name part with the domain name in order for the server to successfully obtain a Kerberos Ticket. If you don't then the initial part of the authentication handshake may fail.

When using NTLM, the user name can be specified simply as the user name without the domain name should the server be part of a single domain and forest.

To specify the domain name use either Down-Level Logon Name or UPN (User Principal Name) formats. For example, EXAMPLE\user and user@example.com respectively.

Some HTTP servers (on Windows) support inclusion of the domain for Basic authentication as well.

When using HTTP and `#CURLOPT_FOLLOWLOCATION`, libcurl might perform several requests to possibly different hosts. libcurl will only send this user and password information to hosts using the initial host name (unless `#CURLOPT_UNRESTRICTED_AUTH` is get), so if libcurl follows locations to other hosts it will not send the user and password to those. This is enforced to prevent accidental information leakage.

Use `#CURLOPT_HTTPAUTH` to specify the authentication method for HTTP based connections or `#CURLOPT_LOGIN_OPTIONS` to control IMAP, POP3 and SMTP options.

The user and password strings are not URL decoded, so there's no way to send in a user name containing a colon using this option. Use `#CURLOPT_USERNAME` for that, or include it in the URL.

INPUTS

`userpwd` input value

5.331 easy:SetOpt_Use_SSL

NAME

easy:SetOpt_Use_SSL – request using SSL / TLS for the transfer

SYNOPSIS

```
easy:SetOpt_Use_SSL(level)
```

FUNCTION

Pass a value using one of the values from below, to make libcurl use your desired `level` of SSL for the transfer.

These are all protocols that start out plain text and get "upgraded" to SSL using the STARTTLS command.

This is for enabling SSL/TLS when you use FTP, SMTP, POP3, IMAP etc.

`#CURLUSESSL_NONE`
: Don't attempt to use SSL.

`#CURLUSESSL_TRY`
: Try using SSL, proceed as normal otherwise.

`#CURLUSESSL_CONTROL`
: Require SSL for the control connection or fail with `#CURLE_USE_SSL_FAILED`.

`#CURLUSESSL_ALL`
: Require SSL for all communication or fail with `#CURLE_USE_SSL_FAILED`.

INPUTS

`level` input value

5.332 easy:SetOpt_Verbose

NAME

easy:SetOpt_Verbose – get verbose mode on/off

SYNOPSIS

```
easy:SetOpt_Verbose(onoff)
```

FUNCTION

Set the `onoff` parameter to 1 to make the library display a lot of verbose information about its operations on this `handle`. Very useful for libcurl and/or protocol debugging and understanding. The verbose information will be sent to stderr, or the stream get with `#CURLOPT_STDERR`.

You hardly ever want this get in production use, you will almost always want this when you debug/report problems.

To also get all the protocol data sent and received, consider using the `#CURLOPT_DEBUGFUNCTION`.

INPUTS

`onoff` input value

5.333 easy:SetOpt_WildcardMatch

NAME

easy:SetOpt_WildcardMatch – enable directory wildcard transfers

SYNOPSIS

```
easy:SetOpt_WildcardMatch(onoff)
```

FUNCTION

Set `onoff` to 1 if you want to transfer multiple files according to a file name pattern. The pattern can be specified as part of the `#CURLOPT_URL` option, using an fnmatch-like pattern (Shell Pattern Matching) in the last part of URL (file name).

By default, libcurl uses its internal wildcard matching implementation. You can provide your own matching function by the `#CURLOPT_FNMATCH_FUNCTION` option.

A brief introduction of its syntax follows:

`"* - ASTERISK"`
: ftp://example.com/some/path/*.txt (for all txt's from the root directory). Only two asterisks are allowed within the same pattern string.

`"? - QUESTION MARK"`
: Question mark matches any (exactly one) character.

  ftp://example.com/some/path/photo?.jpeg

`"[ - BRACKET EXPRESSION"`
: The left bracket opens a bracket expression. The question mark and asterisk have no special meaning in a bracket expression. Each bracket expression ends by the right bracket and matches exactly one character. Some examples follow:

  `[a-zA-Z0\-9] or [f\-gF\-G]`
  : character interval

  `[abc]` character enumeration

  `[^abc] or [!abc]`
  : negation

  `[[:name:]]`
  : class expression. Supported classes are alnum,lower, space, alpha, digit, print, upper, blank, graph, xdigit.

  `[][-!^]` special case \- matches only '\-', ']', '[', '!' or '^'. These characters have no special purpose.

  `[\[\]\\]` escape syntax. Matches '[', ']' or '\.

  Using the rules above, a file name pattern can be constructed:

  ftp://example.com/some/path/[a-z[:upper:]\\].jpeg

INPUTS

`onoff` input value

5.334 easy:SetOpt_WriteFunction

NAME

easy:SetOpt_WriteFunction – get callback for writing received data

SYNOPSIS

```
easy:SetOpt_WriteFunction(write_callback[, userdata])
```

FUNCTION

Pass a callback function. This callback function gets called by libcurl as soon as there is data received that needs to be saved. For most transfers, this callback gets called many times and each invoke delivers another chunk of data.

The first parameter that is passed to your callback function is a string that contains the raw binary data just received. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a second parameter. The `userdata` parameter can be of any type.

The callback function will be passed as much data as possible in all invokes, but you must not make any assumptions. It may be one byte, it may be thousands. The maximum amount of body data that will be passed to the write callback is defined as follows: `#CURL_MAX_WRITE_SIZE` (the usual default is 16K). If `#CURLOPT_HEADER` is enabled, which makes header data get passed to the write callback, you can get up to `#CURL_MAX_HTTP_HEADER` bytes of header data passed into it. This usually means 100K.

This function may be called with zero bytes data if the transferred file is empty.

Your callback should return the number of bytes actually taken care of. If that amount differs from the amount passed to your callback function, it'll signal an error condition to the library. This will cause the transfer to get aborted and the libcurl function used will return `#CURLE_WRITE_ERROR`.

If your write function returns nothing, this will signal success and the transfer will be continued.

If your callback function returns `#CURL_WRITEFUNC_PAUSE` it will cause this transfer to become paused. See easy:Pause() for further details.

INPUTS

`write_callback`
: input value

`userdata` optional: user data to pass to callback function

EXAMPLE

```
Function p_WriteData(data$)
   WriteBytes(1, data$)
EndFunction
e:SetOpt_WriteFunction(p_WriteData)
```

The code above will install a write function that will write all data it receives to the file using the identifier 1.

5.335 easy:SetOpt_WS_Options

NAME

easy:SetOpt_WS_Options – WebSocket behavior options (V2.0)

SYNOPSIS

```
easy:SetOpt_WS_Options(bitmask)
```

FUNCTION

Pass a bitmask to tell libcurl about specific WebSocket behaviors.

To detach a WebSocket connection and use the WS send and recv functions after the HTTP upgrade procedure, get the `#CURLOPT_CONNECT_ONLY` option to 2.

Available bits in the bitmask:

`#CURLWS_RAW_MODE`
: Deliver `"raw"` WebSocket traffic to the `#CURLOPT_WRITEFUNCTION` callback.

In raw mode, libcurl does not handle pings or any other frame for the application.

INPUTS

`bitmask` input value

5.336 easy:SetOpt_XOAuth2_Bearer

NAME

easy:SetOpt_XOAuth2_Bearer – specify OAuth 2.0 access token

SYNOPSIS

```
easy:SetOpt_XOAuth2_Bearer(token)
```

FUNCTION

Pass a string as parameter containing the OAuth 2.0 Bearer Access Token for use with HTTP, IMAP, POP3 and SMTP servers that support the OAuth 2.0 Authorization Framework.

Note: For IMAP, POP3 and SMTP, the user name used to generate the Bearer Token should be supplied via the `#CURLOPT_USERNAME` option.

INPUTS

`token` input value

5.337 easy:Unescape

NAME

easy:Unescape – URL decodes the given string

SYNOPSIS

```
e$ = easy:Unescape(s$)
```

FUNCTION

This function converts the given URL encoded input string to a "plain string" and returns that. All input characters that are URL encoded (%XX where XX is a two-digit hexadecimal number) are converted to their binary versions.

INPUTS

`s$` string to unescape

RESULTS

`e$` unescaped string

5.338 easy:UnsetOpt

NAME

easy:UnsetOpt – unset option for a curl easy handle

SYNOPSIS

`easy:UnsetOpt(option)`

FUNCTION

This method can be used to unset an option on a curl easy handle, i.e. the option is reset to its default value.

The following option types are currently supported:

`#CURLOPT_ABSTRACT_UNIX_SOCKET`
: Path to an abstract Unix domain socket. See Section 5.339 [easy:UnsetOpt_Abstract_Unix_Socket], page 234, for details.

`#CURLOPT_ACCEPT_ENCODING`
: Accept-Encoding and automatic decompressing data. See Section 5.340 [easy:UnsetOpt_Accept_Encoding], page 235, for details.

`#CURLOPT_ACCEPTTIMEOUT_MS`
: Timeout for waiting for the server's connect back to be accepted. See Section 5.341 [easy:UnsetOpt_AcceptTimeout_MS], page 235, for details.

`#CURLOPT_ADDRESS_SCOPE`
: IPv6 scope for local addresses. See Section 5.342 [easy:UnsetOpt_Address_Scope], page 235, for details.

`#CURLOPT_ALTSVC`
: Specify the Alt-Svc: cache file name. See Section 5.343 [easy:UnsetOpt_AltSvc], page 236, for details. (V2.0)

`#CURLOPT_ALTSVC_CTRL`
: Enable and configure Alt-Svc: treatment. See Section 5.344 [easy:UnsetOpt_AltSvc_Ctrl], page 236, for details. (V2.0)

`#CURLOPT_APPEND`
: Append to remote file. See Section 5.345 [easy:UnsetOpt_Append], page 236, for details.

#CURLOPT_AUTOREFERER
: Automatically get Referer: header. See Section 5.346 [easy:UnsetOpt_AutoReferer], page 236, for details.

#CURLOPT_AWS_SIGV4
: AWS HTTP V4 Signature. See Section 5.347 [easy:UnsetOpt_AWS_SigV4], page 237, for details. (V2.0)

#CURLOPT_BUFFERSIZE
: Ask for alternate buffer size. See Section 5.348 [easy:UnsetOpt_BufferSize], page 237, for details.

#CURLOPT_CA_CACHE_TIMEOUT
: Timeout for CA cache. See Section 5.349 [easy:UnsetOpt_CA_Cache_Timeout], page 237, for details. (V2.0)

#CURLOPT_CAINFO
: CA cert bundle. See Section 5.350 [easy:UnsetOpt_CAInfo], page 238, for details.

#CURLOPT_CAINFO_BLOB
: CA cert bundle memory buffer. See Section 5.351 [easy:UnsetOpt_CAInfo_Blob], page 238, for details. (V2.0)

#CURLOPT_CAPATH
: Path to CA cert bundle. See Section 5.352 [easy:UnsetOpt_CAPath], page 238, for details.

#CURLOPT_CERTINFO
: Extract certificate info. See Section 5.353 [easy:UnsetOpt_CertInfo], page 238, for details.

#CURLOPT_CHUNK_BGN_FUNCTION
: Callback for wildcard download start of chunk. See Section 5.354 [easy:UnsetOpt_Chunk_BGN_Function], page 239, for details.

#CURLOPT_CHUNK_END_FUNCTION
: Callback for wildcard download end of chunk. See Section 5.355 [easy:UnsetOpt_Chunk_End_Function], page 239, for details.

#CURLOPT_CONNECT_ONLY
: Only connect, nothing else. See Section 5.356 [easy:UnsetOpt_Connect_Only], page 239, for details.

#CURLOPT_CONNECT_TO
: Connect to a specific host and port. See Section 5.359 [easy:UnsetOpt_Connect_To], page 240, for details.

#CURLOPT_CONNECTTIMEOUT
: Timeout for the connection phase. See Section 5.357 [easy:UnsetOpt_ConnectTimeout], page 240, for details.

#CURLOPT_CONNECTTIMEOUT_MS
: Millisecond timeout for the connection phase. See Section 5.358 [easy:UnsetOpt_ConnectTimeout_MS], page 240, for details.

#CURLOPT_COOKIE
: Cookie(s) to send. See Section 5.360 [easy:UnsetOpt_Cookie], page 240, for details.

#CURLOPT_COOKIEFILE
: File to read cookies from. See Section 5.361 [easy:UnsetOpt_CookieFile], page 241, for details.

#CURLOPT_COOKIEJAR
: File to write cookies to. See Section 5.362 [easy:UnsetOpt_CookieJar], page 241, for details.

#CURLOPT_COOKIELIST
: Add or control cookies. See Section 5.363 [easy:UnsetOpt_CookieList], page 241, for details.

#CURLOPT_COOKIESESSION
: Start a new cookie session. See Section 5.364 [easy:UnsetOpt_CookieSession], page 242, for details.

#CURLOPT_CRLF
: Convert newlines. See Section 5.365 [easy:UnsetOpt_CRLF], page 242, for details.

#CURLOPT_CRLFILE
: Certificate Revocation List. See Section 5.366 [easy:UnsetOpt_CRLFile], page 242, for details.

#CURLOPT_CURLU
: Set URL to work on with a URL handle. See Section 5.367 [easy:UnsetOpt_CURLU], page 242, for details. (V2.0)

#CURLOPT_CUSTOMREQUEST
: Custom request/method. See Section 5.368 [easy:UnsetOpt_CustomRequest], page 243, for details.

#CURLOPT_DEBUGFUNCTION
: Callback for debug information. See Section 5.369 [easy:UnsetOpt_DebugFunction], page 243, for details.

#CURLOPT_DEFAULT_PROTOCOL
: Default protocol. See Section 5.370 [easy:UnsetOpt_Default_Protocol], page 243, for details.

#CURLOPT_DIRLISTONLY
: List only. See Section 5.371 [easy:UnsetOpt_DirListOnly], page 244, for details.

#CURLOPT_DNS_CACHE_TIMEOUT
: Timeout for DNS cache. See Section 5.373 [easy:UnsetOpt_DNS_Cache_Timeout], page 244, for details.

#CURLOPT_DNS_INTERFACE
: Bind name resolves to this interface. See Section 5.374 [easy:UnsetOpt_DNS_Interface], page 244, for details.

`#CURLOPT_DNS_LOCAL_IP4`
: Bind name resolves to this IP4 address. See Section 5.375 [easy:UnsetOpt_DNS_Local_IP4], page 245, for details.

`#CURLOPT_DNS_LOCAL_IP6`
: Bind name resolves to this IP6 address. See Section 5.376 [easy:UnsetOpt_DNS_Local_IP6], page 245, for details.

`#CURLOPT_DNS_SERVERS`
: Preferred DNS servers. See Section 5.377 [easy:UnsetOpt_DNS_Servers], page 245, for details.

`#CURLOPT_DNS_SHUFFLE_ADDRESSES`
: Shuffle addresses before use. See Section 5.378 [easy:UnsetOpt_DNS_Shuffle_Addresses], page 246, for details. (V2.0)

`#CURLOPT_DNS_USE_GLOBAL_CACHE`
: OBSOLETE Enable global DNS cache. See Section 5.379 [easy:UnsetOpt_DNS_Use_Global_Cache], page 246, for details.

`#CURLOPT_DISALLOW_USERNAME_IN_URL`
: Do not allow username in URL. See Section 5.372 [easy:UnsetOpt_Disallow_Username_In_URL], page 244, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYHOST`
: Verify the host name in the DoH (DNS-over-HTTPS) SSL certificate. See Section 5.380 [easy:UnsetOpt_DoH_SSL_VerifyHost], page 246, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYPEER`
: Verify the DoH (DNS-over-HTTPS) SSL certificate. See Section 5.381 [easy:UnsetOpt_DoH_SSL_VerifyPeer], page 246, for details. (V2.0)

`#CURLOPT_DOH_SSL_VERIFYSTATUS`
: Verify the DoH (DNS-over-HTTPS) SSL certificate's status. See Section 5.382 [easy:UnsetOpt_DoH_SSL_VerifyStatus], page 247, for details. (V2.0)

`#CURLOPT_DOH_URL`
: Use this DoH server for name resolves. See Section 5.383 [easy:UnsetOpt_DoH_URL], page 247, for details. (V2.0)

`#CURLOPT_EGDSOCKET`
: Identify EGD socket for entropy. See Section 5.384 [easy:UnsetOpt_EGDSocket], page 247, for details.

`#CURLOPT_EXPECT_100_TIMEOUT_MS`
: 100-continue timeout. See Section 5.385 [easy:UnsetOpt_Expect_100_Timeout_MS], page 248, for details.

`#CURLOPT_FAILONERROR`
: Fail on HTTP 4xx errors. See Section 5.386 [easy:UnsetOpt_FailOnError], page 248, for details.

#CURLOPT_FILETIME
: Request file modification date and time. See Section 5.387 [easy:UnsetOpt_FileTime], page 248, for details.

#CURLOPT_FNMATCH_FUNCTION
: Callback for wildcard matching. See Section 5.388 [easy:UnsetOpt_FNMatch_Function], page 248, for details.

#CURLOPT_FOLLOWLOCATION
: Follow HTTP redirects. See Section 5.389 [easy:UnsetOpt_FollowLocation], page 249, for details.

#CURLOPT_FORBID_REUSE
: Prevent subsequent connections from re-using this. See Section 5.390 [easy:UnsetOpt_Forbid_Reuse], page 249, for details.

#CURLOPT_FRESH_CONNECT
: Use a new connection. See Section 5.391 [easy:UnsetOpt_Fresh_Connect], page 249, for details.

#CURLOPT_FTP_ACCOUNT
: Send ACCT command. See Section 5.392 [easy:UnsetOpt_FTP_Account], page 250, for details.

#CURLOPT_FTP_ALTERNATIVE_TO_USER
: Alternative to USER. See Section 5.393 [easy:UnsetOpt_FTP_Alternative_To_User], page 250, for details.

#CURLOPT_FTP_CREATE_MISSING_DIRS
: Create missing directories on the remote server. See Section 5.394 [easy:UnsetOpt_FTP_Create_Missing_Dirs], page 250, for details.

#CURLOPT_FTP_FILEMETHOD
: Specify how to reach files. See Section 5.395 [easy:UnsetOpt_FTP_FileMethod], page 250, for details.

#CURLOPT_FTP_RESPONSE_TIMEOUT
: Timeout for FTP responses. See Section 5.397 [easy:UnsetOpt_FTP_Response_Timeout], page 251, for details.

#CURLOPT_FTP_SKIP_PASV_IP
: Ignore the IP address in the PASV response. See Section 5.398 [easy:UnsetOpt_FTP_Skip_PASV_IP], page 251, for details.

#CURLOPT_FTP_SSL_CCC
: Back to non-TLS again after authentication. See Section 5.400 [easy:UnsetOpt_FTP_SSL_CCC], page 252, for details.

#CURLOPT_FTP_USE_EPRT
: Use EPTR. See Section 5.401 [easy:UnsetOpt_FTP_Use_Eprt], page 252, for details.

#CURLOPT_FTP_USE_EPSV
: Use EPSV. See Section 5.402 [easy:UnsetOpt_FTP_Use_Epsv], page 252, for details.

`#CURLOPT_FTP_USE_PRET`
Use PRET. See Section 5.403 [easy:UnsetOpt_FTP_Use_Pret], page 253, for details.

`#CURLOPT_FTPPORT`
Use active FTP. See Section 5.396 [easy:UnsetOpt_FTPPort], page 251, for details.

`#CURLOPT_FTPSSLAUTH`
Control how to do TLS. See Section 5.399 [easy:UnsetOpt_FTPSSLAuth], page 252, for details.

`#CURLOPT_GSSAPI_DELEGATION`
Disable GSS-API delegation. See Section 5.404 [easy:UnsetOpt_GSSAPI_Delegation], page 253, for details.

`#CURLOPT_HAPPY_EYEBALLS_TIMEOUT_MS`
Timeout for happy eyeballs. See Section 5.405 [easy:UnsetOpt_Happy_Eyeballs_Timeout_MS], page 253, for details. (V2.0)

`#CURLOPT_HAPROXYPROTOCOL`
Send an HAProxy PROXY protocol v1 header. See Section 5.406 [easy:UnsetOpt_HAProxyProtocol], page 254, for details. (V2.0)

`#CURLOPT_HEADER`
Include the header in the body output. See Section 5.407 [easy:UnsetOpt_Header], page 254, for details.

`#CURLOPT_HEADERFUNCTION`
Callback for writing received headers. See Section 5.408 [easy:UnsetOpt_HeaderFunction], page 254, for details.

`#CURLOPT_HEADEROPT`
Control custom headers. See Section 5.409 [easy:UnsetOpt_HeaderOpt], page 254, for details.

`#CURLOPT_HSTS`
Set HSTS cache file. See Section 5.410 [easy:UnsetOpt_HSTS], page 255, for details. (V2.0)

`#CURLOPT_HSTS_CTRL`
Enable HSTS. See Section 5.411 [easy:UnsetOpt_HSTS_Ctrl], page 255, for details. (V2.0)

`#CURLOPT_HSTSREADFUNCTION`
Set HSTS read callback. See Section 5.412 [easy:UnsetOpt_HSTSReadFunction], page 255, for details. (V2.0)

`#CURLOPT_HSTSWRITEFUNCTION`
Set HSTS write callback. See Section 5.413 [easy:UnsetOpt_HSTSWriteFunction], page 256, for details. (V2.0)

`#CURLOPT_HTTP09_ALLOWED`
Allow HTTP/0. See Section 5.414 [easy:UnsetOpt_HTTP09_Allowed], page 256, for details. (V2.0)

#CURLOPT_HTTP200ALIASES
: Alternative versions of 200 OK. See Section 5.415 [easy:UnsetOpt_HTTP200Aliases], page 256, for details.

#CURLOPT_HTTP_CONTENT_DECODING
: Disable Content decoding. See Section 5.417 [easy:UnsetOpt_HTTP_Content_Decoding], page 257, for details.

#CURLOPT_HTTP_TRANSFER_DECODING
: Disable Transfer decoding. See Section 5.422 [easy:UnsetOpt_HTTP_Transfer_Decoding], page 258, for details.

#CURLOPT_HTTP_VERSION
: HTTP version to use. See Section 5.423 [easy:UnsetOpt_HTTP_Version], page 258, for details.

#CURLOPT_HTTPAUTH
: HTTP server authentication methods. See Section 5.416 [easy:UnsetOpt_HTTPAuth], page 256, for details.

#CURLOPT_HTTPGET
: Do an HTTP GET request. See Section 5.418 [easy:UnsetOpt_HTTPGet], page 257, for details.

#CURLOPT_HTTPHEADER
: Custom HTTP headers. See Section 5.419 [easy:UnsetOpt_HTTPHeader], page 257, for details.

#CURLOPT_HTTPPOST
: Multipart formpost HTTP POST. See Section 5.420 [easy:UnsetOpt_HTTPPost], page 258, for details.

#CURLOPT_HTTPPROXYTUNNEL
: Tunnel through the HTTP proxy. See Section 5.421 [easy:UnsetOpt_HTTPProxyTunnel], page 258, for details.

#CURLOPT_IGNORE_CONTENT_LENGTH
: Ignore Content-Length. See Section 5.424 [easy:UnsetOpt_Ignore_Content_Length], page 259, for details.

#CURLOPT_INFILESIZE
: Size of file to send. See Section 5.425 [easy:UnsetOpt_InFileSize], page 259, for details.

#CURLOPT_INFILESIZE_LARGE
: Size of file to send. See Section 5.426 [easy:UnsetOpt_InFileSize_Large], page 259, for details.

#CURLOPT_INTERFACE
: Bind connection locally to this. See Section 5.427 [easy:UnsetOpt_Interface], page 260, for details.

#CURLOPT_IPRESOLVE
: IP version to resolve to. See Section 5.428 [easy:UnsetOpt_IPResolve], page 260, for details.

#CURLOPT_ISSUERCERT
Issuer certificate. See Section 5.429 [easy:UnsetOpt_IssuerCert], page 260, for details.

#CURLOPT_ISSUERCERT_BLOB
Issuer certificate memory buffer. See Section 5.430 [easy:UnsetOpt_IssuerCert_Blob], page 260, for details. (V2.0)

#CURLOPT_KEEP_SENDING_ON_ERROR
Keep sending on HTTP >= 300 errors. See Section 5.431 [easy:UnsetOpt_Keep_Sending_On_Error], page 261, for details.

#CURLOPT_KEYPASSWD
Client key password. See Section 5.432 [easy:UnsetOpt_KeyPasswd], page 261, for details.

#CURLOPT_KRBLEVEL
Kerberos security level. See Section 5.433 [easy:UnsetOpt_KRBLevel], page 261, for details.

#CURLOPT_LOCALPORT
Bind connection locally to this port. See Section 5.434 [easy:UnsetOpt_LocalPort], page 262, for details.

#CURLOPT_LOCALPORTRANGE
Bind connection locally to port range. See Section 5.435 [easy:UnsetOpt_LocalPortRange], page 262, for details.

#CURLOPT_LOGIN_OPTIONS
Login options. See Section 5.436 [easy:UnsetOpt_Login_Options], page 262, for details.

#CURLOPT_LOW_SPEED_LIMIT
Low speed limit to abort transfer. See Section 5.437 [easy:UnsetOpt_Low_Speed_Limit], page 262, for details.

#CURLOPT_LOW_SPEED_TIME
Time to be below the speed to trigger low speed abort. See Section 5.438 [easy:UnsetOpt_Low_Speed_Time], page 263, for details.

#CURLOPT_MAIL_AUTH
Authentication address. See Section 5.439 [easy:UnsetOpt_Mail_Auth], page 263, for details.

#CURLOPT_MAIL_FROM
Address of the sender. See Section 5.440 [easy:UnsetOpt_Mail_From], page 263, for details.

#CURLOPT_MAIL_RCPT
Address of the recipients. See Section 5.441 [easy:UnsetOpt_Mail_RCPT], page 264, for details.

#CURLOPT_MAIL_RCPT_ALLOWFAILS
Allow RCPT TO command to fail for some recipients. See Section 5.442 [easy:UnsetOpt_Mail_RCPT_AllowFails], page 264, for details. (V2.0)

#CURLOPT_MAXAGE_CONN
: Limit the age (idle time) of connections for reuse. See Section 5.443 [easy:UnsetOpt_MaxAge_Conn], page 264, for details. (V2.0)

#CURLOPT_MAXLIFETIME_CONN
: Limit the age (since creation) of connections for reuse. See Section 5.447 [easy:UnsetOpt_MaxLifeTime_Conn], page 265, for details. (V2.0)

#CURLOPT_PREREQFUNCTION
: Callback to be called after a connection is established but before a request is made on that connection. See Section 5.472 [easy:UnsetOpt_PreReqFunction], page 272, for details. (V2.0)

#CURLOPT_MAX_RECV_SPEED_LARGE
: Cap the download speed to this. See Section 5.448 [easy:UnsetOpt_Max_Recv_Speed_Large], page 266, for details.

#CURLOPT_MAX_SEND_SPEED_LARGE
: Cap the upload speed to this. See Section 5.450 [easy:UnsetOpt_Max_Send_Speed_Large], page 266, for details.

#CURLOPT_MAXCONNECTS
: Maximum number of connections in the connection pool. See Section 5.444 [easy:UnsetOpt_MaxConnects], page 264, for details.

#CURLOPT_MAXFILESIZE
: Maximum file size to get. See Section 5.445 [easy:UnsetOpt_MaxFileSize], page 265, for details.

#CURLOPT_MAXFILESIZE_LARGE
: Maximum file size to get. See Section 5.446 [easy:UnsetOpt_MaxFileSize_Large], page 265, for details.

#CURLOPT_MAXREDIRS
: Maximum number of redirects to follow. See Section 5.449 [easy:UnsetOpt_MaxRedirs], page 266, for details.

#CURLOPT_MIME_OPTIONS
: Set MIME option flags. See Section 5.451 [easy:UnsetOpt_MIME_Options], page 266, for details. (V2.0)

#CURLOPT_MIMEPOST
: Post/send MIME data. See Section 5.452 [easy:UnsetOpt_MIMEPost], page 267, for details. (V2.0)

#CURLOPT_NETRC
: Enable .netrc parsing. See Section 5.453 [easy:UnsetOpt_Netrc], page 267, for details.

#CURLOPT_NETRC_FILE
: .netrc file name. See Section 5.454 [easy:UnsetOpt_Netrc_File], page 267, for details.

#CURLOPT_NEW_DIRECTORY_PERMS
: Mode for creating new remote directories. See Section 5.455 [easy:UnsetOpt_New_Directory_Perms], page 268, for details.

#CURLOPT_NEW_FILE_PERMS
: Mode for creating new remote files. See Section 5.456 [easy:UnsetOpt_New_File_Perms], page 268, for details.

#CURLOPT_NOBODY
: Do not get the body contents. See Section 5.457 [easy:UnsetOpt_Nobody], page 268, for details.

#CURLOPT_NOPROGRESS
: Shut off the progress meter. See Section 5.458 [easy:UnsetOpt_NoProgress], page 268, for details.

#CURLOPT_NOPROXY
: Filter out hosts from proxy use. See Section 5.459 [easy:UnsetOpt_NoProxy], page 269, for details.

#CURLOPT_NOSIGNAL
: Do not install signal handlers. See Section 5.460 [easy:UnsetOpt_NoSignal], page 269, for details.

#CURLOPT_PASSWORD
: Password. See Section 5.461 [easy:UnsetOpt_Password], page 269, for details.

#CURLOPT_PATH_AS_IS
: Disable squashing /. See Section 5.462 [easy:UnsetOpt_Path_As_Is], page 270, for details.

#CURLOPT_PINNEDPUBLICKEY
: Set pinned SSL public key . See Section 5.463 [easy:UnsetOpt_PinnedPublicKey], page 270, for details.

#CURLOPT_PIPEWAIT
: Wait on connection to pipeline on it. See Section 5.464 [easy:UnsetOpt_PipeWait], page 270, for details.

#CURLOPT_PORT
: Port number to connect to. See Section 5.465 [easy:UnsetOpt_Port], page 270, for details.

#CURLOPT_POST
: How to act on redirects after POST. See Section 5.466 [easy:UnsetOpt_Post], page 271, for details.

#CURLOPT_POSTFIELDS
: Send a POST with this data. See Section 5.467 [easy:UnsetOpt_PostFields], page 271, for details.

#CURLOPT_POSTQUOTE
: Commands to run after transfer. See Section 5.468 [easy:UnsetOpt_PostQuote], page 271, for details.

#CURLOPT_POSTREDIR
How to act on redirects after POST. See Section 5.469 [easy:UnsetOpt_PostRedir], page 272, for details.

#CURLOPT_PRE_PROXY
Socks proxy to use. See Section 5.470 [easy:UnsetOpt_Pre_Proxy], page 272, for details.

#CURLOPT_PREQUOTE
Commands to run just before transfer. See Section 5.471 [easy:UnsetOpt_Prequote], page 272, for details.

#CURLOPT_PROGRESSFUNCTION
Callback for progress meter. See Section 5.473 [easy:UnsetOpt_ProgressFunction], page 273, for details.

#CURLOPT_PROTOCOLS
Allowed protocols. See Section 5.474 [easy:UnsetOpt_Protocols], page 273, for details.

#CURLOPT_PROTOCOLS_STR
Allowed protocols. See Section 5.475 [easy:UnsetOpt_Protocols_Str], page 273, for details. (V2.0)

#CURLOPT_PROXY
Proxy to use. See Section 5.476 [easy:UnsetOpt_Proxy], page 274, for details.

#CURLOPT_PROXY_CAINFO
Proxy CA cert bundle. See Section 5.478 [easy:UnsetOpt_Proxy_CAInfo], page 274, for details.

#CURLOPT_PROXY_CAINFO_BLOB
Proxy CA cert bundle memory buffer. See Section 5.479 [easy:UnsetOpt_Proxy_CAInfo_Blob], page 274, for details. (V2.0)

#CURLOPT_PROXY_CAPATH
Path to proxy CA cert bundle. See Section 5.480 [easy:UnsetOpt_Proxy_CAPath], page 275, for details.

#CURLOPT_PROXY_CRLFILE
Proxy Certificate Revocation List. See Section 5.481 [easy:UnsetOpt_Proxy_CRLFile], page 275, for details.

#CURLOPT_PROXY_ISSUERCERT
Proxy issuer certificate. See Section 5.483 [easy:UnsetOpt_Proxy_IssuerCert], page 276, for details. (V2.0)

#CURLOPT_PROXY_ISSUERCERT_BLOB
Proxy issuer certificate memory buffer. See Section 5.484 [easy:UnsetOpt_Proxy_IssuerCert_Blob], page 276, for details. (V2.0)

#CURLOPT_PROXY_KEYPASSWD
Proxy client key password. See Section 5.485 [easy:UnsetOpt_Proxy_KeyPasswd], page 276, for details.

#CURLOPT_PROXY_PINNEDPUBLICKEY
: Set the proxy's pinned SSL public key. See Section 5.487 [easy:UnsetOpt_Proxy_PinnedPublicKey], page 277, for details.

#CURLOPT_PROXY_SERVICE_NAME
: Proxy authentication service name. See Section 5.489 [easy:UnsetOpt_Proxy_Service_Name], page 277, for details.

#CURLOPT_PROXY_SSLCERT
: Proxy client cert. See Section 5.490 [easy:UnsetOpt_Proxy_SSLCert], page 278, for details.

#CURLOPT_PROXY_SSLCERT_BLOB
: Proxy client cert memory buffer. See Section 5.491 [easy:UnsetOpt_Proxy_SSLCert_Blob], page 278, for details. (V2.0)

#CURLOPT_PROXY_SSLCERTTYPE
: Proxy client cert type. See Section 5.492 [easy:UnsetOpt_Proxy_SSLCertType], page 278, for details.

#CURLOPT_PROXY_SSL_CIPHER_LIST
: Proxy ciphers to use. See Section 5.493 [easy:UnsetOpt_Proxy_SSL_Cipher_List], page 278, for details.

#CURLOPT_PROXY_SSLKEY
: Proxy client key. See Section 5.494 [easy:UnsetOpt_Proxy_SSLKey], page 279, for details.

#CURLOPT_PROXY_SSLKEY_BLOB
: Proxy client key. See Section 5.495 [easy:UnsetOpt_Proxy_SSLKey_Blob], page 279, for details. (V2.0)

#CURLOPT_PROXY_SSLKEYTYPE
: Proxy client key type. See Section 5.496 [easy:UnsetOpt_Proxy_SSLKeyType], page 279, for details.

#CURLOPT_PROXY_SSL_OPTIONS
: Control proxy SSL behavior. See Section 5.497 [easy:UnsetOpt_Proxy_SSL_Options], page 280, for details.

#CURLOPT_PROXY_SSL_VERIFYHOST
: Verify the host name in the proxy SSL certificate. See Section 5.498 [easy:UnsetOpt_Proxy_SSL_VerifyHost], page 280, for details.

#CURLOPT_PROXY_SSL_VERIFYPEER
: Verify the proxy SSL certificate. See Section 5.499 [easy:UnsetOpt_Proxy_SSL_VerifyPeer], page 280, for details.

#CURLOPT_PROXY_SSLVERSION
: Proxy SSL version to use. See Section 5.500 [easy:UnsetOpt_Proxy_SSLVersion], page 280, for details.

#CURLOPT_PROXY_TLSAUTH_PASSWORD
: Proxy TLS authentication password. See Section 5.501 [easy:UnsetOpt_Proxy_TLSAuth_Password], page 281, for details.

#CURLOPT_PROXY_TLSAUTH_TYPE
Proxy TLS authentication methods. See Section 5.502 [easy:UnsetOpt_Proxy_TLSAuth_Type], page 281, for details.

#CURLOPT_PROXY_TLSAUTH_USERNAME
Proxy TLS authentication user name. See Section 5.503 [easy:UnsetOpt_Proxy_TLSAuth_UserName], page 281, for details.

#CURLOPT_PROXY_TRANSFER_MODE
Add transfer mode to URL over proxy. See Section 5.504 [easy:UnsetOpt_Proxy_Transfer_Mode], page 282, for details.

#CURLOPT_PROXYAUTH
HTTP proxy authentication methods. See Section 5.477 [easy:UnsetOpt_ProxyAuth], page 274, for details.

#CURLOPT_PROXYHEADER
Custom HTTP headers sent to proxy. See Section 5.482 [easy:UnsetOpt_ProxyHeader], page 275, for details.

#CURLOPT_PROXYPASSWORD
Proxy password. See Section 5.486 [easy:UnsetOpt_ProxyPassword], page 276, for details.

#CURLOPT_PROXYPORT
Proxy port to use. See Section 5.488 [easy:UnsetOpt_ProxyPort], page 277, for details.

#CURLOPT_PROXYTYPE
Proxy type. See Section 5.505 [easy:UnsetOpt_ProxyType], page 282, for details.

#CURLOPT_PROXYUSERNAME
Proxy user name. See Section 5.506 [easy:UnsetOpt_ProxyUserName], page 282, for details.

#CURLOPT_PROXYUSERPWD
Proxy user name and password. See Section 5.507 [easy:UnsetOpt_ProxyUserPwd], page 282, for details.

#CURLOPT_PUT
Issue an HTTP PUT request. See Section 5.508 [easy:UnsetOpt_Put], page 283, for details.

#CURLOPT_QUICK_EXIT
To be get by toplevel tools like "curl" to skip lengthy cleanups when they are about to call exit() anyway. See Section 5.509 [easy:UnsetOpt_Quick_Exit], page 283, for details. (V2.0)

#CURLOPT_QUOTE
Commands to run before transfer. See Section 5.510 [easy:UnsetOpt_Quote], page 283, for details.

`#CURLOPT_RANDOM_FILE`
: Provide source for entropy random data. See Section 5.511 [easy:UnsetOpt_Random_File], page 284, for details.

`#CURLOPT_RANGE`
: Range requests. See Section 5.512 [easy:UnsetOpt_Range], page 284, for details.

`#CURLOPT_READFUNCTION`
: Callback for reading data. See Section 5.513 [easy:UnsetOpt_ReadFunction], page 284, for details.

`#CURLOPT_REDIR_PROTOCOLS`
: Protocols to allow redirects to. See Section 5.514 [easy:UnsetOpt_Redir_Protocols], page 284, for details.

`#CURLOPT_REDIR_PROTOCOLS_STR`
: Protocols to allow redirects to. See Section 5.515 [easy:UnsetOpt_Redir_Protocols_Str], page 285, for details. (V2.0)

`#CURLOPT_REFERER`
: Referer: header. See Section 5.516 [easy:UnsetOpt_Referer], page 285, for details.

`#CURLOPT_REQUEST_TARGET`
: Set the request target. See Section 5.517 [easy:UnsetOpt_Request_Target], page 285, for details.

`#CURLOPT_RESOLVE`
: Callback to be called before a new resolve request is started. See Section 5.518 [easy:UnsetOpt_Resolve], page 286, for details.

`#CURLOPT_RESOLVER_START_FUNCTION`
: Callback to be called before a new resolve request is started. See Section 5.519 [easy:UnsetOpt_Resolver_Start_Function], page 286, for details. (V2.0)

`#CURLOPT_RESUME_FROM`
: Resume a transfer. See Section 5.520 [easy:UnsetOpt_Resume_From], page 286, for details.

`#CURLOPT_RESUME_FROM_LARGE`
: Resume a transfer. See Section 5.521 [easy:UnsetOpt_Resume_From_Large], page 286, for details.

`#CURLOPT_RTSP_CLIENT_CSEQ`
: Client CSEQ number. See Section 5.522 [easy:UnsetOpt_RTSP_Client_CSeq], page 287, for details.

`#CURLOPT_RTSP_REQUEST`
: RTSP request. See Section 5.523 [easy:UnsetOpt_RTSP_Request], page 287, for details.

#CURLOPT_RTSP_SERVER_CSEQ
CSEQ number for RTSP Server->Client request. See Section 5.524 [easy:UnsetOpt_RTSP_Server_CSeq], page 287, for details.

#CURLOPT_RTSP_SESSION_ID
RTSP session-id. See Section 5.525 [easy:UnsetOpt_RTSP_Session_ID], page 288, for details.

#CURLOPT_RTSP_STREAM_URI
RTSP stream URI. See Section 5.526 [easy:UnsetOpt_RTSP_Stream_URI], page 288, for details.

#CURLOPT_RTSP_TRANSPORT
RTSP Transport: header. See Section 5.527 [easy:UnsetOpt_RTSP_Transport], page 288, for details.

#CURLOPT_SASL_AUTHZID
SASL authorization identity (identity to act as). See Section 5.528 [easy:UnsetOpt_SASL_AuthZID], page 288, for details. (V2.0)

#CURLOPT_SASL_IR
Enable SASL initial response. See Section 5.529 [easy:UnsetOpt_SASL_IR], page 289, for details.

#CURLOPT_SEEKFUNCTION
Callback for seek operations. See Section 5.530 [easy:UnsetOpt_SeekFunction], page 289, for details.

#CURLOPT_SERVICE_NAME
Authentication service name. See Section 5.531 [easy:UnsetOpt_Service_Name], page 289, for details.

#CURLOPT_SHARE
Share object to use. See Section 5.532 [easy:UnsetOpt_Share], page 290, for details.

#CURLOPT_SOCKS5_AUTH
Socks5 authentication methods. See Section 5.533 [easy:UnsetOpt_Socks5_Auth], page 290, for details.

#CURLOPT_SOCKS5_GSSAPI_NEC
Socks5 GSSAPI NEC mode. See Section 5.534 [easy:UnsetOpt_Socks5_GSSAPI_NEC], page 290, for details.

#CURLOPT_SOCKS5_GSSAPI_SERVICE
Socks5 GSSAPI service name. See Section 5.535 [easy:UnsetOpt_Socks5_GSSAPI_Service], page 290, for details.

#CURLOPT_SSH_AUTH_TYPES
SSH authentication types. See Section 5.536 [easy:UnsetOpt_SSH_Auth_Types], page 291, for details.

#CURLOPT_SSH_COMPRESSION
Enable SSH compression. See Section 5.537 [easy:UnsetOpt_SSH_Compression], page 291, for details. (V2.0)

#CURLOPT_SSH_HOSTKEYFUNCTION
: Callback for checking host key handling. See Section 5.538 [easy:UnsetOpt_SSH_HostKeyFunction], page 291, for details. (V2.0)

#CURLOPT_SSH_HOST_PUBLIC_KEY_MD5
: MD5 of host's public key. See Section 5.539 [easy:UnsetOpt_SSH_Host_Public_Key_MD5], page 292, for details.

#CURLOPT_SSH_KNOWNHOSTS
: File name with known hosts. See Section 5.540 [easy:UnsetOpt_SSH_KnownHosts], page 292, for details.

#CURLOPT_SSH_PRIVATE_KEYFILE
: File name of private key. See Section 5.541 [easy:UnsetOpt_SSH_Private_KeyFile], page 292, for details.

#CURLOPT_SSH_PUBLIC_KEYFILE
: File name of public key. See Section 5.542 [easy:UnsetOpt_SSH_Public_KeyFile], page 292, for details.

#CURLOPT_SSLCERT
: Client cert. See Section 5.543 [easy:UnsetOpt_SSLCert], page 293, for details.

#CURLOPT_SSLCERT_BLOB
: Client cert memory buffer. See Section 5.544 [easy:UnsetOpt_SSLCert_Blob], page 293, for details. (V2.0)

#CURLOPT_SSLCERTTYPE
: Client cert type. See Section 5.545 [easy:UnsetOpt_SSLCertType], page 293, for details.

#CURLOPT_SSL_CIPHER_LIST
: Ciphers to use. See Section 5.546 [easy:UnsetOpt_SSL_Cipher_List], page 294, for details.

#CURLOPT_SSL_EC_CURVES
: Set key exchange curves. See Section 5.547 [easy:UnsetOpt_SSL_EC_Curves], page 294, for details. (V2.0)

#CURLOPT_SSL_ENABLE_ALPN
: Enable use of ALPN. See Section 5.548 [easy:UnsetOpt_SSL_Enable_Alpn], page 294, for details.

#CURLOPT_SSL_ENABLE_NPN
: Enable use of NPN. See Section 5.549 [easy:UnsetOpt_SSL_Enable_Npn], page 294, for details.

#CURLOPT_SSLENGINE
: Use identifier with SSL engine. See Section 5.550 [easy:UnsetOpt_SSLEngine], page 295, for details.

#CURLOPT_SSLENGINE_DEFAULT
: Default SSL engine. See Section 5.551 [easy:UnsetOpt_SSLEngine_Default], page 295, for details.

#CURLOPT_SSL_FALSESTART
: Enable TLS False Start. See Section 5.552 [easy:UnsetOpt_SSL_FalseStart], page 295, for details.

#CURLOPT_SSLKEY
: Client key. See Section 5.553 [easy:UnsetOpt_SSLKey], page 296, for details.

#CURLOPT_SSLKEY_BLOB
: Client key memory buffer. See Section 5.554 [easy:UnsetOpt_SSLKey_Blob], page 296, for details. (V2.0)

#CURLOPT_SSLKEYTYPE
: Client key type. See Section 5.555 [easy:UnsetOpt_SSLKeyType], page 296, for details.

#CURLOPT_SSL_OPTIONS
: Control SSL behavior. See Section 5.556 [easy:UnsetOpt_SSL_Options], page 296, for details.

#CURLOPT_SSL_SESSIONID_CACHE
: Disable SSL session-id cache. See Section 5.557 [easy:UnsetOpt_SSL_SessionID_Cache], page 297, for details.

#CURLOPT_SSL_VERIFYHOST
: Verify the host name in the SSL certificate. See Section 5.558 [easy:UnsetOpt_SSL_VerifyHost], page 297, for details.

#CURLOPT_SSL_VERIFYPEER
: Verify the SSL certificate. See Section 5.559 [easy:UnsetOpt_SSL_VerifyPeer], page 297, for details.

#CURLOPT_SSL_VERIFYSTATUS
: Verify the SSL certificate's status. See Section 5.560 [easy:UnsetOpt_SSL_VerifyStatus], page 298, for details.

#CURLOPT_SSLVERSION
: SSL version to use. See Section 5.561 [easy:UnsetOpt_SSLVersion], page 298, for details.

#CURLOPT_STREAM_DEPENDS
: This HTTP/2 stream depends on another. See Section 5.562 [easy:UnsetOpt_Stream_Depends], page 298, for details.

#CURLOPT_STREAM_DEPENDS_E
: This HTTP/2 stream depends on another exclusively. See Section 5.563 [easy:UnsetOpt_Stream_Depends_e], page 298, for details.

#CURLOPT_STREAM_WEIGHT
: Set this HTTP/2 stream's weight. See Section 5.564 [easy:UnsetOpt_Stream_Weight], page 299, for details.

#CURLOPT_SUPPRESS_CONNECT_HEADERS
: Suppress proxy CONNECT response headers from user callbacks. See Section 5.565 [easy:UnsetOpt_Suppress_Connect_Headers], page 299, for details.

#CURLOPT_TCP_FASTOPEN
: Enable TFO, TCP Fast Open. See Section 5.566 [easy:UnsetOpt_TCP_FastOpen], page 299, for details.

#CURLOPT_TCP_KEEPALIVE
: Enable TCP keep-alive. See Section 5.567 [easy:UnsetOpt_TCP_KeepAlive], page 300, for details.

#CURLOPT_TCP_KEEPIDLE
: Idle time before sending keep-alive. See Section 5.568 [easy:UnsetOpt_TCP_KeepIdle], page 300, for details.

#CURLOPT_TCP_KEEPINTVL
: Interval between keep-alive probes. See Section 5.569 [easy:UnsetOpt_TCP_KeepIntvl], page 300, for details.

#CURLOPT_TCP_NODELAY
: Disable the Nagle algorithm. See Section 5.570 [easy:UnsetOpt_TCP_NoDelay], page 300, for details.

#CURLOPT_TELNETOPTIONS
: TELNET options. See Section 5.571 [easy:UnsetOpt_TelnetOptions], page 301, for details.

#CURLOPT_TFTP_BLKSIZE
: TFTP block size. See Section 5.572 [easy:UnsetOpt_TFTP_BlkSize], page 301, for details.

#CURLOPT_TFTP_NO_OPTIONS
: Do not send TFTP options requests. See Section 5.573 [easy:UnsetOpt_TFTP_No_Options], page 301, for details.

#CURLOPT_TIMECONDITION
: Make a time conditional request. See Section 5.574 [easy:UnsetOpt_TimeCondition], page 302, for details.

#CURLOPT_TIMEOUT
: Timeout for the entire request. See Section 5.575 [easy:UnsetOpt_Timeout], page 302, for details.

#CURLOPT_TIMEOUT_MS
: Millisecond timeout for the entire request. See Section 5.576 [easy:UnsetOpt_Timeout_MS], page 302, for details.

#CURLOPT_TIMEVALUE
: Time value for the time conditional request. See Section 5.577 [easy:UnsetOpt_TimeValue], page 302, for details.

#CURLOPT_TIMEVALUE_LARGE
: Time value for the time conditional request. See Section 5.578 [easy:UnsetOpt_TimeValue_Large], page 303, for details. (V2.0)

#CURLOPT_TLS13_CIPHERS
: Ciphers suites to use for TLS 1.3. See Section 5.579 [easy:UnsetOpt_TLS13_Ciphers], page 303, for details. (V2.0)

#CURLOPT_TLSAUTH_PASSWORD
: TLS authentication password. See Section 5.580 [easy:UnsetOpt_TLSAuth_Password], page 303, for details.

#CURLOPT_TLSAUTH_TYPE
: TLS authentication methods. See Section 5.581 [easy:UnsetOpt_TLSAuth_Type], page 304, for details.

#CURLOPT_TLSAUTH_USERNAME
: TLS authentication user name. See Section 5.582 [easy:UnsetOpt_TLSAuth_UserName], page 304, for details.

#CURLOPT_TRAILERFUNCTION
: Set callback for sending trailing headers. See Section 5.583 [easy:UnsetOpt_TrailerFunction], page 304, for details. (V2.0)

#CURLOPT_TRANSFER_ENCODING
: Request Transfer-Encoding. See Section 5.584 [easy:UnsetOpt_Transfer_Encoding], page 304, for details.

#CURLOPT_TRANSFERTEXT
: Use text transfer. See Section 5.585 [easy:UnsetOpt_TransferText], page 305, for details.

#CURLOPT_UNIX_SOCKET_PATH
: Path to a Unix domain socket. See Section 5.586 [easy:UnsetOpt_Unix_Socket_Path], page 305, for details.

#CURLOPT_UNRESTRICTED_AUTH
: Do not restrict authentication to original host. See Section 5.587 [easy:UnsetOpt_Unrestricted_Auth], page 305, for details.

#CURLOPT_UPKEEP_INTERVAL_MS
: Sets the interval at which connection upkeep are performed. See Section 5.588 [easy:UnsetOpt_Upkeep_Interval_MS], page 306, for details. (V2.0)

#CURLOPT_UPLOAD
: Upload data. See Section 5.589 [easy:UnsetOpt_Upload], page 306, for details.

#CURLOPT_UPLOAD_BUFFERSIZE
: Set upload buffer size. See Section 5.590 [easy:UnsetOpt_Upload_Buffersize], page 306, for details. (V2.0)

#CURLOPT_URL
: URL to work on. See Section 5.591 [easy:UnsetOpt_URL], page 306, for details.

#CURLOPT_USE_SSL
: Use TLS/SSL. See Section 5.595 [easy:UnsetOpt_Use_SSL], page 308, for details.

`#CURLOPT_USERAGENT`
User-Agent: header. See Section 5.592 [easy:UnsetOpt_UserAgent], page 307, for details.

`#CURLOPT_USERNAME`
User name. See Section 5.593 [easy:UnsetOpt_UserName], page 307, for details.

`#CURLOPT_USERPWD`
User name and password. See Section 5.594 [easy:UnsetOpt_UserPwd], page 307, for details.

`#CURLOPT_VERBOSE`
Display verbose information. See Section 5.596 [easy:UnsetOpt_Verbose], page 308, for details.

`#CURLOPT_WILDCARDMATCH`
Transfer multiple files according to a file name pattern. See Section 5.597 [easy:UnsetOpt_WildcardMatch], page 308, for details.

`#CURLOPT_WRITEFUNCTION`
Callback for writing data. See Section 5.598 [easy:UnsetOpt_WriteFunction], page 308, for details.

`#CURLOPT_WS_OPTIONS`
Set WebSocket options. See Section 5.599 [easy:UnsetOpt_WS_Options], page 309, for details. (V2.0)

`#CURLOPT_XOAUTH2_BEARER`
OAuth2 bearer token. See Section 5.600 [easy:UnsetOpt_XOAuth2_Bearer], page 309, for details.

INPUTS

`option` option type to unset

EXAMPLE

```
e:UnsetOpt(#CURLOPT_URL)
e:UnsetOpt(#CURLOPT_VERBOSE)
e:UnsetOpt(#CURLOPT_FOLLOWLOCATION)
```

The code above unsets some options on an easy handle, i.e. it resets those options to their default values.

5.339 easy:UnsetOpt_Abstract_Unix_Socket

NAME

easy:UnsetOpt_Abstract_Unix_Socket – get an abstract Unix domain socket

SYNOPSIS

```
easy:UnsetOpt_Abstract_Unix_Socket()
```

FUNCTION

See Section 5.75 [easy:SetOpt_Abstract_Unix_Socket], page 74, for details.

INPUTS
none

5.340 easy:UnsetOpt_Accept_Encoding

NAME
easy:UnsetOpt_Accept_Encoding – enables automatic decompression of HTTP downloads

SYNOPSIS
`easy:UnsetOpt_Accept_Encoding()`

FUNCTION
See Section 5.76 [`easy:SetOpt_Accept_Encoding`], page 75, for details.

INPUTS
none

5.341 easy:UnsetOpt_AcceptTimeout_MS

NAME
easy:UnsetOpt_AcceptTimeout_MS – timeout waiting for FTP server to connect back

SYNOPSIS
`easy:UnsetOpt_AcceptTimeout_MS()`

FUNCTION
See Section 5.77 [`easy:SetOpt_AcceptTimeout_MS`], page 76, for details.

INPUTS
none

5.342 easy:UnsetOpt_Address_Scope

NAME
easy:UnsetOpt_Address_Scope – get scope for local IPv6 addresses

SYNOPSIS
`easy:UnsetOpt_Address_Scope()`

FUNCTION
See Section 5.78 [`easy:SetOpt_Address_Scope`], page 76, for details.

INPUTS
none

5.343 easy:UnsetOpt_AltSvc

NAME
easy:UnsetOpt_AltSvc – alt-svc cache file name (V2.0)

SYNOPSIS
`easy:UnsetOpt_AltSvc()`

FUNCTION
See Section 5.79 [`easy:SetOpt_AltSvc`], page 77, for details.

INPUTS
none

5.344 easy:UnsetOpt_AltSvc_Ctrl

NAME
easy:UnsetOpt_AltSvc_Ctrl – control alt-svc behavior (V2.0)

SYNOPSIS
`easy:UnsetOpt_AltSvc_Ctrl()`

FUNCTION
See Section 5.80 [`easy:SetOpt_AltSvc_Ctrl`], page 77, for details.

INPUTS
none

5.345 easy:UnsetOpt_Append

NAME
easy:UnsetOpt_Append – enable appending to the remote file

SYNOPSIS
`easy:UnsetOpt_Append()`

FUNCTION
See Section 5.81 [`easy:SetOpt_Append`], page 78, for details.

INPUTS
none

5.346 easy:UnsetOpt_AutoReferer

NAME
easy:UnsetOpt_AutoReferer – automatically update the referer header

SYNOPSIS
`easy:UnsetOpt_AutoReferer()`

FUNCTION
See Section 5.82 [easy:SetOpt_AutoReferer], page 78, for details.

INPUTS
none

5.347 easy:UnsetOpt_AWS_SigV4

NAME
easy:UnsetOpt_AWS_SigV4 – V4 signature (V2.0)

SYNOPSIS
`easy:UnsetOpt_AWS_SigV4()`

FUNCTION
See Section 5.83 [easy:SetOpt_AWS_SigV4], page 78, for details.

INPUTS
none

5.348 easy:UnsetOpt_BufferSize

NAME
easy:UnsetOpt_BufferSize – get preferred receive buffer size

SYNOPSIS
`easy:UnsetOpt_BufferSize()`

FUNCTION
See Section 5.84 [easy:SetOpt_BufferSize], page 79, for details.

INPUTS
none

5.349 easy:UnsetOpt_CA_Cache_Timeout

NAME
easy:UnsetOpt_CA_Cache_Timeout – life-time for cached certificate stores (V2.0)

SYNOPSIS
`easy:UnsetOpt_CA_Cache_Timeout()`

FUNCTION
See Section 5.85 [easy:SetOpt_CA_Cache_Timeout], page 80, for details.

INPUTS
none

5.350 easy:UnsetOpt_CAInfo

NAME
easy:UnsetOpt_CAInfo – path to Certificate Authority (CA) bundle

SYNOPSIS
`easy:UnsetOpt_CAInfo()`

FUNCTION
See Section 5.86 [easy:SetOpt_CAInfo], page 80, for details.

INPUTS
none

5.351 easy:UnsetOpt_CAInfo_Blob

NAME
easy:UnsetOpt_CAInfo_Blob – Certificate Authority (CA) bundle in PEM format (V2.0)

SYNOPSIS
`easy:UnsetOpt_CAInfo_Blob()`

FUNCTION
See Section 5.87 [easy:SetOpt_CAInfo_Blob], page 81, for details.

INPUTS
none

5.352 easy:UnsetOpt_CAPath

NAME
easy:UnsetOpt_CAPath – specify directory holding CA certificates

SYNOPSIS
`easy:UnsetOpt_CAPath()`

FUNCTION
See Section 5.88 [easy:SetOpt_CAPath], page 81, for details.

INPUTS
none

5.353 easy:UnsetOpt_CertInfo

NAME
easy:UnsetOpt_CertInfo – request SSL certificate information

SYNOPSIS
`easy:UnsetOpt_CertInfo()`

FUNCTION
See Section 5.89 [easy:SetOpt_CertInfo], page 82, for details.

INPUTS
none

5.354 easy:UnsetOpt_Chunk_BGN_Function

NAME
easy:UnsetOpt_Chunk_BGN_Function – callback before a transfer with FTP wildcard-match

SYNOPSIS
`easy:UnsetOpt_Chunk_BGN_Function()`

FUNCTION
See Section 5.90 [easy:SetOpt_Chunk_BGN_Function], page 82, for details.

INPUTS
none

5.355 easy:UnsetOpt_Chunk_End_Function

NAME
easy:UnsetOpt_Chunk_End_Function – callback after a transfer with FTP wildcard-match

SYNOPSIS
`easy:UnsetOpt_Chunk_End_Function()`

FUNCTION
See Section 5.91 [easy:SetOpt_Chunk_End_Function], page 83, for details.

INPUTS
none

5.356 easy:UnsetOpt_Connect_Only

NAME
easy:UnsetOpt_Connect_Only – stop when connected to target server

SYNOPSIS
`easy:UnsetOpt_Connect_Only()`

FUNCTION
See Section 5.92 [easy:SetOpt_Connect_Only], page 84, for details.

INPUTS
none

5.357 easy:UnsetOpt_ConnectTimeout

NAME
easy:UnsetOpt_ConnectTimeout – timeout for the connect phase

SYNOPSIS
`easy:UnsetOpt_ConnectTimeout()`

FUNCTION
See Section 5.93 [easy:SetOpt_ConnectTimeout], page 84, for details.

INPUTS
none

5.358 easy:UnsetOpt_ConnectTimeout_MS

NAME
easy:UnsetOpt_ConnectTimeout_MS – timeout for the connect phase

SYNOPSIS
`easy:UnsetOpt_ConnectTimeout_MS()`

FUNCTION
See Section 5.94 [easy:SetOpt_ConnectTimeout_MS], page 84, for details.

INPUTS
none

5.359 easy:UnsetOpt_Connect_To

NAME
easy:UnsetOpt_Connect_To – Connect to a specific host and port instead of the URL's host and port

SYNOPSIS
`easy:UnsetOpt_Connect_To()`

FUNCTION
See Section 5.95 [easy:SetOpt_Connect_To], page 85, for details.

INPUTS
none

5.360 easy:UnsetOpt_Cookie

NAME
easy:UnsetOpt_Cookie – get contents of HTTP Cookie header

SYNOPSIS
`easy:UnsetOpt_Cookie()`

FUNCTION
See Section 5.96 [easy:SetOpt_Cookie], page 86, for details.

INPUTS
none

5.361 easy:UnsetOpt_CookieFile

NAME
easy:UnsetOpt_CookieFile – file name to read cookies from

SYNOPSIS
`easy:UnsetOpt_CookieFile()`

FUNCTION
See Section 5.97 [easy:SetOpt_CookieFile], page 87, for details.

INPUTS
none

5.362 easy:UnsetOpt_CookieJar

NAME
easy:UnsetOpt_CookieJar – file name to store cookies to

SYNOPSIS
`easy:UnsetOpt_CookieJar()`

FUNCTION
See Section 5.98 [easy:SetOpt_CookieJar], page 87, for details.

INPUTS
none

5.363 easy:UnsetOpt_CookieList

NAME
easy:UnsetOpt_CookieList – add to or manipulate cookies held in memory

SYNOPSIS
`easy:UnsetOpt_CookieList()`

FUNCTION
See Section 5.99 [easy:SetOpt_CookieList], page 88, for details.

INPUTS
none

5.364 easy:UnsetOpt_CookieSession

NAME
easy:UnsetOpt_CookieSession – start a new cookie session

SYNOPSIS
`easy:UnsetOpt_CookieSession()`

FUNCTION
See Section 5.100 [`easy:SetOpt_CookieSession`], page 88, for details.

INPUTS
none

5.365 easy:UnsetOpt_CRLF

NAME
easy:UnsetOpt_CRLF – enable/disable CRLF conversion

SYNOPSIS
`easy:UnsetOpt_CRLF()`

FUNCTION
See Section 5.101 [`easy:SetOpt_CRLF`], page 89, for details.

INPUTS
none

5.366 easy:UnsetOpt_CRLFile

NAME
easy:UnsetOpt_CRLFile – specify a Certificate Revocation List file

SYNOPSIS
`easy:UnsetOpt_CRLFile()`

FUNCTION
See Section 5.102 [`easy:SetOpt_CRLFile`], page 89, for details.

INPUTS
none

5.367 easy:UnsetOpt_CURLU

NAME
easy:UnsetOpt_CURLU – URL in URL handle format (V2.0)

SYNOPSIS
`easy:UnsetOpt_CURLU()`

FUNCTION
See Section 5.103 [easy:SetOpt_CURLU], page 90, for details.

INPUTS
none

5.368 easy:UnsetOpt_CustomRequest

NAME
easy:UnsetOpt_CustomRequest – custom string for request

SYNOPSIS
`easy:UnsetOpt_CustomRequest()`

FUNCTION
See Section 5.104 [easy:SetOpt_CustomRequest], page 90, for details.

INPUTS
none

5.369 easy:UnsetOpt_DebugFunction

NAME
easy:UnsetOpt_DebugFunction – debug callback

SYNOPSIS
`easy:UnsetOpt_DebugFunction()`

FUNCTION
See Section 5.105 [easy:SetOpt_DebugFunction], page 91, for details.

INPUTS
none

5.370 easy:UnsetOpt_Default_Protocol

NAME
easy:UnsetOpt_Default_Protocol – default protocol to use if the URL is missing a

SYNOPSIS
`easy:UnsetOpt_Default_Protocol()`

FUNCTION
See Section 5.106 [easy:SetOpt_Default_Protocol], page 92, for details.

INPUTS
none

5.371 easy:UnsetOpt_DirListOnly

NAME
easy:UnsetOpt_DirListOnly – ask for names only in a directory listing

SYNOPSIS
`easy:UnsetOpt_DirListOnly()`

FUNCTION
See Section 5.107 [easy:SetOpt_DirListOnly], page 93, for details.

INPUTS
none

5.372 easy:UnsetOpt_Disallow_Username_In_URL

NAME
easy:UnsetOpt_Disallow_Username_In_URL – disallow specifying username in the URL (V2.0)

SYNOPSIS
`easy:UnsetOpt_Disallow_Username_In_URL()`

FUNCTION
See Section 5.108 [easy:SetOpt_Disallow_Username_In_URL], page 94, for details.

INPUTS
none

5.373 easy:UnsetOpt_DNS_Cache_Timeout

NAME
easy:UnsetOpt_DNS_Cache_Timeout – get life-time for DNS cache entries

SYNOPSIS
`easy:UnsetOpt_DNS_Cache_Timeout()`

FUNCTION
See Section 5.109 [easy:SetOpt_DNS_Cache_Timeout], page 94, for details.

INPUTS
none

5.374 easy:UnsetOpt_DNS_Interface

NAME
easy:UnsetOpt_DNS_Interface – get interface to speak DNS over

SYNOPSIS
`easy:UnsetOpt_DNS_Interface()`

FUNCTION
See Section 5.110 [easy:SetOpt_DNS_Interface], page 95, for details.

INPUTS
none

5.375 easy:UnsetOpt_DNS_Local_IP4

NAME
easy:UnsetOpt_DNS_Local_IP4 – IPv4 address to bind DNS resolves to

SYNOPSIS
`easy:UnsetOpt_DNS_Local_IP4()`

FUNCTION
See Section 5.111 [easy:SetOpt_DNS_Local_IP4], page 95, for details.

INPUTS
none

5.376 easy:UnsetOpt_DNS_Local_IP6

NAME
easy:UnsetOpt_DNS_Local_IP6 – IPv6 address to bind DNS resolves to

SYNOPSIS
`easy:UnsetOpt_DNS_Local_IP6()`

FUNCTION
See Section 5.112 [easy:SetOpt_DNS_Local_IP6], page 95, for details.

INPUTS
none

5.377 easy:UnsetOpt_DNS_Servers

NAME
easy:UnsetOpt_DNS_Servers – get preferred DNS servers

SYNOPSIS
`easy:UnsetOpt_DNS_Servers()`

FUNCTION
See Section 5.113 [easy:SetOpt_DNS_Servers], page 96, for details.

INPUTS
none

5.378 easy:UnsetOpt_DNS_Shuffle_Addresses

NAME
easy:UnsetOpt_DNS_Shuffle_Addresses – shuffle IP addresses for hostname (V2.0)

SYNOPSIS
`easy:UnsetOpt_DNS_Shuffle_Addresses()`

FUNCTION
See Section 5.114 [`easy:SetOpt_DNS_Shuffle_Addresses`], page 96, for details.

INPUTS
none

5.379 easy:UnsetOpt_DNS_Use_Global_Cache

NAME
easy:UnsetOpt_DNS_Use_Global_Cache – enable/disable global DNS cache

SYNOPSIS
`easy:UnsetOpt_DNS_Use_Global_Cache()`

FUNCTION
See Section 5.115 [`easy:SetOpt_DNS_Use_Global_Cache`], page 96, for details.

INPUTS
none

5.380 easy:UnsetOpt_DoH_SSL_VerifyHost

NAME
easy:UnsetOpt_DoH_SSL_VerifyHost – verify the host name in the DoH SSL certificate (V2.0)

SYNOPSIS
`easy:UnsetOpt_DoH_SSL_VerifyHost()`

FUNCTION
See Section 5.116 [`easy:SetOpt_DoH_SSL_VerifyHost`], page 97, for details.

INPUTS
none

5.381 easy:UnsetOpt_DoH_SSL_VerifyPeer

NAME
easy:UnsetOpt_DoH_SSL_VerifyPeer – verify the DoH SSL certificate (V2.0)

SYNOPSIS
`easy:UnsetOpt_DoH_SSL_VerifyPeer()`

FUNCTION
See Section 5.117 [easy:SetOpt_DoH_SSL_VerifyPeer], page 97, for details.

INPUTS
none

5.382 easy:UnsetOpt_DoH_SSL_VerifyStatus

NAME
easy:UnsetOpt_DoH_SSL_VerifyStatus – verify the DoH SSL certificate's status (V2.0)

SYNOPSIS
`easy:UnsetOpt_DoH_SSL_VerifyStatus()`

FUNCTION
See Section 5.118 [easy:SetOpt_DoH_SSL_VerifyStatus], page 98, for details.

INPUTS
none

5.383 easy:UnsetOpt_DoH_URL

NAME
easy:UnsetOpt_DoH_URL – provide the DNS-over-HTTPS URL (V2.0)

SYNOPSIS
`easy:UnsetOpt_DoH_URL()`

FUNCTION
See Section 5.119 [easy:SetOpt_DoH_URL], page 99, for details.

INPUTS
none

5.384 easy:UnsetOpt_EGDSocket

NAME
easy:UnsetOpt_EGDSocket – get EGD socket path

SYNOPSIS
`easy:UnsetOpt_EGDSocket()`

FUNCTION
See Section 5.120 [easy:SetOpt_EGDSocket], page 99, for details.

INPUTS
none

5.385 easy:UnsetOpt_Expect_100_Timeout_MS

NAME
easy:UnsetOpt_Expect_100_Timeout_MS – timeout for Expect: 100-continue response

SYNOPSIS
`easy:UnsetOpt_Expect_100_Timeout_MS()`

FUNCTION
See Section 5.121 [easy:SetOpt_Expect_100_Timeout_MS], page 99, for details.

INPUTS
none

5.386 easy:UnsetOpt_FailOnError

NAME
easy:UnsetOpt_FailOnError – request failure on HTTP response >= 400

SYNOPSIS
`easy:UnsetOpt_FailOnError()`

FUNCTION
See Section 5.122 [easy:SetOpt_FailOnError], page 100, for details.

INPUTS
none

5.387 easy:UnsetOpt_FileTime

NAME
easy:UnsetOpt_FileTime – get the modification time of the remote resource

SYNOPSIS
`easy:UnsetOpt_FileTime()`

FUNCTION
See Section 5.123 [easy:SetOpt_FileTime], page 100, for details.

INPUTS
none

5.388 easy:UnsetOpt_FNMatch_Function

NAME
easy:UnsetOpt_FNMatch_Function – wildcard matching function callback

SYNOPSIS
`easy:UnsetOpt_FNMatch_Function()`

FUNCTION
See Section 5.124 [easy:SetOpt_FNMatch_Function], page 101, for details.

INPUTS
none

5.389 easy:UnsetOpt_FollowLocation

NAME
easy:UnsetOpt_FollowLocation – follow HTTP 3xx redirects

SYNOPSIS
`easy:UnsetOpt_FollowLocation()`

FUNCTION
See Section 5.125 [easy:SetOpt_FollowLocation], page 101, for details.

INPUTS
none

5.390 easy:UnsetOpt_Forbid_Reuse

NAME
easy:UnsetOpt_Forbid_Reuse – make connection get closed at once after use

SYNOPSIS
`easy:UnsetOpt_Forbid_Reuse()`

FUNCTION
See Section 5.126 [easy:SetOpt_Forbid_Reuse], page 102, for details.

INPUTS
none

5.391 easy:UnsetOpt_Fresh_Connect

NAME
easy:UnsetOpt_Fresh_Connect – force a new connection to be used

SYNOPSIS
`easy:UnsetOpt_Fresh_Connect()`

FUNCTION
See Section 5.127 [easy:SetOpt_Fresh_Connect], page 102, for details.

INPUTS
none

5.392 easy:UnsetOpt_FTP_Account

NAME
easy:UnsetOpt_FTP_Account – get account info for FTP

SYNOPSIS
`easy:UnsetOpt_FTP_Account()`

FUNCTION
See Section 5.128 [easy:SetOpt_FTP_Account], page 103, for details.

INPUTS
none

5.393 easy:UnsetOpt_FTP_Alternative_To_User

NAME
easy:UnsetOpt_FTP_Alternative_To_User – command to use instead of USER with FTP

SYNOPSIS
`easy:UnsetOpt_FTP_Alternative_To_User()`

FUNCTION
See Section 5.129 [easy:SetOpt_FTP_Alternative_To_User], page 103, for details.

INPUTS
none

5.394 easy:UnsetOpt_FTP_Create_Missing_Dirs

NAME
easy:UnsetOpt_FTP_Create_Missing_Dirs – create missing dirs for FTP and SFTP

SYNOPSIS
`easy:UnsetOpt_FTP_Create_Missing_Dirs()`

FUNCTION
See Section 5.130 [easy:SetOpt_FTP_Create_Missing_Dirs], page 103, for details.

INPUTS
none

5.395 easy:UnsetOpt_FTP_FileMethod

NAME
easy:UnsetOpt_FTP_FileMethod – select directory traversing method for FTP

SYNOPSIS
`easy:UnsetOpt_FTP_FileMethod()`

FUNCTION
See Section 5.131 [easy:SetOpt_FTP_FileMethod], page 104, for details.

INPUTS
none

5.396 easy:UnsetOpt_FTPPort

NAME
easy:UnsetOpt_FTPPort – make FTP transfer active

SYNOPSIS
`easy:UnsetOpt_FTPPort()`

FUNCTION
See Section 5.132 [easy:SetOpt_FTPPort], page 105, for details.

INPUTS
none

5.397 easy:UnsetOpt_FTP_Response_Timeout

NAME
easy:UnsetOpt_FTP_Response_Timeout – time allowed to wait for FTP response

SYNOPSIS
`easy:UnsetOpt_FTP_Response_Timeout()`

FUNCTION
See Section 5.133 [easy:SetOpt_FTP_Response_Timeout], page 105, for details.

INPUTS
none

5.398 easy:UnsetOpt_FTP_Skip_PASV_IP

NAME
easy:UnsetOpt_FTP_Skip_PASV_IP – ignore the IP address in the PASV response

SYNOPSIS
`easy:UnsetOpt_FTP_Skip_PASV_IP()`

FUNCTION
See Section 5.134 [easy:SetOpt_FTP_Skip_PASV_IP], page 106, for details.

INPUTS
none

5.399 easy:UnsetOpt_FTPSSLAuth

NAME

easy:UnsetOpt_FTPSSLAuth – get order in which to attempt TLS vs SSL when using FTP

SYNOPSIS

`easy:UnsetOpt_FTPSSLAuth()`

FUNCTION

See Section 5.135 [easy:SetOpt_FTPSSLAuth], page 106, for details.

INPUTS

none

5.400 easy:UnsetOpt_FTP_SSL_CCC

NAME

easy:UnsetOpt_FTP_SSL_CCC – switch off SSL again with FTP after auth

SYNOPSIS

`easy:UnsetOpt_FTP_SSL_CCC()`

FUNCTION

See Section 5.136 [easy:SetOpt_FTP_SSL_CCC], page 107, for details.

INPUTS

none

5.401 easy:UnsetOpt_FTP_Use_Eprt

NAME

easy:UnsetOpt_FTP_Use_Eprt – enable/disable use of EPRT with FTP

SYNOPSIS

`easy:UnsetOpt_FTP_Use_Eprt()`

FUNCTION

See Section 5.137 [easy:SetOpt_FTP_Use_Eprt], page 107, for details.

INPUTS

none

5.402 easy:UnsetOpt_FTP_Use_Epsv

NAME

easy:UnsetOpt_FTP_Use_Epsv – enable/disable use of EPSV

SYNOPSIS

`easy:UnsetOpt_FTP_Use_Epsv()`

FUNCTION
See Section 5.138 [easy:SetOpt_FTP_Use_Epsv], page 107, for details.

INPUTS
none

5.403 easy:UnsetOpt_FTP_Use_Pret

NAME
easy:UnsetOpt_FTP_Use_Pret – enable the PRET command

SYNOPSIS
`easy:UnsetOpt_FTP_Use_Pret()`

FUNCTION
See Section 5.139 [easy:SetOpt_FTP_Use_Pret], page 108, for details.

INPUTS
none

5.404 easy:UnsetOpt_GSSAPI_Delegation

NAME
easy:UnsetOpt_GSSAPI_Delegation – get allowed GSS-API delegation

SYNOPSIS
`easy:UnsetOpt_GSSAPI_Delegation()`

FUNCTION
See Section 5.140 [easy:SetOpt_GSSAPI_Delegation], page 108, for details.

INPUTS
none

5.405 easy:UnsetOpt_Happy_Eyeballs_Timeout_MS

NAME
easy:UnsetOpt_Happy_Eyeballs_Timeout_MS – head start for IPv6 for happy eyeballs (V2.0)

SYNOPSIS
`easy:UnsetOpt_Happy_Eyeballs_Timeout_MS()`

FUNCTION
See Section 5.141 [easy:SetOpt_Happy_Eyeballs_Timeout_MS], page 109, for details.

INPUTS
none

5.406 easy:UnsetOpt_HAProxyProtocol

NAME
easy:UnsetOpt_HAProxyProtocol – send HAProxy PROXY protocol v1 header (V2.0)

SYNOPSIS
`easy:UnsetOpt_HAProxyProtocol()`

FUNCTION
See Section 5.142 [easy:SetOpt_HAProxyProtocol], page 109, for details.

INPUTS
none

5.407 easy:UnsetOpt_Header

NAME
easy:UnsetOpt_Header – pass headers to the data stream

SYNOPSIS
`easy:UnsetOpt_Header()`

FUNCTION
See Section 5.143 [easy:SetOpt_Header], page 109, for details.

INPUTS
none

5.408 easy:UnsetOpt_HeaderFunction

NAME
easy:UnsetOpt_HeaderFunction – callback that receives header data

SYNOPSIS
`easy:UnsetOpt_HeaderFunction()`

FUNCTION
See Section 5.144 [easy:SetOpt_HeaderFunction], page 110, for details.

INPUTS
none

5.409 easy:UnsetOpt_HeaderOpt

NAME
easy:UnsetOpt_HeaderOpt – get how to send HTTP headers

SYNOPSIS
`easy:UnsetOpt_HeaderOpt()`

FUNCTION
See Section 5.145 [easy:SetOpt_HeaderOpt], page 111, for details.

INPUTS
none

5.410 easy:UnsetOpt_HSTS

NAME
easy:UnsetOpt_HSTS – HSTS cache file name (V2.0)

SYNOPSIS
`easy:UnsetOpt_HSTS()`

FUNCTION
See Section 5.146 [easy:SetOpt_HSTS], page 111, for details.

INPUTS
none

5.411 easy:UnsetOpt_HSTS_Ctrl

NAME
easy:UnsetOpt_HSTS_Ctrl – control HSTS behavior (V2.0)

SYNOPSIS
`easy:UnsetOpt_HSTS_Ctrl()`

FUNCTION
See Section 5.147 [easy:SetOpt_HSTS_Ctrl], page 112, for details.

INPUTS
none

5.412 easy:UnsetOpt_HSTSReadFunction

NAME
easy:UnsetOpt_HSTSReadFunction – read callback for HSTS hosts (V2.0)

SYNOPSIS
`easy:UnsetOpt_HSTSReadFunction()`

FUNCTION
See Section 5.148 [easy:SetOpt_HSTSReadFunction], page 113, for details.

INPUTS
none

5.413 easy:UnsetOpt_HSTSWriteFunction

NAME
easy:UnsetOpt_HSTSWriteFunction – write callback for HSTS hosts (V2.0)

SYNOPSIS
`easy:UnsetOpt_HSTSWriteFunction()`

FUNCTION
See Section 5.149 [easy:SetOpt_HSTSWriteFunction], page 113, for details.

INPUTS
none

5.414 easy:UnsetOpt_HTTP09_Allowed

NAME
easy:UnsetOpt_HTTP09_Allowed – allow HTTP/0.9 response (V2.0)

SYNOPSIS
`easy:UnsetOpt_HTTP09_Allowed()`

FUNCTION
See Section 5.150 [easy:SetOpt_HTTP09_Allowed], page 114, for details.

INPUTS
none

5.415 easy:UnsetOpt_HTTP200Aliases

NAME
easy:UnsetOpt_HTTP200Aliases – specify alternative matches for HTTP 200 OK

SYNOPSIS
`easy:UnsetOpt_HTTP200Aliases()`

FUNCTION
See Section 5.151 [easy:SetOpt_HTTP200Aliases], page 115, for details.

INPUTS
none

5.416 easy:UnsetOpt_HTTPAuth

NAME
easy:UnsetOpt_HTTPAuth – get HTTP server authentication methods to try

SYNOPSIS
`easy:UnsetOpt_HTTPAuth()`

FUNCTION
See Section 5.152 [easy:SetOpt_HTTPAuth], page 115, for details.

INPUTS
none

5.417 easy:UnsetOpt_HTTP_Content_Decoding

NAME
easy:UnsetOpt_HTTP_Content_Decoding – enable/disable HTTP content decoding

SYNOPSIS
`easy:UnsetOpt_HTTP_Content_Decoding()`

FUNCTION
See Section 5.153 [easy:SetOpt_HTTP_Content_Decoding], page 117, for details.

INPUTS
none

5.418 easy:UnsetOpt_HTTPGet

NAME
easy:UnsetOpt_HTTPGet – ask for an HTTP GET request

SYNOPSIS
`easy:UnsetOpt_HTTPGet()`

FUNCTION
See Section 5.154 [easy:SetOpt_HTTPGet], page 117, for details.

INPUTS
none

5.419 easy:UnsetOpt_HTTPHeader

NAME
easy:UnsetOpt_HTTPHeader – get custom HTTP headers

SYNOPSIS
`easy:UnsetOpt_HTTPHeader()`

FUNCTION
See Section 5.155 [easy:SetOpt_HTTPHeader], page 118, for details.

INPUTS
none

5.420 easy:UnsetOpt_HTTPPost

NAME
easy:UnsetOpt_HTTPPost – specify the multipart formpost content

SYNOPSIS
`easy:UnsetOpt_HTTPPost()`

FUNCTION
See Section 5.156 [easy:SetOpt_HTTPPost], page 118, for details.

INPUTS
none

5.421 easy:UnsetOpt_HTTPProxyTunnel

NAME
easy:UnsetOpt_HTTPProxyTunnel – tunnel through HTTP proxy

SYNOPSIS
`easy:UnsetOpt_HTTPProxyTunnel()`

FUNCTION
See Section 5.157 [easy:SetOpt_HTTPProxyTunnel], page 119, for details.

INPUTS
none

5.422 easy:UnsetOpt_HTTP_Transfer_Decoding

NAME
easy:UnsetOpt_HTTP_Transfer_Decoding – enable/disable HTTP transfer decoding

SYNOPSIS
`easy:UnsetOpt_HTTP_Transfer_Decoding()`

FUNCTION
See Section 5.158 [easy:SetOpt_HTTP_Transfer_Decoding], page 119, for details.

INPUTS
none

5.423 easy:UnsetOpt_HTTP_Version

NAME
easy:UnsetOpt_HTTP_Version – specify HTTP protocol version to use

SYNOPSIS
`easy:UnsetOpt_HTTP_Version()`

FUNCTION
See Section 5.159 [easy:SetOpt_HTTP_Version], page 120, for details.

INPUTS
none

5.424 easy:UnsetOpt_Ignore_Content_Length

NAME
easy:UnsetOpt_Ignore_Content_Length – ignore content length

SYNOPSIS
`easy:UnsetOpt_Ignore_Content_Length()`

FUNCTION
See Section 5.160 [easy:SetOpt_Ignore_Content_Length], page 121, for details.

INPUTS
none

5.425 easy:UnsetOpt_InFileSize

NAME
easy:UnsetOpt_InFileSize – get size of the input file to send off

SYNOPSIS
`easy:UnsetOpt_InFileSize()`

FUNCTION
See Section 5.161 [easy:SetOpt_InFileSize], page 121, for details.

INPUTS
none

5.426 easy:UnsetOpt_InFileSize_Large

NAME
easy:UnsetOpt_InFileSize_Large – get size of the input file to send off

SYNOPSIS
`easy:UnsetOpt_InFileSize_Large()`

FUNCTION
See Section 5.162 [easy:SetOpt_InFileSize_Large], page 122, for details.

INPUTS
none

5.427 easy:UnsetOpt_Interface

NAME
easy:UnsetOpt_Interface – source interface for outgoing traffic

SYNOPSIS
`easy:UnsetOpt_Interface()`

FUNCTION
See Section 5.163 [easy:SetOpt_Interface], page 122, for details.

INPUTS
none

5.428 easy:UnsetOpt_IPResolve

NAME
easy:UnsetOpt_IPResolve – specify which IP protocol version to use

SYNOPSIS
`easy:UnsetOpt_IPResolve()`

FUNCTION
See Section 5.164 [easy:SetOpt_IPResolve], page 123, for details.

INPUTS
none

5.429 easy:UnsetOpt_IssuerCert

NAME
easy:UnsetOpt_IssuerCert – issuer SSL certificate filename

SYNOPSIS
`easy:UnsetOpt_IssuerCert()`

FUNCTION
See Section 5.165 [easy:SetOpt_IssuerCert], page 123, for details.

INPUTS
none

5.430 easy:UnsetOpt_IssuerCert_Blob

NAME
easy:UnsetOpt_IssuerCert_Blob – issuer SSL certificate from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_IssuerCert_Blob()`

FUNCTION
See Section 5.166 [easy:SetOpt_IssuerCert_Blob], page 124, for details.

INPUTS
none

5.431 easy:UnsetOpt_Keep_Sending_On_Error

NAME
easy:UnsetOpt_Keep_Sending_On_Error – keep sending on early HTTP response >= 300

SYNOPSIS
`easy:UnsetOpt_Keep_Sending_On_Error()`

FUNCTION
See Section 5.167 [easy:SetOpt_Keep_Sending_On_Error], page 124, for details.

INPUTS
none

5.432 easy:UnsetOpt_KeyPasswd

NAME
easy:UnsetOpt_KeyPasswd – get passphrase to private key

SYNOPSIS
`easy:UnsetOpt_KeyPasswd()`

FUNCTION
See Section 5.168 [easy:SetOpt_KeyPasswd], page 125, for details.

INPUTS
none

5.433 easy:UnsetOpt_KRBLevel

NAME
easy:UnsetOpt_KRBLevel – get FTP kerberos security level

SYNOPSIS
`easy:UnsetOpt_KRBLevel()`

FUNCTION
See Section 5.169 [easy:SetOpt_KRBLevel], page 125, for details.

INPUTS
none

5.434 easy:UnsetOpt_LocalPort

NAME
easy:UnsetOpt_LocalPort – get local port number to use for socket

SYNOPSIS
`easy:UnsetOpt_LocalPort()`

FUNCTION
See Section 5.170 [easy:SetOpt_LocalPort], page 125, for details.

INPUTS
none

5.435 easy:UnsetOpt_LocalPortRange

NAME
easy:UnsetOpt_LocalPortRange – number of additional local ports to try

SYNOPSIS
`easy:UnsetOpt_LocalPortRange()`

FUNCTION
See Section 5.171 [easy:SetOpt_LocalPortRange], page 126, for details.

INPUTS
none

5.436 easy:UnsetOpt_Login_Options

NAME
easy:UnsetOpt_Login_Options – get login options

SYNOPSIS
`easy:UnsetOpt_Login_Options()`

FUNCTION
See Section 5.172 [easy:SetOpt_Login_Options], page 126, for details.

INPUTS
none

5.437 easy:UnsetOpt_Low_Speed_Limit

NAME
easy:UnsetOpt_Low_Speed_Limit – get low speed limit in bytes per second

SYNOPSIS
`easy:UnsetOpt_Low_Speed_Limit()`

FUNCTION
See Section 5.173 [easy:SetOpt_Low_Speed_Limit], page 126, for details.

INPUTS
none

5.438 easy:UnsetOpt_Low_Speed_Time

NAME
easy:UnsetOpt_Low_Speed_Time – get low speed limit time period

SYNOPSIS
`easy:UnsetOpt_Low_Speed_Time()`

FUNCTION
See Section 5.174 [easy:SetOpt_Low_Speed_Time], page 127, for details.

INPUTS
none

5.439 easy:UnsetOpt_Mail_Auth

NAME
easy:UnsetOpt_Mail_Auth – SMTP authentication address

SYNOPSIS
`easy:UnsetOpt_Mail_Auth()`

FUNCTION
See Section 5.175 [easy:SetOpt_Mail_Auth], page 127, for details.

INPUTS
none

5.440 easy:UnsetOpt_Mail_From

NAME
easy:UnsetOpt_Mail_From – SMTP sender address

SYNOPSIS
`easy:UnsetOpt_Mail_From()`

FUNCTION
See Section 5.176 [easy:SetOpt_Mail_From], page 128, for details.

INPUTS
none

5.441 easy:UnsetOpt_Mail_RCPT

NAME
easy:UnsetOpt_Mail_RCPT – list of SMTP mail recipients

SYNOPSIS
`easy:UnsetOpt_Mail_RCPT()`

FUNCTION
See Section 5.177 [easy:SetOpt_Mail_RCPT], page 128, for details.

INPUTS
none

5.442 easy:UnsetOpt_Mail_RCPT_AllowFails

NAME
easy:UnsetOpt_Mail_RCPT_AllowFails – allow RCPT TO command to fail for some recipients (V2.0)

SYNOPSIS
`easy:UnsetOpt_Mail_RCPT_AllowFails()`

FUNCTION
See Section 5.178 [easy:SetOpt_Mail_RCPT_AllowFails], page 128, for details.

INPUTS
none

5.443 easy:UnsetOpt_MaxAge_Conn

NAME
easy:UnsetOpt_MaxAge_Conn – max idle time allowed for reusing a connection (V2.0)

SYNOPSIS
`easy:UnsetOpt_MaxAge_Conn()`

FUNCTION
See Section 5.179 [easy:SetOpt_MaxAge_Conn], page 129, for details.

INPUTS
none

5.444 easy:UnsetOpt_MaxConnects

NAME
easy:UnsetOpt_MaxConnects – maximum connection cache size

SYNOPSIS
`easy:UnsetOpt_MaxConnects()`

FUNCTION
See Section 5.180 [easy:SetOpt_MaxConnects], page 129, for details.

INPUTS
none

5.445 easy:UnsetOpt_MaxFileSize

NAME
easy:UnsetOpt_MaxFileSize – maximum file size allowed to download

SYNOPSIS
```
easy:UnsetOpt_MaxFileSize()
```

FUNCTION
See Section 5.181 [easy:SetOpt_MaxFileSize], page 130, for details.

INPUTS
none

5.446 easy:UnsetOpt_MaxFileSize_Large

NAME
easy:UnsetOpt_MaxFileSize_Large – maximum file size allowed to download

SYNOPSIS
```
easy:UnsetOpt_MaxFileSize_Large()
```

FUNCTION
See Section 5.182 [easy:SetOpt_MaxFileSize_Large], page 130, for details.

INPUTS
none

5.447 easy:UnsetOpt_MaxLifeTime_Conn

NAME
easy:UnsetOpt_MaxLifeTime_Conn – max lifetime (since creation) allowed for reusing a connection (V2.0)

SYNOPSIS
```
easy:UnsetOpt_MaxLifeTime_Conn()
```

FUNCTION
See Section 5.183 [easy:SetOpt_MaxLifeTime_Conn], page 131, for details.

INPUTS
none

5.448 easy:UnsetOpt_Max_Recv_Speed_Large

NAME
easy:UnsetOpt_Max_Recv_Speed_Large – rate limit data download speed

SYNOPSIS
`easy:UnsetOpt_Max_Recv_Speed_Large()`

FUNCTION
See Section 5.184 [easy:SetOpt_Max_Recv_Speed_Large], page 131, for details.

INPUTS
none

5.449 easy:UnsetOpt_MaxRedirs

NAME
easy:UnsetOpt_MaxRedirs – maximum number of redirects allowed

SYNOPSIS
`easy:UnsetOpt_MaxRedirs()`

FUNCTION
See Section 5.185 [easy:SetOpt_MaxRedirs], page 132, for details.

INPUTS
none

5.450 easy:UnsetOpt_Max_Send_Speed_Large

NAME
easy:UnsetOpt_Max_Send_Speed_Large – rate limit data upload speed

SYNOPSIS
`easy:UnsetOpt_Max_Send_Speed_Large()`

FUNCTION
See Section 5.186 [easy:SetOpt_Max_Send_Speed_Large], page 132, for details.

INPUTS
none

5.451 easy:UnsetOpt_MIME_Options

NAME
easy:UnsetOpt_MIME_Options – get MIME option flags (V2.0)

SYNOPSIS
`easy:UnsetOpt_MIME_Options()`

FUNCTION
See Section 5.187 [easy:SetOpt_MIME_Options], page 132, for details.

INPUTS
none

5.452 easy:UnsetOpt_MIMEPost

NAME
easy:UnsetOpt_MIMEPost – send data from mime structure (V2.0)

SYNOPSIS
`easy:UnsetOpt_MIMEPost()`

FUNCTION
See Section 5.188 [easy:SetOpt_MIMEPost], page 133, for details.

INPUTS
none

5.453 easy:UnsetOpt_Netrc

NAME
easy:UnsetOpt_Netrc – request that .netrc is used

SYNOPSIS
`easy:UnsetOpt_Netrc()`

FUNCTION
See Section 5.189 [easy:SetOpt_Netrc], page 133, for details.

INPUTS
none

5.454 easy:UnsetOpt_Netrc_File

NAME
easy:UnsetOpt_Netrc_File – file name to read .netrc info from

SYNOPSIS
`easy:UnsetOpt_Netrc_File()`

FUNCTION
See Section 5.190 [easy:SetOpt_Netrc_File], page 134, for details.

INPUTS
none

5.455 easy:UnsetOpt_New_Directory_Perms

NAME
easy:UnsetOpt_New_Directory_Perms – permissions for remotely created directories

SYNOPSIS
`easy:UnsetOpt_New_Directory_Perms()`

FUNCTION
See Section 5.191 [easy:SetOpt_New_Directory_Perms], page 134, for details.

INPUTS
none

5.456 easy:UnsetOpt_New_File_Perms

NAME
easy:UnsetOpt_New_File_Perms – permissions for remotely created files

SYNOPSIS
`easy:UnsetOpt_New_File_Perms()`

FUNCTION
See Section 5.192 [easy:SetOpt_New_File_Perms], page 135, for details.

INPUTS
none

5.457 easy:UnsetOpt_Nobody

NAME
easy:UnsetOpt_Nobody – do the download request without getting the body

SYNOPSIS
`easy:UnsetOpt_Nobody()`

FUNCTION
See Section 5.193 [easy:SetOpt_Nobody], page 135, for details.

INPUTS
none

5.458 easy:UnsetOpt_NoProgress

NAME
easy:UnsetOpt_NoProgress – switch off the progress meter

SYNOPSIS
`easy:UnsetOpt_NoProgress()`

FUNCTION
See Section 5.194 [easy:SetOpt_NoProgress], page 136, for details.

INPUTS
none

5.459 easy:UnsetOpt_NoProxy

NAME
easy:UnsetOpt_NoProxy – disable proxy use for specific hosts

SYNOPSIS
`easy:UnsetOpt_NoProxy()`

FUNCTION
See Section 5.195 [easy:SetOpt_NoProxy], page 136, for details.

INPUTS
none

5.460 easy:UnsetOpt_NoSignal

NAME
easy:UnsetOpt_NoSignal – skip all signal handling

SYNOPSIS
`easy:UnsetOpt_NoSignal()`

FUNCTION
See Section 5.196 [easy:SetOpt_NoSignal], page 136, for details.

INPUTS
none

5.461 easy:UnsetOpt_Password

NAME
easy:UnsetOpt_Password – password to use in authentication

SYNOPSIS
`easy:UnsetOpt_Password()`

FUNCTION
See Section 5.197 [easy:SetOpt_Password], page 137, for details.

INPUTS
none

5.462 easy:UnsetOpt_Path_As_Is

NAME
easy:UnsetOpt_Path_As_Is – do not handle dot dot sequences

SYNOPSIS
`easy:UnsetOpt_Path_As_Is()`

FUNCTION
See Section 5.198 [easy:SetOpt_Path_As_Is], page 137, for details.

INPUTS
none

5.463 easy:UnsetOpt_PinnedPublicKey

NAME
easy:UnsetOpt_PinnedPublicKey – get pinned public key

SYNOPSIS
`easy:UnsetOpt_PinnedPublicKey()`

FUNCTION
See Section 5.199 [easy:SetOpt_PinnedPublicKey], page 138, for details.

INPUTS
none

5.464 easy:UnsetOpt_PipeWait

NAME
easy:UnsetOpt_PipeWait – wait for pipelining/multiplexing

SYNOPSIS
`easy:UnsetOpt_PipeWait()`

FUNCTION
See Section 5.200 [easy:SetOpt_PipeWait], page 138, for details.

INPUTS
none

5.465 easy:UnsetOpt_Port

NAME
easy:UnsetOpt_Port – get remote port number to work with

SYNOPSIS
`easy:UnsetOpt_Port()`

FUNCTION
See Section 5.201 [easy:SetOpt_Port], page 139, for details.

INPUTS
none

5.466 easy:UnsetOpt_Post

NAME
easy:UnsetOpt_Post – request an HTTP POST

SYNOPSIS
`easy:UnsetOpt_Post()`

FUNCTION
See Section 5.202 [easy:SetOpt_Post], page 139, for details.

INPUTS
none

5.467 easy:UnsetOpt_PostFields

NAME
easy:UnsetOpt_PostFields – specify data to POST to server

SYNOPSIS
`easy:UnsetOpt_PostFields()`

FUNCTION
See Section 5.203 [easy:SetOpt_PostFields], page 140, for details.

INPUTS
none

5.468 easy:UnsetOpt_PostQuote

NAME
easy:UnsetOpt_PostQuote – (S)FTP commands to run after the transfer

SYNOPSIS
`easy:UnsetOpt_PostQuote()`

FUNCTION
See Section 5.204 [easy:SetOpt_PostQuote], page 141, for details.

INPUTS
none

5.469 easy:UnsetOpt_PostRedir

NAME
easy:UnsetOpt_PostRedir – how to act on an HTTP POST redirect

SYNOPSIS
`easy:UnsetOpt_PostRedir()`

FUNCTION
See Section 5.205 [easy:SetOpt_PostRedir], page 141, for details.

INPUTS
none

5.470 easy:UnsetOpt_Pre_Proxy

NAME
easy:UnsetOpt_Pre_Proxy – get pre-proxy to use

SYNOPSIS
`easy:UnsetOpt_Pre_Proxy()`

FUNCTION
See Section 5.206 [easy:SetOpt_Pre_Proxy], page 142, for details.

INPUTS
none

5.471 easy:UnsetOpt_Prequote

NAME
easy:UnsetOpt_Prequote – commands to run before an FTP transfer

SYNOPSIS
`easy:UnsetOpt_Prequote()`

FUNCTION
See Section 5.207 [easy:SetOpt_Prequote], page 142, for details.

INPUTS
none

5.472 easy:UnsetOpt_PreReqFunction

NAME
easy:UnsetOpt_PreReqFunction – user callback called when a connection has been (V2.0)

SYNOPSIS
`easy:UnsetOpt_PreReqFunction()`

FUNCTION
See Section 5.208 [easy:SetOpt_PreReqFunction], page 143, for details.

INPUTS
none

5.473 easy:UnsetOpt_ProgressFunction

NAME
easy:UnsetOpt_ProgressFunction – callback to progress meter function

SYNOPSIS
`easy:UnsetOpt_ProgressFunction()`

FUNCTION
See Section 5.209 [easy:SetOpt_ProgressFunction], page 144, for details.

INPUTS
none

5.474 easy:UnsetOpt_Protocols

NAME
easy:UnsetOpt_Protocols – get allowed protocols

SYNOPSIS
`easy:UnsetOpt_Protocols()`

FUNCTION
See Section 5.210 [easy:SetOpt_Protocols], page 144, for details.

INPUTS
none

5.475 easy:UnsetOpt_Protocols_Str

NAME
easy:UnsetOpt_Protocols_Str – allowed protocols (V2.0)

SYNOPSIS
`easy:UnsetOpt_Protocols_Str()`

FUNCTION
See Section 5.211 [easy:SetOpt_Protocols_Str], page 145, for details.

INPUTS
none

5.476 easy:UnsetOpt_Proxy

NAME
easy:UnsetOpt_Proxy – get proxy to use

SYNOPSIS
`easy:UnsetOpt_Proxy()`

FUNCTION
See Section 5.212 [easy:SetOpt_Proxy], page 147, for details.

INPUTS
none

5.477 easy:UnsetOpt_ProxyAuth

NAME
easy:UnsetOpt_ProxyAuth – get HTTP proxy authentication methods to try

SYNOPSIS
`easy:UnsetOpt_ProxyAuth()`

FUNCTION
See Section 5.213 [easy:SetOpt_ProxyAuth], page 148, for details.

INPUTS
none

5.478 easy:UnsetOpt_Proxy_CAInfo

NAME
easy:UnsetOpt_Proxy_CAInfo – path to proxy Certificate Authority (CA) bundle

SYNOPSIS
`easy:UnsetOpt_Proxy_CAInfo()`

FUNCTION
See Section 5.214 [easy:SetOpt_Proxy_CAInfo], page 148, for details.

INPUTS
none

5.479 easy:UnsetOpt_Proxy_CAInfo_Blob

NAME
easy:UnsetOpt_Proxy_CAInfo_Blob – proxy Certificate Authority (CA) bundle in PEM format (V2.0)

SYNOPSIS
`easy:UnsetOpt_Proxy_CAInfo_Blob()`

FUNCTION
See Section 5.215 [easy:SetOpt_Proxy_CAInfo_Blob], page 149, for details.

INPUTS
none

5.480 easy:UnsetOpt_Proxy_CAPath

NAME
easy:UnsetOpt_Proxy_CAPath – specify directory holding proxy CA certificates

SYNOPSIS
`easy:UnsetOpt_Proxy_CAPath()`

FUNCTION
See Section 5.216 [easy:SetOpt_Proxy_CAPath], page 149, for details.

INPUTS
none

5.481 easy:UnsetOpt_Proxy_CRLFile

NAME
easy:UnsetOpt_Proxy_CRLFile – specify a proxy Certificate Revocation List file

SYNOPSIS
`easy:UnsetOpt_Proxy_CRLFile()`

FUNCTION
See Section 5.217 [easy:SetOpt_Proxy_CRLFile], page 150, for details.

INPUTS
none

5.482 easy:UnsetOpt_ProxyHeader

NAME
easy:UnsetOpt_ProxyHeader – custom HTTP headers to pass to proxy

SYNOPSIS
`easy:UnsetOpt_ProxyHeader()`

FUNCTION
See Section 5.218 [easy:SetOpt_ProxyHeader], page 150, for details.

INPUTS
none

5.483 easy:UnsetOpt_Proxy_IssuerCert

NAME
easy:UnsetOpt_Proxy_IssuerCert – proxy issuer SSL certificate filename (V2.0)

SYNOPSIS
`easy:UnsetOpt_Proxy_IssuerCert()`

FUNCTION
See Section 5.219 [easy:SetOpt_Proxy_IssuerCert], page 151, for details.

INPUTS
none

5.484 easy:UnsetOpt_Proxy_IssuerCert_Blob

NAME
easy:UnsetOpt_Proxy_IssuerCert_Blob – proxy issuer SSL certificate from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_Proxy_IssuerCert_Blob()`

FUNCTION
See Section 5.220 [easy:SetOpt_Proxy_IssuerCert_Blob], page 151, for details.

INPUTS
none

5.485 easy:UnsetOpt_Proxy_KeyPasswd

NAME
easy:UnsetOpt_Proxy_KeyPasswd – get passphrase to proxy private key

SYNOPSIS
`easy:UnsetOpt_Proxy_KeyPasswd()`

FUNCTION
See Section 5.221 [easy:SetOpt_Proxy_KeyPasswd], page 152, for details.

INPUTS
none

5.486 easy:UnsetOpt_ProxyPassword

NAME
easy:UnsetOpt_ProxyPassword – password to use with proxy authentication

SYNOPSIS
`easy:UnsetOpt_ProxyPassword()`

FUNCTION
See Section 5.222 [easy:SetOpt_ProxyPassword], page 152, for details.

INPUTS
none

5.487 easy:UnsetOpt_Proxy_PinnedPublicKey

NAME
easy:UnsetOpt_Proxy_PinnedPublicKey – get pinned public key for https proxy

SYNOPSIS
`easy:UnsetOpt_Proxy_PinnedPublicKey()`

FUNCTION
See Section 5.223 [easy:SetOpt_Proxy_PinnedPublicKey], page 152, for details.

INPUTS
none

5.488 easy:UnsetOpt_ProxyPort

NAME
easy:UnsetOpt_ProxyPort – port number the proxy listens on

SYNOPSIS
`easy:UnsetOpt_ProxyPort()`

FUNCTION
See Section 5.224 [easy:SetOpt_ProxyPort], page 153, for details.

INPUTS
none

5.489 easy:UnsetOpt_Proxy_Service_Name

NAME
easy:UnsetOpt_Proxy_Service_Name – proxy authentication service name

SYNOPSIS
`easy:UnsetOpt_Proxy_Service_Name()`

FUNCTION
See Section 5.225 [easy:SetOpt_Proxy_Service_Name], page 153, for details.

INPUTS
none

5.490 easy:UnsetOpt_Proxy_SSLCert

NAME
easy:UnsetOpt_Proxy_SSLCert – get SSL proxy client certificate

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLCert()`

FUNCTION
See Section 5.226 [easy:SetOpt_Proxy_SSLCert], page 154, for details.

INPUTS
none

5.491 easy:UnsetOpt_Proxy_SSLCert_Blob

NAME
easy:UnsetOpt_Proxy_SSLCert_Blob – SSL proxy client certificate from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLCert_Blob()`

FUNCTION
See Section 5.227 [easy:SetOpt_Proxy_SSLCert_Blob], page 154, for details.

INPUTS
none

5.492 easy:UnsetOpt_Proxy_SSLCertType

NAME
easy:UnsetOpt_Proxy_SSLCertType – specify type of the proxy client SSL certificate

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLCertType()`

FUNCTION
See Section 5.228 [easy:SetOpt_Proxy_SSLCertType], page 154, for details.

INPUTS
none

5.493 easy:UnsetOpt_Proxy_SSL_Cipher_List

NAME
easy:UnsetOpt_Proxy_SSL_Cipher_List – specify ciphers to use for proxy TLS

SYNOPSIS
`easy:UnsetOpt_Proxy_SSL_Cipher_List()`

FUNCTION
See Section 5.229 [easy:SetOpt_Proxy_SSL_Cipher_List], page 155, for details.

INPUTS
none

5.494 easy:UnsetOpt_Proxy_SSLKey

NAME
easy:UnsetOpt_Proxy_SSLKey – specify private keyfile for TLS and SSL proxy client cert

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLKey()`

FUNCTION
See Section 5.230 [easy:SetOpt_Proxy_SSLKey], page 155, for details.

INPUTS
none

5.495 easy:UnsetOpt_Proxy_SSLKey_Blob

NAME
easy:UnsetOpt_Proxy_SSLKey_Blob – private key for proxy cert from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLKey_Blob()`

FUNCTION
See Section 5.231 [easy:SetOpt_Proxy_SSLKey_Blob], page 156, for details.

INPUTS
none

5.496 easy:UnsetOpt_Proxy_SSLKeyType

NAME
easy:UnsetOpt_Proxy_SSLKeyType – get type of the proxy private key file

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLKeyType()`

FUNCTION
See Section 5.232 [easy:SetOpt_Proxy_SSLKeyType], page 156, for details.

INPUTS
none

5.497 easy:UnsetOpt_Proxy_SSL_Options

NAME
easy:UnsetOpt_Proxy_SSL_Options – get proxy SSL behavior options

SYNOPSIS
`easy:UnsetOpt_Proxy_SSL_Options()`

FUNCTION
See Section 5.233 [easy:SetOpt_Proxy_SSL_Options], page 156, for details.

INPUTS
none

5.498 easy:UnsetOpt_Proxy_SSL_VerifyHost

NAME
easy:UnsetOpt_Proxy_SSL_VerifyHost – verify the proxy certificate's name against host

SYNOPSIS
`easy:UnsetOpt_Proxy_SSL_VerifyHost()`

FUNCTION
See Section 5.234 [easy:SetOpt_Proxy_SSL_VerifyHost], page 157, for details.

INPUTS
none

5.499 easy:UnsetOpt_Proxy_SSL_VerifyPeer

NAME
easy:UnsetOpt_Proxy_SSL_VerifyPeer – verify the proxy's SSL certificate

SYNOPSIS
`easy:UnsetOpt_Proxy_SSL_VerifyPeer()`

FUNCTION
See Section 5.235 [easy:SetOpt_Proxy_SSL_VerifyPeer], page 158, for details.

INPUTS
none

5.500 easy:UnsetOpt_Proxy_SSLVersion

NAME
easy:UnsetOpt_Proxy_SSLVersion – get preferred proxy TLS/SSL version

SYNOPSIS
`easy:UnsetOpt_Proxy_SSLVersion()`

FUNCTION
See Section 5.236 [easy:SetOpt_Proxy_SSLVersion], page 158, for details.

INPUTS
none

5.501 easy:UnsetOpt_Proxy_TLSAuth_Password

NAME
easy:UnsetOpt_Proxy_TLSAuth_Password – password to use for proxy TLS authentication

SYNOPSIS
`easy:UnsetOpt_Proxy_TLSAuth_Password()`

FUNCTION
See Section 5.237 [easy:SetOpt_Proxy_TLSAuth_Password], page 160, for details.

INPUTS
none

5.502 easy:UnsetOpt_Proxy_TLSAuth_Type

NAME
easy:UnsetOpt_Proxy_TLSAuth_Type – get proxy TLS authentication methods

SYNOPSIS
`easy:UnsetOpt_Proxy_TLSAuth_Type()`

FUNCTION
See Section 5.238 [easy:SetOpt_Proxy_TLSAuth_Type], page 160, for details.

INPUTS
none

5.503 easy:UnsetOpt_Proxy_TLSAuth_UserName

NAME
easy:UnsetOpt_Proxy_TLSAuth_UserName – user name to use for proxy TLS authentication

SYNOPSIS
`easy:UnsetOpt_Proxy_TLSAuth_UserName()`

FUNCTION
See Section 5.239 [easy:SetOpt_Proxy_TLSAuth_UserName], page 160, for details.

INPUTS
none

5.504 easy:UnsetOpt_Proxy_Transfer_Mode

NAME
easy:UnsetOpt_Proxy_Transfer_Mode – append FTP transfer mode to URL for proxy

SYNOPSIS
`easy:UnsetOpt_Proxy_Transfer_Mode()`

FUNCTION
See Section 5.240 [easy:SetOpt_Proxy_Transfer_Mode], page 161, for details.

INPUTS
none

5.505 easy:UnsetOpt_ProxyType

NAME
easy:UnsetOpt_ProxyType – proxy protocol type

SYNOPSIS
`easy:UnsetOpt_ProxyType()`

FUNCTION
See Section 5.241 [easy:SetOpt_ProxyType], page 161, for details.

INPUTS
none

5.506 easy:UnsetOpt_ProxyUserName

NAME
easy:UnsetOpt_ProxyUserName – user name to use for proxy authentication

SYNOPSIS
`easy:UnsetOpt_ProxyUserName()`

FUNCTION
See Section 5.242 [easy:SetOpt_ProxyUserName], page 162, for details.

INPUTS
none

5.507 easy:UnsetOpt_ProxyUserPwd

NAME
easy:UnsetOpt_ProxyUserPwd – user name and password to use for proxy authentication

SYNOPSIS
`easy:UnsetOpt_ProxyUserPwd()`

FUNCTION
See Section 5.243 [easy:SetOpt_ProxyUserPwd], page 162, for details.

INPUTS
none

5.508 easy:UnsetOpt_Put

NAME
easy:UnsetOpt_Put – make an HTTP PUT request

SYNOPSIS
`easy:UnsetOpt_Put()`

FUNCTION
See Section 5.244 [easy:SetOpt_Put], page 163, for details.

INPUTS
none

5.509 easy:UnsetOpt_Quick_Exit

NAME
easy:UnsetOpt_Quick_Exit – allow to exit quickly (V2.0)

SYNOPSIS
`easy:UnsetOpt_Quick_Exit()`

FUNCTION
See Section 5.245 [easy:SetOpt_Quick_Exit], page 163, for details.

INPUTS
none

5.510 easy:UnsetOpt_Quote

NAME
easy:UnsetOpt_Quote – (S)FTP commands to run before transfer

SYNOPSIS
`easy:UnsetOpt_Quote()`

FUNCTION
See Section 5.246 [easy:SetOpt_Quote], page 163, for details.

INPUTS
none

5.511 easy:UnsetOpt_Random_File

NAME
easy:UnsetOpt_Random_File – specify a source for random data

SYNOPSIS
`easy:UnsetOpt_Random_File()`

FUNCTION
See Section 5.247 [easy:SetOpt_Random_File], page 165, for details.

INPUTS
none

5.512 easy:UnsetOpt_Range

NAME
easy:UnsetOpt_Range – get byte range to request

SYNOPSIS
`easy:UnsetOpt_Range()`

FUNCTION
See Section 5.248 [easy:SetOpt_Range], page 165, for details.

INPUTS
none

5.513 easy:UnsetOpt_ReadFunction

NAME
easy:UnsetOpt_ReadFunction – read callback for data uploads

SYNOPSIS
`easy:UnsetOpt_ReadFunction()`

FUNCTION
See Section 5.249 [easy:SetOpt_ReadFunction], page 165, for details.

INPUTS
none

5.514 easy:UnsetOpt_Redir_Protocols

NAME
easy:UnsetOpt_Redir_Protocols – get protocols allowed to redirect to

SYNOPSIS
`easy:UnsetOpt_Redir_Protocols()`

FUNCTION
See Section 5.250 [easy:SetOpt_Redir_Protocols], page 167, for details.

INPUTS
none

5.515 easy:UnsetOpt_Redir_Protocols_Str

NAME
easy:UnsetOpt_Redir_Protocols_Str – protocols allowed to redirect to (V2.0)

SYNOPSIS
`easy:UnsetOpt_Redir_Protocols_Str()`

FUNCTION
See Section 5.251 [easy:SetOpt_Redir_Protocols_Str], page 168, for details.

INPUTS
none

5.516 easy:UnsetOpt_Referer

NAME
easy:UnsetOpt_Referer – get the HTTP referer header

SYNOPSIS
`easy:UnsetOpt_Referer()`

FUNCTION
See Section 5.252 [easy:SetOpt_Referer], page 169, for details.

INPUTS
none

5.517 easy:UnsetOpt_Request_Target

NAME
easy:UnsetOpt_Request_Target – specify an alternative target for this request

SYNOPSIS
`easy:UnsetOpt_Request_Target()`

FUNCTION
See Section 5.253 [easy:SetOpt_Request_Target], page 169, for details.

INPUTS
none

5.518 easy:UnsetOpt_Resolve

NAME
easy:UnsetOpt_Resolve – provide custom host name to IP address resolves

SYNOPSIS
`easy:UnsetOpt_Resolve()`

FUNCTION
See Section 5.254 [easy:SetOpt_Resolve], page 170, for details.

INPUTS
none

5.519 easy:UnsetOpt_Resolver_Start_Function

NAME
easy:UnsetOpt_Resolver_Start_Function – callback called before a new name resolve is started (V2.0)

SYNOPSIS
`easy:UnsetOpt_Resolver_Start_Function()`

FUNCTION
See Section 5.255 [easy:SetOpt_Resolver_Start_Function], page 170, for details.

INPUTS
none

5.520 easy:UnsetOpt_Resume_From

NAME
easy:UnsetOpt_Resume_From – get a point to resume transfer from

SYNOPSIS
`easy:UnsetOpt_Resume_From()`

FUNCTION
See Section 5.256 [easy:SetOpt_Resume_From], page 171, for details.

INPUTS
none

5.521 easy:UnsetOpt_Resume_From_Large

NAME
easy:UnsetOpt_Resume_From_Large – get a point to resume transfer from

SYNOPSIS
`easy:UnsetOpt_Resume_From_Large()`

FUNCTION
See Section 5.257 [easy:SetOpt_Resume_From_Large], page 171, for details.

INPUTS
none

5.522 easy:UnsetOpt_RTSP_Client_CSeq

NAME
easy:UnsetOpt_RTSP_Client_CSeq – get the RTSP client CSEQ number

SYNOPSIS
`easy:UnsetOpt_RTSP_Client_CSeq()`

FUNCTION
See Section 5.258 [easy:SetOpt_RTSP_Client_CSeq], page 172, for details.

INPUTS
none

5.523 easy:UnsetOpt_RTSP_Request

NAME
easy:UnsetOpt_RTSP_Request – specify RTSP request

SYNOPSIS
`easy:UnsetOpt_RTSP_Request()`

FUNCTION
See Section 5.259 [easy:SetOpt_RTSP_Request], page 172, for details.

INPUTS
none

5.524 easy:UnsetOpt_RTSP_Server_CSeq

NAME
easy:UnsetOpt_RTSP_Server_CSeq – get the RTSP server CSEQ number

SYNOPSIS
`easy:UnsetOpt_RTSP_Server_CSeq()`

FUNCTION
See Section 5.260 [easy:SetOpt_RTSP_Server_CSeq], page 174, for details.

INPUTS
none

5.525 easy:UnsetOpt_RTSP_Session_ID

NAME
easy:UnsetOpt_RTSP_Session_ID – get RTSP session ID

SYNOPSIS
`easy:UnsetOpt_RTSP_Session_ID()`

FUNCTION
See Section 5.261 [easy:SetOpt_RTSP_Session_ID], page 174, for details.

INPUTS
none

5.526 easy:UnsetOpt_RTSP_Stream_URI

NAME
easy:UnsetOpt_RTSP_Stream_URI – get RTSP stream URI

SYNOPSIS
`easy:UnsetOpt_RTSP_Stream_URI()`

FUNCTION
See Section 5.262 [easy:SetOpt_RTSP_Stream_URI], page 174, for details.

INPUTS
none

5.527 easy:UnsetOpt_RTSP_Transport

NAME
easy:UnsetOpt_RTSP_Transport – get RTSP Transport: header

SYNOPSIS
`easy:UnsetOpt_RTSP_Transport()`

FUNCTION
See Section 5.263 [easy:SetOpt_RTSP_Transport], page 175, for details.

INPUTS
none

5.528 easy:UnsetOpt_SASL_AuthZID

NAME
easy:UnsetOpt_SASL_AuthZID – authorization identity (identity to act as) (V2.0)

SYNOPSIS
`easy:UnsetOpt_SASL_AuthZID()`

FUNCTION
See Section 5.264 [easy:SetOpt_SASL_AuthZID], page 175, for details.

INPUTS
none

5.529 easy:UnsetOpt_SASL_IR

NAME
easy:UnsetOpt_SASL_IR – enable sending initial response in first packet

SYNOPSIS
`easy:UnsetOpt_SASL_IR()`

FUNCTION
See Section 5.265 [easy:SetOpt_SASL_IR], page 176, for details.

INPUTS
none

5.530 easy:UnsetOpt_SeekFunction

NAME
easy:UnsetOpt_SeekFunction – user callback for seeking in input stream

SYNOPSIS
`easy:UnsetOpt_SeekFunction()`

FUNCTION
See Section 5.266 [easy:SetOpt_SeekFunction], page 176, for details.

INPUTS
none

5.531 easy:UnsetOpt_Service_Name

NAME
easy:UnsetOpt_Service_Name – authentication service name

SYNOPSIS
`easy:UnsetOpt_Service_Name()`

FUNCTION
See Section 5.267 [easy:SetOpt_Service_Name], page 177, for details.

INPUTS
none

5.532 easy:UnsetOpt_Share

NAME
easy:UnsetOpt_Share – specify share handle to use

SYNOPSIS
`easy:UnsetOpt_Share()`

FUNCTION
See Section 5.268 [easy:SetOpt_Share], page 177, for details.

INPUTS
none

5.533 easy:UnsetOpt_Socks5_Auth

NAME
easy:UnsetOpt_Socks5_Auth – get allowed methods for SOCKS5 proxy authentication

SYNOPSIS
`easy:UnsetOpt_Socks5_Auth()`

FUNCTION
See Section 5.269 [easy:SetOpt_Socks5_Auth], page 178, for details.

INPUTS
none

5.534 easy:UnsetOpt_Socks5_GSSAPI_NEC

NAME
easy:UnsetOpt_Socks5_GSSAPI_NEC – get socks proxy gssapi negotiation protection

SYNOPSIS
`easy:UnsetOpt_Socks5_GSSAPI_NEC()`

FUNCTION
See Section 5.270 [easy:SetOpt_Socks5_GSSAPI_NEC], page 178, for details.

INPUTS
none

5.535 easy:UnsetOpt_Socks5_GSSAPI_Service

NAME
easy:UnsetOpt_Socks5_GSSAPI_Service – SOCKS5 proxy authentication service name

SYNOPSIS
`easy:UnsetOpt_Socks5_GSSAPI_Service()`

FUNCTION
See Section 5.271 [easy:SetOpt_Socks5_GSSAPI_Service], page 179, for details.

INPUTS
none

5.536 easy:UnsetOpt_SSH_Auth_Types

NAME
easy:UnsetOpt_SSH_Auth_Types – get desired auth types for SFTP and SCP

SYNOPSIS
`easy:UnsetOpt_SSH_Auth_Types()`

FUNCTION
See Section 5.272 [easy:SetOpt_SSH_Auth_Types], page 179, for details.

INPUTS
none

5.537 easy:UnsetOpt_SSH_Compression

NAME
easy:UnsetOpt_SSH_Compression – enable SSH compression (V2.0)

SYNOPSIS
`easy:UnsetOpt_SSH_Compression()`

FUNCTION
See Section 5.273 [easy:SetOpt_SSH_Compression], page 179, for details.

INPUTS
none

5.538 easy:UnsetOpt_SSH_HostKeyFunction

NAME
easy:UnsetOpt_SSH_HostKeyFunction – callback to check host key (V2.0)

SYNOPSIS
`easy:UnsetOpt_SSH_HostKeyFunction()`

FUNCTION
See Section 5.274 [easy:SetOpt_SSH_HostKeyFunction], page 180, for details.

INPUTS
none

5.539 easy:UnsetOpt_SSH_Host_Public_Key_MD5

NAME
easy:UnsetOpt_SSH_Host_Public_Key_MD5 – checksum of SSH server public key

SYNOPSIS
`easy:UnsetOpt_SSH_Host_Public_Key_MD5()`

FUNCTION
See Section 5.275 [`easy:SetOpt_SSH_Host_Public_Key_MD5`], page 181, for details.

INPUTS
none

5.540 easy:UnsetOpt_SSH_KnownHosts

NAME
easy:UnsetOpt_SSH_KnownHosts – file name holding the SSH known hosts

SYNOPSIS
`easy:UnsetOpt_SSH_KnownHosts()`

FUNCTION
See Section 5.276 [`easy:SetOpt_SSH_KnownHosts`], page 181, for details.

INPUTS
none

5.541 easy:UnsetOpt_SSH_Private_KeyFile

NAME
easy:UnsetOpt_SSH_Private_KeyFile – get private key file for SSH auth

SYNOPSIS
`easy:UnsetOpt_SSH_Private_KeyFile()`

FUNCTION
See Section 5.277 [`easy:SetOpt_SSH_Private_KeyFile`], page 181, for details.

INPUTS
none

5.542 easy:UnsetOpt_SSH_Public_KeyFile

NAME
easy:UnsetOpt_SSH_Public_KeyFile – get public key file for SSH auth

SYNOPSIS
`easy:UnsetOpt_SSH_Public_KeyFile()`

FUNCTION
See Section 5.278 [easy:SetOpt_SSH_Public_KeyFile], page 182, for details.

INPUTS
none

5.543 easy:UnsetOpt_SSLCert

NAME
easy:UnsetOpt_SSLCert – get SSL client certificate

SYNOPSIS
`easy:UnsetOpt_SSLCert()`

FUNCTION
See Section 5.279 [easy:SetOpt_SSLCert], page 182, for details.

INPUTS
none

5.544 easy:UnsetOpt_SSLCert_Blob

NAME
easy:UnsetOpt_SSLCert_Blob – SSL client certificate from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_SSLCert_Blob()`

FUNCTION
See Section 5.280 [easy:SetOpt_SSLCert_Blob], page 183, for details.

INPUTS
none

5.545 easy:UnsetOpt_SSLCertType

NAME
easy:UnsetOpt_SSLCertType – specify type of the client SSL certificate

SYNOPSIS
`easy:UnsetOpt_SSLCertType()`

FUNCTION
See Section 5.281 [easy:SetOpt_SSLCertType], page 183, for details.

INPUTS
none

5.546 easy:UnsetOpt_SSL_Cipher_List

NAME
easy:UnsetOpt_SSL_Cipher_List – specify ciphers to use for TLS

SYNOPSIS
`easy:UnsetOpt_SSL_Cipher_List()`

FUNCTION
See Section 5.282 [`easy:SetOpt_SSL_Cipher_List`], page 183, for details.

INPUTS
none

5.547 easy:UnsetOpt_SSL_EC_Curves

NAME
easy:UnsetOpt_SSL_EC_Curves – key exchange curves (V2.0)

SYNOPSIS
`easy:UnsetOpt_SSL_EC_Curves()`

FUNCTION
See Section 5.283 [`easy:SetOpt_SSL_EC_Curves`], page 184, for details.

INPUTS
none

5.548 easy:UnsetOpt_SSL_Enable_Alpn

NAME
easy:UnsetOpt_SSL_Enable_Alpn – enable ALPN

SYNOPSIS
`easy:UnsetOpt_SSL_Enable_Alpn()`

FUNCTION
See Section 5.284 [`easy:SetOpt_SSL_Enable_Alpn`], page 184, for details.

INPUTS
none

5.549 easy:UnsetOpt_SSL_Enable_Npn

NAME
easy:UnsetOpt_SSL_Enable_Npn – enable NPN

SYNOPSIS
`easy:UnsetOpt_SSL_Enable_Npn()`

FUNCTION
See Section 5.285 [easy:SetOpt_SSL_Enable_Npn], page 185, for details.

INPUTS
none

5.550 easy:UnsetOpt_SSLEngine

NAME
easy:UnsetOpt_SSLEngine – get SSL engine identifier

SYNOPSIS
`easy:UnsetOpt_SSLEngine()`

FUNCTION
See Section 5.286 [easy:SetOpt_SSLEngine], page 185, for details.

INPUTS
none

5.551 easy:UnsetOpt_SSLEngine_Default

NAME
easy:UnsetOpt_SSLEngine_Default – make SSL engine default

SYNOPSIS
`easy:UnsetOpt_SSLEngine_Default()`

FUNCTION
See Section 5.287 [easy:SetOpt_SSLEngine_Default], page 185, for details.

INPUTS
none

5.552 easy:UnsetOpt_SSL_FalseStart

NAME
easy:UnsetOpt_SSL_FalseStart – enable TLS false start

SYNOPSIS
`easy:UnsetOpt_SSL_FalseStart()`

FUNCTION
See Section 5.288 [easy:SetOpt_SSL_FalseStart], page 186, for details.

INPUTS
none

5.553 easy:UnsetOpt_SSLKey

NAME
easy:UnsetOpt_SSLKey – specify private keyfile for TLS and SSL client cert

SYNOPSIS
`easy:UnsetOpt_SSLKey()`

FUNCTION
See Section 5.289 [easy:SetOpt_SSLKey], page 186, for details.

INPUTS
none

5.554 easy:UnsetOpt_SSLKey_Blob

NAME
easy:UnsetOpt_SSLKey_Blob – private key for client cert from memory blob (V2.0)

SYNOPSIS
`easy:UnsetOpt_SSLKey_Blob()`

FUNCTION
See Section 5.290 [easy:SetOpt_SSLKey_Blob], page 186, for details.

INPUTS
none

5.555 easy:UnsetOpt_SSLKeyType

NAME
easy:UnsetOpt_SSLKeyType – get type of the private key file

SYNOPSIS
`easy:UnsetOpt_SSLKeyType()`

FUNCTION
See Section 5.291 [easy:SetOpt_SSLKeyType], page 187, for details.

INPUTS
none

5.556 easy:UnsetOpt_SSL_Options

NAME
easy:UnsetOpt_SSL_Options – get SSL behavior options

SYNOPSIS
`easy:UnsetOpt_SSL_Options()`

FUNCTION
See Section 5.292 [easy:SetOpt_SSL_Options], page 187, for details.

INPUTS
none

5.557 easy:UnsetOpt_SSL_SessionID_Cache

NAME
easy:UnsetOpt_SSL_SessionID_Cache – enable/disable use of the SSL session-ID cache

SYNOPSIS
`easy:UnsetOpt_SSL_SessionID_Cache()`

FUNCTION
See Section 5.293 [easy:SetOpt_SSL_SessionID_Cache], page 188, for details.

INPUTS
none

5.558 easy:UnsetOpt_SSL_VerifyHost

NAME
easy:UnsetOpt_SSL_VerifyHost – verify the certificate's name against host

SYNOPSIS
`easy:UnsetOpt_SSL_VerifyHost()`

FUNCTION
See Section 5.294 [easy:SetOpt_SSL_VerifyHost], page 188, for details.

INPUTS
none

5.559 easy:UnsetOpt_SSL_VerifyPeer

NAME
easy:UnsetOpt_SSL_VerifyPeer – verify the peer's SSL certificate

SYNOPSIS
`easy:UnsetOpt_SSL_VerifyPeer()`

FUNCTION
See Section 5.295 [easy:SetOpt_SSL_VerifyPeer], page 189, for details.

INPUTS
none

5.560 easy:UnsetOpt_SSL_VerifyStatus

NAME
easy:UnsetOpt_SSL_VerifyStatus – verify the certificate's status

SYNOPSIS
`easy:UnsetOpt_SSL_VerifyStatus()`

FUNCTION
See Section 5.296 [easy:SetOpt_SSL_VerifyStatus], page 190, for details.

INPUTS
none

5.561 easy:UnsetOpt_SSLVersion

NAME
easy:UnsetOpt_SSLVersion – get preferred TLS/SSL version

SYNOPSIS
`easy:UnsetOpt_SSLVersion()`

FUNCTION
See Section 5.297 [easy:SetOpt_SSLVersion], page 190, for details.

INPUTS
none

5.562 easy:UnsetOpt_Stream_Depends

NAME
easy:UnsetOpt_Stream_Depends – get stream this transfer depends on

SYNOPSIS
`easy:UnsetOpt_Stream_Depends()`

FUNCTION
See Section 5.298 [easy:SetOpt_Stream_Depends], page 191, for details.

INPUTS
none

5.563 easy:UnsetOpt_Stream_Depends_e

NAME
easy:UnsetOpt_Stream_Depends_e – get stream this transfer depends on exclusively

SYNOPSIS
`easy:UnsetOpt_Stream_Depends_e()`

FUNCTION
See Section 5.299 [easy:SetOpt_Stream_Depends_e], page 192, for details.

INPUTS
none

5.564 easy:UnsetOpt_Stream_Weight

NAME
easy:UnsetOpt_Stream_Weight – get numerical stream weight

SYNOPSIS
`easy:UnsetOpt_Stream_Weight()`

FUNCTION
See Section 5.300 [easy:SetOpt_Stream_Weight], page 192, for details.

INPUTS
none

5.565 easy:UnsetOpt_Suppress_Connect_Headers

NAME
easy:UnsetOpt_Suppress_Connect_Headers – Suppress proxy CONNECT response headers from user callbacks

SYNOPSIS
`easy:UnsetOpt_Suppress_Connect_Headers()`

FUNCTION
See Section 5.301 [easy:SetOpt_Suppress_Connect_Headers], page 193, for details.

INPUTS
none

5.566 easy:UnsetOpt_TCP_FastOpen

NAME
easy:UnsetOpt_TCP_FastOpen – enable TCP Fast Open

SYNOPSIS
`easy:UnsetOpt_TCP_FastOpen()`

FUNCTION
See Section 5.302 [easy:SetOpt_TCP_FastOpen], page 194, for details.

INPUTS
none

5.567 easy:UnsetOpt_TCP_KeepAlive

NAME
easy:UnsetOpt_TCP_KeepAlive – enable TCP keep-alive probing

SYNOPSIS
`easy:UnsetOpt_TCP_KeepAlive()`

FUNCTION
See Section 5.303 [`easy:SetOpt_TCP_KeepAlive`], page 194, for details.

INPUTS
none

5.568 easy:UnsetOpt_TCP_KeepIdle

NAME
easy:UnsetOpt_TCP_KeepIdle – get TCP keep-alive idle time wait

SYNOPSIS
`easy:UnsetOpt_TCP_KeepIdle()`

FUNCTION
See Section 5.304 [`easy:SetOpt_TCP_KeepIdle`], page 194, for details.

INPUTS
none

5.569 easy:UnsetOpt_TCP_KeepIntvl

NAME
easy:UnsetOpt_TCP_KeepIntvl – get TCP keep-alive interval

SYNOPSIS
`easy:UnsetOpt_TCP_KeepIntvl()`

FUNCTION
See Section 5.305 [`easy:SetOpt_TCP_KeepIntvl`], page 195, for details.

INPUTS
none

5.570 easy:UnsetOpt_TCP_NoDelay

NAME
easy:UnsetOpt_TCP_NoDelay – get the TCP_NODELAY option

SYNOPSIS
`easy:UnsetOpt_TCP_NoDelay()`

FUNCTION
See Section 5.306 [easy:SetOpt_TCP_NoDelay], page 195, for details.

INPUTS
none

5.571 easy:UnsetOpt_TelnetOptions

NAME
easy:UnsetOpt_TelnetOptions – custom telnet options

SYNOPSIS
`easy:UnsetOpt_TelnetOptions()`

FUNCTION
See Section 5.307 [easy:SetOpt_TelnetOptions], page 196, for details.

INPUTS
none

5.572 easy:UnsetOpt_TFTP_BlkSize

NAME
easy:UnsetOpt_TFTP_BlkSize – TFTP block size

SYNOPSIS
`easy:UnsetOpt_TFTP_BlkSize()`

FUNCTION
See Section 5.308 [easy:SetOpt_TFTP_BlkSize], page 196, for details.

INPUTS
none

5.573 easy:UnsetOpt_TFTP_No_Options

NAME
easy:UnsetOpt_TFTP_No_Options – Do not send TFTP options requests.

SYNOPSIS
`easy:UnsetOpt_TFTP_No_Options()`

FUNCTION
See Section 5.309 [easy:SetOpt_TFTP_No_Options], page 196, for details.

INPUTS
none

5.574 easy:UnsetOpt_TimeCondition

NAME
easy:UnsetOpt_TimeCondition – select condition for a time request

SYNOPSIS
`easy:UnsetOpt_TimeCondition()`

FUNCTION
See Section 5.310 [easy:SetOpt_TimeCondition], page 197, for details.

INPUTS
none

5.575 easy:UnsetOpt_Timeout

NAME
easy:UnsetOpt_Timeout – get maximum time the request is allowed to take

SYNOPSIS
`easy:UnsetOpt_Timeout()`

FUNCTION
See Section 5.311 [easy:SetOpt_Timeout], page 197, for details.

INPUTS
none

5.576 easy:UnsetOpt_Timeout_MS

NAME
easy:UnsetOpt_Timeout_MS – get maximum time the request is allowed to take

SYNOPSIS
`easy:UnsetOpt_Timeout_MS()`

FUNCTION
See Section 5.312 [easy:SetOpt_Timeout_MS], page 198, for details.

INPUTS
none

5.577 easy:UnsetOpt_TimeValue

NAME
easy:UnsetOpt_TimeValue – get time value for conditional

SYNOPSIS
`easy:UnsetOpt_TimeValue()`

FUNCTION
See Section 5.313 [easy:SetOpt_TimeValue], page 198, for details.

INPUTS
none

5.578 easy:UnsetOpt_TimeValue_Large

NAME
easy:UnsetOpt_TimeValue_Large – time value for conditional (V2.0)

SYNOPSIS
`easy:UnsetOpt_TimeValue_Large()`

FUNCTION
See Section 5.314 [easy:SetOpt_TimeValue_Large], page 199, for details.

INPUTS
none

5.579 easy:UnsetOpt_TLS13_Ciphers

NAME
easy:UnsetOpt_TLS13_Ciphers – ciphers suites to use for TLS 1.3 (V2.0)

SYNOPSIS
`easy:UnsetOpt_TLS13_Ciphers()`

FUNCTION
See Section 5.315 [easy:SetOpt_TLS13_Ciphers], page 199, for details.

INPUTS
none

5.580 easy:UnsetOpt_TLSAuth_Password

NAME
easy:UnsetOpt_TLSAuth_Password – password to use for TLS authentication

SYNOPSIS
`easy:UnsetOpt_TLSAuth_Password()`

FUNCTION
See Section 5.316 [easy:SetOpt_TLSAuth_Password], page 200, for details.

INPUTS
none

5.581 easy:UnsetOpt_TLSAuth_Type

NAME
easy:UnsetOpt_TLSAuth_Type – get TLS authentication methods

SYNOPSIS
`easy:UnsetOpt_TLSAuth_Type()`

FUNCTION
See Section 5.317 [easy:SetOpt_TLSAuth_Type], page 200, for details.

INPUTS
none

5.582 easy:UnsetOpt_TLSAuth_UserName

NAME
easy:UnsetOpt_TLSAuth_UserName – user name to use for TLS authentication

SYNOPSIS
`easy:UnsetOpt_TLSAuth_UserName()`

FUNCTION
See Section 5.318 [easy:SetOpt_TLSAuth_UserName], page 200, for details.

INPUTS
none

5.583 easy:UnsetOpt_TrailerFunction

NAME
easy:UnsetOpt_TrailerFunction – callback for sending trailing headers (V2.0)

SYNOPSIS
`easy:UnsetOpt_TrailerFunction()`

FUNCTION
See Section 5.319 [easy:SetOpt_TrailerFunction], page 201, for details.

INPUTS
none

5.584 easy:UnsetOpt_Transfer_Encoding

NAME
easy:UnsetOpt_Transfer_Encoding – ask for HTTP Transfer Encoding

SYNOPSIS
`easy:UnsetOpt_Transfer_Encoding()`

FUNCTION
See Section 5.320 [easy:SetOpt_Transfer_Encoding], page 201, for details.

INPUTS
none

5.585 easy:UnsetOpt_TransferText

NAME
easy:UnsetOpt_TransferText – request a text based transfer for FTP

SYNOPSIS
`easy:UnsetOpt_TransferText()`

FUNCTION
See Section 5.321 [easy:SetOpt_TransferText], page 202, for details.

INPUTS
none

5.586 easy:UnsetOpt_Unix_Socket_Path

NAME
easy:UnsetOpt_Unix_Socket_Path – get Unix domain socket

SYNOPSIS
`easy:UnsetOpt_Unix_Socket_Path()`

FUNCTION
See Section 5.322 [easy:SetOpt_Unix_Socket_Path], page 202, for details.

INPUTS
none

5.587 easy:UnsetOpt_Unrestricted_Auth

NAME
easy:UnsetOpt_Unrestricted_Auth – send credentials to other hosts too

SYNOPSIS
`easy:UnsetOpt_Unrestricted_Auth()`

FUNCTION
See Section 5.323 [easy:SetOpt_Unrestricted_Auth], page 203, for details.

INPUTS
none

5.588 easy:UnsetOpt_Upkeep_Interval_MS

NAME
easy:UnsetOpt_Upkeep_Interval_MS – connection upkeep interval (V2.0)

SYNOPSIS
`easy:UnsetOpt_Upkeep_Interval_MS()`

FUNCTION
See Section 5.324 [`easy:SetOpt_Upkeep_Interval_MS`], page 203, for details.

INPUTS
none

5.589 easy:UnsetOpt_Upload

NAME
easy:UnsetOpt_Upload – enable data upload

SYNOPSIS
`easy:UnsetOpt_Upload()`

FUNCTION
See Section 5.325 [`easy:SetOpt_Upload`], page 204, for details.

INPUTS
none

5.590 easy:UnsetOpt_Upload_Buffersize

NAME
easy:UnsetOpt_Upload_Buffersize – upload buffer size (V2.0)

SYNOPSIS
`easy:UnsetOpt_Upload_Buffersize()`

FUNCTION
See Section 5.326 [`easy:SetOpt_Upload_Buffersize`], page 204, for details.

INPUTS
none

5.591 easy:UnsetOpt_URL

NAME
easy:UnsetOpt_URL – provide the URL to use in the request

SYNOPSIS
`easy:UnsetOpt_URL()`

FUNCTION
See Section 5.327 [easy:SetOpt_URL], page 205, for details.

INPUTS
none

5.592 easy:UnsetOpt_UserAgent

NAME
easy:UnsetOpt_UserAgent – get HTTP user-agent header

SYNOPSIS
`easy:UnsetOpt_UserAgent()`

FUNCTION
See Section 5.328 [easy:SetOpt_UserAgent], page 209, for details.

INPUTS
none

5.593 easy:UnsetOpt_UserName

NAME
easy:UnsetOpt_UserName – user name to use in authentication

SYNOPSIS
`easy:UnsetOpt_UserName()`

FUNCTION
See Section 5.329 [easy:SetOpt_UserName], page 209, for details.

INPUTS
none

5.594 easy:UnsetOpt_UserPwd

NAME
easy:UnsetOpt_UserPwd – user name and password to use in authentication

SYNOPSIS
`easy:UnsetOpt_UserPwd()`

FUNCTION
See Section 5.330 [easy:SetOpt_UserPwd], page 210, for details.

INPUTS
none

5.595 easy:UnsetOpt_Use_SSL

NAME
easy:UnsetOpt_Use_SSL – request using SSL / TLS for the transfer

SYNOPSIS
`easy:UnsetOpt_Use_SSL()`

FUNCTION
See Section 5.331 [easy:SetOpt_Use_SSL], page 211, for details.

INPUTS
none

5.596 easy:UnsetOpt_Verbose

NAME
easy:UnsetOpt_Verbose – get verbose mode on/off

SYNOPSIS
`easy:UnsetOpt_Verbose()`

FUNCTION
See Section 5.332 [easy:SetOpt_Verbose], page 211, for details.

INPUTS
none

5.597 easy:UnsetOpt_WildcardMatch

NAME
easy:UnsetOpt_WildcardMatch – enable directory wildcard transfers

SYNOPSIS
`easy:UnsetOpt_WildcardMatch()`

FUNCTION
See Section 5.333 [easy:SetOpt_WildcardMatch], page 212, for details.

INPUTS
none

5.598 easy:UnsetOpt_WriteFunction

NAME
easy:UnsetOpt_WriteFunction – get callback for writing received data

SYNOPSIS
`easy:UnsetOpt_WriteFunction()`

FUNCTION

See Section 5.334 [easy:SetOpt_WriteFunction], page 213, for details.

INPUTS

none

5.599 easy:UnsetOpt_WS_Options

NAME

easy:UnsetOpt_WS_Options – WebSocket behavior options (V2.0)

SYNOPSIS

```
easy:UnsetOpt_WS_Options()
```

FUNCTION

See Section 5.335 [easy:SetOpt_WS_Options], page 214, for details.

INPUTS

none

5.600 easy:UnsetOpt_XOAuth2_Bearer

NAME

easy:UnsetOpt_XOAuth2_Bearer – specify OAuth 2.0 access token

SYNOPSIS

```
easy:UnsetOpt_XOAuth2_Bearer()
```

FUNCTION

See Section 5.336 [easy:SetOpt_XOAuth2_Bearer], page 214, for details.

INPUTS

none

5.601 easy:Upkeep

NAME

easy:Upkeep – perform any connection upkeep checks

SYNOPSIS

```
easy:Upkeep()
```

FUNCTION

Some protocols have "connection upkeep" mechanisms. These mechanisms usually send some traffic on existing connections in order to keep them alive; this can prevent connections from being closed due to overzealous firewalls, for example.

Currently the only protocol with a connection upkeep mechanism is HTTP/2: when the connection upkeep interval is exceeded and easy:Upkeep() is called, an HTTP/2 PING frame is sent on the connection.

This function must be explicitly called in order to perform the upkeep work. The connection upkeep interval is set with `#CURLOPT_UPKEEP_INTERVAL_MS`.

INPUTS

none

6 Form methods

6.1 form:AddBuffer

NAME

form:AddBuffer – add file upload section from buffer

SYNOPSIS

```
form:AddBuffer(name, filename, content[, type, headers])
```

FUNCTION

form:AddBuffer() is used to append a file upload section (from a buffer source) when building a multipart/formdata HTTP POST (sometimes referred to as RFC 2388-style posts). Once you've added all the sections you want included, pass the form handle as parameter to `#CURLOPT_HTTPPOST`. See Section 5.156 [`easy:SetOpt_HTTPPost`], page 118, for details.

You must call form:Free() after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

First, there are some basics you need to understand about multipart/formdata posts. Each part consists of at least a `NAME` and a `CONTENTS` part. If the part is made for file upload, there are also a stored `CONTENT-TYPE` and a `FILENAME`. Below, we'll discuss what options you use to get these properties in the parts you want to add to your post.

The `name` argument must be a string which provides the name of this part. The name is not allowed to contain zero-valued bytes. The `filename` argument must be a string which provides the filename field in the content header. The `content` argument must contain the actual data to send. The optional argument `type` can be used to get the content-type for the part and the optional argument `headers` can be used to specify extra headers for the form POST section. This takes a table containing a list and appends the list of headers to those libcurl automatically generates.

INPUTS

`name`	name of the part
`filename`	filename for the content header
`content`	actual data to send
`type`	optional: content-type for the part
`headers`	optional: extra headers for the POST section

6.2 form:AddContent

NAME

form:AddContent – add a section to a multipart/formdata HTTP POST

SYNOPSIS

```
form:AddContent(name, content[, type, headers])
```

FUNCTION

form:AddContent() is used to append a section when building a multipart/formdata HTTP POST (sometimes referred to as RFC 2388-style posts). Once you've added all the sections you want included, pass the form handle as parameter to `#CURLOPT_HTTPPOST`. See Section 5.156 [`easy:SetOpt_HTTPPost`], page 118, for details.

You must call form:Free() after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

First, there are some basics you need to understand about multipart/formdata posts. Each part consists of at least a `NAME` and a `CONTENTS` part. If the part is made for file upload, there are also a stored `CONTENT-TYPE` and a `FILENAME`. Below, we'll discuss what options you use to get these properties in the parts you want to add to your post.

The `name` argument must be a string which provides the name of this part. The name is not allowed to contain zero-valued bytes. The `content` argument must contain the actual data to send. The optional argument `type` can be used to get the content-type for the part and the optional argument `headers` can be used to specify extra headers for the form POST section. This takes a table containing a list and appends the list of headers to those libcurl automatically generates.

INPUTS

`name`	name of the part
`content`	actual data to send
`type`	optional: content-type for the part
`headers`	optional: extra headers for the POST section

6.3 form:AddFile

NAME

form:AddFile – add file upload section to a multipart/formdata HTTP POST

SYNOPSIS

```
form:AddFile(name, path[, type, filename, headers])
```

FUNCTION

form:AddFile() is used to append a file upload section when building a multipart/formdata HTTP POST (sometimes referred to as RFC 2388-style posts). Once you've added all the sections you want included, pass the form handle as parameter to `#CURLOPT_HTTPPOST`. See Section 5.156 [`easy:SetOpt_HTTPPost`], page 118, for details.

You must call form:Free() after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

First, there are some basics you need to understand about multipart/formdata posts. Each part consists of at least a `NAME` and a `CONTENTS` part. If the part is made for file upload, there are also a stored `CONTENT-TYPE` and a `FILENAME`. Below, we'll discuss what options you use to get these properties in the parts you want to add to your post.

The `name` argument must be a string which provides the name of this part. The name is not allowed to contain zero-valued bytes.

The `path` argument must be get to the path of a filename to be uploaded. Libcurl sets the filename field to the basename of the provided filename, it reads the contents of the file and passes them as data and sets the content-type if the given file match one of the internally known file extensions. The given upload file has to exist in its full in the file system already when the upload starts, as libcurl needs to read the correct file size beforehand. The specified file needs to kept around until the associated transfer is done.

The optional argument `type` can be used to get the content-type for the part. The optional `filename` argument can be used to use a different file name than the one derived from `path` for the upload. The optional argument `headers` can be used to specify extra headers for the form POST section. This takes a table containing a list and appends the list of headers to those libcurl automatically generates.

INPUTS

`name`	name of the part
`path`	path to a file to upload
`type`	optional: content-type for the part
`filename`	optional: filename for the content header
`headers`	optional: extra headers for the POST section

6.4 form:AddFiles

NAME

form:AddFiles – add multiple file upload sections to a HTTP POST

SYNOPSIS

```
form:AddFiles(name, table)
```

FUNCTION

form:AddFiles() is used to append multiple file upload sections when building a multipart/formdata HTTP POST (sometimes referred to as RFC 2388-style posts). Once you've added all the sections you want included, pass the form handle as parameter to `#CURLOPT_HTTPPOST`. See Section 5.156 [`easy:SetOpt_HTTPPost`], page 118, for details.

You must call form:Free() after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

First, there are some basics you need to understand about multipart/formdata posts. Each part consists of at least a `NAME` and a `CONTENTS` part. If the part is made for file upload, there are also a stored `CONTENT-TYPE` and a `FILENAME`. Below, we'll discuss what options you use to get these properties in the parts you want to add to your post.

The `name` argument must be a string which provides the name of this part. The name is not allowed to contain zero-valued bytes.

The `table` argument must contain a table describing a list of files to be added to the form post object. There must be one item per file in the table. The individual table items can be of three different types:

1. a string: In that case, the string must simply contain the path to the file to be uploaded.
2. a table with two strings: In that case, the first string must contain the path to the file to be uploaded and the second string must contain the content-type for the file.
3. a table with three strings: Same as above, but the third string must contain a file name that should be used for the part instead of the file name derived from the path specified in the first string in the table

INPUTS

`name` name of the part

`table` table containing the files to be uploaded (see above)

6.5 form:AddStream

NAME

form:AddStream – add stream data to a multipart/formdata HTTP POST

SYNOPSIS

```
form:AddStream(name, len, func[, userdata, type, filename, headers])
```

FUNCTION

form:AddStream() is used to append a stream data section when building a multipart/formdata HTTP POST (sometimes referred to as RFC 2388-style posts). Once you've added all the sections you want included, pass the form handle as parameter to `#CURLOPT_HTTPPOST`. See Section 5.156 [`easy:SetOpt_HTTPPost`], page 118, for details.

You must call form:Free() after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a `"Expect: 100-continue"` header. You can disable this header with `#CURLOPT_HTTPHEADER` as usual.

First, there are some basics you need to understand about multipart/formdata posts. Each part consists of at least a `NAME` and a `CONTENTS` part. If the part is made for file upload, there are also a stored `CONTENT-TYPE` and a `FILENAME`. Below, we'll discuss what options you use to get these properties in the parts you want to add to your post.

The `name` argument must be a string which provides the name of this part. The name is not allowed to contain zero-valued bytes.

The `len` parameter must contain the number of bytes to add. The `func` parameter must be a callback function which will be called to provide the actual data to add. This callback function will be called repeatedly until it has returned exactly `len` bytes. The callback function behaves exactly like the one in `#CURLOPT_READFUNCTION`. See Section 5.249 [`easy:SetOpt_ReadFunction`], page 165, for details. If you pass the optional `userdata` argument, the value you specify here (it can be of any type) will be passed as the second parameter to your callback function.

The optional argument `type` can be used to get the content-type for the part. The optional `filename` argument can be used to get a the desired file name for the stream data. The optional argument `headers` can be used to specify extra headers for the form POST section. This takes a table containing a list and appends the list of headers to those libcurl automatically generates.

INPUTS

`name`	name of the part
`len`	number of bytes to stream
`func`	callback function that provides the stream data
`userdata`	optional: user data to pass to callback function
`type`	optional: content-type for the part
`filename`	optional: filename for the content header
`headers`	optional: extra headers for the POST section

6.6 form:Free

NAME

form:Free – free a previously build multipart/formdata HTTP POST chain

SYNOPSIS

```
form:Free()
```

FUNCTION

form:Free() is used to clean up data previously built/appended with hurl.Form(). This must be called when the data has been used, which typically means after easy:Perform() has been called.

INPUTS

none

6.7 form:Get

NAME

form:Get – serialize a previously built multipart/formdata HTTP POST chain

SYNOPSIS

```
s$ = form:Get()
form:Get(callback[, userdata])
```

FUNCTION

form:Get() is used to serialize data previously built/appended with hurl.Form().

There are two different ways of using this function: You can either call it with no arguments, in which case it will return a string containing the serialized data. Alternatively, you can also pass a callback function to it. In that case, the function you pass in

`callback` will be called by form:Get() and it will receive the serialized data in the first parameter. If you pass the optional `userdata` argument, the value you specify here (it can be of any type) will be passed as the second parameter to your callback function.

INPUTS

`callback`	callback function
`userdata`	optional: user data to pass to callback function

RESULTS

`s$`	serialized data

7 Mime methods

7.1 mime:AddPart

NAME

mime:AddPart – append a new part to a mime object (V2.0)

SYNOPSIS

```
handle = mime:AddPart([table])
```

FUNCTION

This creates and appends a new part to the given mime object and returns a handle to it. The returned mime part handle can subsequently be populated using functions from the mime part API such as mimepart:Name() and mimepart:Data(). Mime objects can be created using easy:Mime().

Optionally, you can also initialize the mime part object right at creation time by passing the optional table argument. This table can contain the following fields:

`Name` Set the mime part's name. See Section 8.7 [mimepart:Name], page 321, for details.

`Data` Set the mime part's body data. See Section 8.1 [mimepart:Data], page 319, for details.

`FileData` Set the mime part's body data from a file source. See Section 8.3 [mimepart:FileData], page 320, for details.

`Filename` Set the mime part's remote file name. See Section 8.4 [mimepart:Filename], page 320, for details.

`Type` Set the mime part's type. See Section 8.9 [mimepart:Type], page 322, for details.

`Encoder` Set the mime part's encoder. See Section 8.2 [mimepart:Encoder], page 319, for details.

`Headers` Set the mime part's headers. See Section 8.6 [mimepart:Headers], page 321, for details.

`Subparts` Set the mime part's subparts. See Section 8.8 [mimepart:Subparts], page 322, for details.

INPUTS

`table` optional: table argument for initializing the mime part

RESULTS

`handle` mime part handle

7.2 mime:Easy

NAME

mime:Easy – get easy object associated with mime object (V2.0)

SYNOPSIS

```
handle = mime:Easy()
```

FUNCTION

This method returns the easy handle associated with the mime object. It is the easy handle that was used to create the mime object when calling easy:Mime().

INPUTS

none

RESULTS

`handle` easy handle

7.3 mime:Free

NAME

mime:Free – free mime object (V2.0)

SYNOPSIS

```
mime:Free()
```

FUNCTION

Free a mime object created by easy:Mime(). After this function returns, the mime object may no longer be used.

INPUTS

none

8 Mime part methods

8.1 mimepart:Data

NAME

mimepart:Data – set mime part's body data (V2.0)

SYNOPSIS

`mimepart:Data(data$)`

FUNCTION

This sets sets the mime part's body content to `data$`. Note that `data$` can also contain binary data. It is not limited to text. To set the mime part's body data from a file source, use mimepart:FileData() instead.

Mime parts are created using mime:AddPart().

INPUTS

`data$` desired data

8.2 mimepart:Encoder

NAME

mimepart:Encoder – set mime part's encoder (V2.0)

SYNOPSIS

`mimepart:Encoder(enc$)`

FUNCTION

This requests a mime part's content to be encoded before being transmitted.

Upon multipart rendering, the part's content is encoded according to the pertaining scheme and a corresponding `"Content-Transfer-Encoding"` header is added to the part.

Supported encoding schemes are:

`binary` The data is left unchanged, the header is added.

`8bit` Header added, no data change.

`7bit` The data is unchanged, but is each byte is checked to be a 7-bit value; if not, a read error occurs.

`base64` Data is converted to base64 encoding, then split in CRLF-terminated lines of at most 76 characters.

`quoted-printable`
Data is encoded in quoted printable lines of at most 76 characters. Since the resulting size of the final data cannot be determined prior to reading the original data, it is left as unknown, causing chunked transfer in HTTP. For the same reason, this encoder may not be used with IMAP. This encoder targets text data that is mostly ASCII and should not be used with other types of data.

If the original data is already encoded in such a scheme, a custom Content-Transfer-Encoding header should be added with mimepart:Headers() instead of setting a part encoder.

Encoding should not be applied to multiparts, thus the use of this function on a part with content set with mimepart:Subparts() is strongly discouraged.

INPUTS

`enc$` desired encoder

8.3 mimepart:FileData

NAME

mimepart:FileData – set mime part's body data from file (V2.0)

SYNOPSIS

```
mimepart:FileData(file$)
```

FUNCTION

This sets sets the mime part's body data from the file specified by `file$`. This is an alternative to using mimepart:Data(). As a side effect, the part's remote file name is set to the base name of the given filename if it is a valid named file. This can be undone or overridden by a subsequent call to mimepart:Filename().

The contents of the file is read during the file transfer in a streaming manner to allow huge files to get transferred without using much memory. It therefore requires that the file is kept intact during the entire request.

If the file size cannot be determined before actually reading it (such as for a device or named pipe), the whole mime structure containing the part will be transferred as chunks by HTTP and rejected by IMAP.

Mime parts are created using mime:AddPart().

INPUTS

`file$` file whose contents to use as body data

8.4 mimepart:Filename

NAME

mimepart:Filename – set mime part's remote file name (V2.0)

SYNOPSIS

```
mimepart:Filename(filename$)
```

FUNCTION

This sets sets the mime part's remote file name to `filename$`. When a remote file name is set, content data is processed as a file, whatever is the part's content source. A part's remote file name is transmitted to the server in the associated Content-Disposition generated header. Mime parts are created using mime:AddPart().

INPUTS

`filename$`
desired remote file name

8.5 mimepart:Free

NAME

mimepart:Free – free mime part object (V2.0)

SYNOPSIS

`mimepart:Free()`

FUNCTION

Free a mime part object created by mime:AddPart(). After this function returns, the mime part object may no longer be used.

INPUTS

none

8.6 mimepart:Headers

NAME

mimepart:Headers – set mime part's custom headers (V2.0)

SYNOPSIS

`mimepart:Headers(headers)`

FUNCTION

This sets sets the mime part's custom headers to the ones specified in `headers`. The `headers` parameter must be a table containing a list of strings specifying the custom headers.

INPUTS

`headers` custom headers

8.7 mimepart:Name

NAME

mimepart:Name – set mime part's name (V2.0)

SYNOPSIS

`mimepart:Name(name$)`

FUNCTION

This sets the mime part's name to `name$`. This is the way HTTP form fields are named. Mime parts are created using mime:AddPart().

INPUTS

`name$` desired name

8.8 mimepart:Subparts

NAME

mimepart:Subparts – set sub-parts of a multipart mime part (V2.0)

SYNOPSIS

`mimepart:Subparts(subparts[, type, name, filename, headers])`

FUNCTION

This sets a multipart mime part's content from the mime object which is passed in the `subparts` parameter. Mime objects are created using easy:Mime(). Mime parts are created using mime:AddPart().

Optionally, this method can also initialize the mime part object to the data specified in the optional arguments `type`, `name`, `filename`, and `headers`. Passing these arguments is equivalent to calling the methods mimepart:Type(), mimepart:Name(), mimepart:Filename(), and mimepart:Headers(), respectively.

Note that after this methods succeeds, the mime object handle specified in `subparts` belongs to the multipart part and must not be freed explicitly. It may however be updated by subsequent calls to mime API functions.

INPUTS

`subparts`	mime object holding the subparts
`type`	optional: mime part type
`name`	optional: mime part name
`filename`	optional: mime part remote file name
`headers`	optional: mime part headers

8.9 mimepart:Type

NAME

mimepart:Type – set mime part's content type (V2.0)

SYNOPSIS

`mimepart:Type(type$)`

FUNCTION

This sets the mime part's content type to `type$`. In the absence of a mime type and if needed by the protocol specifications, a default mime type is determined by the context:

- If set as a custom header, use this value.
- application/form-data for an HTTP form post.
- If a remote file name is set, the mime type is taken from the file name extension, or application/octet-stream by default.
- For a multipart part, multipart/mixed.
- text/plain in other cases.

Mime parts are created using mime:AddPart().

INPUTS

`type$` desired type

9 Multi methods

9.1 multi:AddHandle

NAME

multi:AddHandle – add an easy handle to a multi session

SYNOPSIS

```
multi:AddHandle(handle)
```

FUNCTION

Adds a standard easy handle to the multi stack. This function call will make this multi handle control the specified easy handle.

While an easy handle is added to a multi stack, you cannot and you must not use easy:Perform() on that handle. After having removed the easy handle from the multi stack again, it is perfectly fine to use it with the easy interface again.

If the easy handle is not get to use a shared (`#CURLOPT_SHARE`) or global DNS cache (`#CURLOPT_DNS_USE_GLOBAL_CACHE`), it will be made to use the DNS cache that is shared between all easy handles within the multi handle when multi:AddHandle() is called.

When an easy interface is added to a multi handle, it will use a shared connection cache owned by the multi handle. Removing and adding new easy handles will not affect the pool of connections or the ability to do connection re-use.

If you have `#CURLMOPT_TIMERFUNCTION` get in the multi handle (and you really should if you're working event-based with multi:SocketAction() and friends), that callback will be called from within this function to ask for an updated timer so that your main event loop will get the activity on this handle to get started.

The easy handle will remain added to the multi handle until you remove it again with multi:RemoveHandle() - even when a transfer with that specific easy handle is completed.

You should remove the easy handle from the multi stack before you terminate first the easy handle and then the multi handle:

1. multi:RemoveHandle()
2. easy:Close()
3. multi:Close()

INPUTS

`handle` easy handle to add to multi handle

9.2 multi:Close

NAME

multi:Close – close down a multi session

SYNOPSIS

```
multi:Close()
```

FUNCTION

Cleans up and removes a whole multi stack. It does not free or touch any individual easy handles in any way - they still need to be closed individually, using the usual easy:Close() way. The order of cleaning up should be:

1. multi:RemoveHandle() before any easy handles are cleaned up
2. easy:Close() can now be called independently since the easy handle is no longer connected to the multi handle
3. multi:Close() should be called when all easy handles are removed

INPUTS

none

9.3 multi:InfoRead

NAME

multi:InfoRead – read multi stack informationals

SYNOPSIS

```
msg, result, remaining, handle = multi:InfoRead()
```

FUNCTION

Ask the multi handle if there are any messages/informationals from the individual transfers. Messages may include informationals such as an error code from the transfer or just the fact that a transfer is completed. More details on these should be written down as well.

This call returns four values: `msg` contains the type of message received. This can be `#CURLMSG_NONE` or `#CURLMSG_DONE`. `result` contains the message result. The `remaining` return value indicates how many messages are still in the queue after this function was called. The `handle` return value contains the easy handle that has previously been added to the multi handle.

When you fetch a message using this function, it is removed from the internal queue so calling this function again will not return the same message again. It will instead return new messages at each new invoke until the queue is emptied.

When `msg` is `#CURLMSG_DONE`, the message identifies a transfer that is done, and then `result` contains the return code for the easy handle that just completed.

INPUTS

none

RESULTS

`msg` message type read (see above for possible types)

`result` message-specific result code

`remaining` number of remaining messages

`handle` the easy handle that has previously been added to the multi handle (V1.1)

9.4 multi:Perform

NAME

multi:Perform – reads/writes available data from each easy handle

SYNOPSIS

```
running = multi:Perform()
```

FUNCTION

This function handles transfers on all the added handles that need attention in an non-blocking fashion.

When an application has found out there's data available for the multi handle or a timeout has elapsed, the application should call this function to read/write whatever there is to read or write right now etc. multi:Perform() returns as soon as the reads/writes are done. This function does not require that there actually is any data available for reading or that data can be written, it can be called just in case. It will return the number of handles that still transfer data.

If the amount of running handles is changed from the previous call (or is less than the amount of easy handles you've added to the multi handle), you know that there is one or more transfers less "running". You can then call multi:InfoRead() to get information about each individual completed transfer, and that returned info includes CURLcode and more. If an added handle fails very quickly, it may never be counted as a running handle.

When `running` is get to zero on the return of this function, there is no longer any transfers in progress.

INPUTS

none

RESULTS

`running` number of running handles

9.5 multi:RemoveHandle

NAME

multi:RemoveHandle – remove an easy handle from a multi session

SYNOPSIS

```
multi:RemoveHandle(handle)
```

FUNCTION

Removes a given handle from the handle. This will make the specified easy handle be removed from this multi handle's control.

When the easy handle has been removed from a multi stack, it is again perfectly legal to invoke easy:Perform() on this easy handle.

Removing an easy handle while being used is perfectly legal and will effectively halt the transfer in progress involving that easy handle. All other easy handles and transfers will remain unaffected.

It is fine to remove a handle at any time during a transfer, just not from within any libcurl callback function.

INPUTS

`handle` easy handle to remove from multi handle

9.6 multi:SetOpt

NAME

multi:SetOpt – get options for a curl multi handle

SYNOPSIS

```
multi:SetOpt(option, param)
```

FUNCTION

multi:SetOpt() is used to tell a libcurl multi handle how to behave. By using the appropriate options to multi:SetOpt(), you can change libcurl's behaviour when using that multi handle. All options are get with the option followed by the parameter `param`. That parameter can be a number, a function, a string, or a table, depending on what the specific option expects. Read this manual carefully as bad input values may cause libcurl to behave badly! You can only get one option in each function call.

The following types are currently supported for `option`:

`#CURLMOPT_CHUNK_LENGTH_PENALTY_SIZE`
: See Section 9.7 [multi:SetOpt_Chunk_Length_Penalty_Size], page 329, for details.

`#CURLMOPT_CONTENT_LENGTH_PENALTY_SIZE`
: See Section 9.8 [multi:SetOpt_Content_Length_Penalty_Size], page 329, for details.

`#CURLMOPT_MAXCONNECTS`
: See Section 9.10 [multi:SetOpt_MaxConnects], page 330, for details.

`#CURLMOPT_MAX_CONCURRENT_STREAMS`
: See Section 9.9 [multi:SetOpt_Max_Concurrent_Streams], page 330, for details. (V2.0)

`#CURLMOPT_MAX_HOST_CONNECTIONS`
: See Section 9.11 [multi:SetOpt_Max_Host_Connections], page 331, for details.

`#CURLMOPT_MAX_PIPELINE_LENGTH`
: See Section 9.12 [multi:SetOpt_Max_Pipeline_Length], page 331, for details.

`#CURLMOPT_MAX_TOTAL_CONNECTIONS`
: See Section 9.13 [multi:SetOpt_Max_Total_Connections], page 331, for details.

`#CURLMOPT_PIPELINING`
: See Section 9.14 [multi:SetOpt_Pipelining], page 332, for details.

`#CURLMOPT_PIPELINING_SERVER_BL`
: See Section 9.15 [multi:SetOpt_Pipelining_Server_Bl], page 333, for details.

`#CURLMOPT_PIPELINING_SITE_BL`
: See Section 9.16 [multi:SetOpt_Pipelining_Site_Bl], page 333, for details.

`#CURLMOPT_SOCKETFUNCTION`
: See Section 9.17 [multi:SetOpt_SocketFunction], page 334, for details.

`#CURLMOPT_TIMERFUNCTION`
: See Section 9.18 [multi:SetOpt_TimerFunction], page 334, for details.

INPUTS

`option` option type to get

`parameter`
: value to get option to

9.7 multi:SetOpt_Chunk_Length_Penalty_Size

NAME

multi:SetOpt_Chunk_Length_Penalty_Size – chunk length threshold for pipelining

SYNOPSIS

```
multi:SetOpt_Chunk_Length_Penalty_Size(size)
```

FUNCTION

Pass a number with a size in bytes. If a pipelined connection is currently processing a chunked (Transfer-encoding: chunked) request with a current chunk length larger than `#CURLMOPT_CHUNK_LENGTH_PENALTY_SIZE`, that pipeline will not be considered for additional requests, even if it is shorter than `#CURLMOPT_MAX_PIPELINE_LENGTH`.

INPUTS

`size` input value

9.8 multi:SetOpt_Content_Length_Penalty_Size

NAME

multi:SetOpt_Content_Length_Penalty_Size – size threshold for pipelining penalty

SYNOPSIS

```
multi:SetOpt_Content_Length_Penalty_Size(size)
```

FUNCTION

Pass a number with a size in bytes. If a pipelined connection is currently processing a request with a Content-Length larger than this `#CURLMOPT_CONTENT_LENGTH_PENALTY_SIZE`, that pipeline will then not be considered for additional requests, even if it is shorter than `#CURLMOPT_MAX_PIPELINE_LENGTH`.

INPUTS

`size` input value

9.9 multi:SetOpt_Max_Concurrent_Streams

NAME

multi:SetOpt_Max_Concurrent_Streams – max concurrent streams for http2 (V2.0)

SYNOPSIS

```
multi:SetOpt_Max_Concurrent_Streams(max)
```

FUNCTION

Pass a value indicating the maximum concurrent streams for http2. The get number will be used as the maximum number of concurrent streams for a connections that libcurl should support on connections done using HTTP/2.

Valid values range from 1 to 2147483647 (2^31 - 1) and defaults to 100. The value passed here would be honored based on other system resources properties.

INPUTS

`max` input value

9.10 multi:SetOpt_MaxConnects

NAME

multi:SetOpt_MaxConnects – get size of connection cache

SYNOPSIS

```
multi:SetOpt_MaxConnects(max)
```

FUNCTION

Pass a number indicating the max. The get number will be used as the maximum amount of simultaneously open connections that libcurl may keep in its connection cache after completed use. By default libcurl will enlarge the size for each added easy handle to make it fit 4 times the number of added easy handles.

By setting this option, you can prevent the cache size from growing beyond the limit get by you.

When the cache is full, curl closes the oldest one in the cache to prevent the number of open connections from increasing.

This option is for the multi handle's use only, when using the easy interface you should instead use the `#CURLOPT_MAXCONNECTS` option.

See `#CURLMOPT_MAX_TOTAL_CONNECTIONS` for limiting the number of active connections.

INPUTS

`max` input value

9.11 multi:SetOpt_Max_Host_Connections

NAME

multi:SetOpt_Max_Host_Connections – get max number of connections to a single host

SYNOPSIS

```
multi:SetOpt_Max_Host_Connections(max)
```

FUNCTION

Pass a number to indicate max. The get number will be used as the maximum amount of simultaneously open connections to a single host (a host being the same as a host name + port number pair). For each new session to a host, libcurl will open a new connection up to the limit get by `#CURLMOPT_MAX_HOST_CONNECTIONS`. When the limit is reached, the sessions will be pending until a connection becomes available. If `#CURLMOPT_PIPELINING` is enabled, libcurl will try to pipeline if the host is capable of it.

The default max value is 0, unlimited. However, for backwards compatibility, setting it to 0 when `#CURLMOPT_PIPELINING` is 1 will not be treated as unlimited. Instead it will open only 1 connection and try to pipeline on it.

This get limit is also used for proxy connections, and then the proxy is considered to be the host for which this limit counts.

INPUTS

`max` input value

9.12 multi:SetOpt_Max_Pipeline_Length

NAME

multi:SetOpt_Max_Pipeline_Length – maximum number of requests in a pipeline

SYNOPSIS

```
multi:SetOpt_Max_Pipeline_Length(max)
```

FUNCTION

Pass a number. The get max number will be used as the maximum amount of outstanding requests in an HTTP/1.1 pipelined connection. This option is only used for HTTP/1.1 pipelining, not for HTTP/2 multiplexing.

When this limit is reached, libcurl will use another connection to the same host (see `#CURLMOPT_MAX_HOST_CONNECTIONS`), or queue the request until one of the pipelines to the host is ready to accept a request. Thus, the total number of requests in-flight is `#CURLMOPT_MAX_HOST_CONNECTIONS` * `#CURLMOPT_MAX_PIPELINE_LENGTH`.

INPUTS

`max` input value

9.13 multi:SetOpt_Max_Total_Connections

NAME

multi:SetOpt_Max_Total_Connections – max simultaneously open connections

SYNOPSIS

```
multi:SetOpt_Max_Total_Connections(amount)
```

FUNCTION

Pass a number for the amount. The get number will be used as the maximum number of simultaneously open connections in total using this multi handle. For each new session, libcurl will open a new connection up to the limit get by `#CURLMOPT_MAX_TOTAL_CONNECTIONS`. When the limit is reached, the sessions will be pending until there are available connections. If `#CURLMOPT_PIPELINING` is enabled, libcurl will try to pipeline or use multiplexing if the host is capable of it.

INPUTS

`amount` input value

9.14 multi:SetOpt_Pipelining

NAME

multi:SetOpt_Pipelining – enable HTTP pipelining and multiplexing

SYNOPSIS

```
multi:SetOpt_Pipelining(bitmask)
```

FUNCTION

Pass in the bitmask parameter to instruct libcurl to enable HTTP pipelining and/or HTTP/2 multiplexing for this multi handle.

When enabled, libcurl will attempt to use those protocol features when doing parallel requests to the same hosts.

For pipelining, this means that if you add a second request that can use an already existing connection, the second request will be "piped" on the same connection rather than being executed in parallel.

For multiplexing, this means that follow-up requests can re-use an existing connection and send the new request multiplexed over that at the same time as other transfers are already using that single connection.

There are several other related options that are interesting to tweak and adjust to alter how libcurl spreads out requests on different connections or not etc.

Before 7.43.0, this option was get to 1 and 0 to enable and disable HTTP/1.1 pipelining.

Starting in 7.43.0, bitmask's second bit also has a meaning, and you can ask for pipelining and multiplexing independently of each other by toggling the correct bits.

`#CURLPIPE_NOTHING`
: Default, which means doing no attempts at pipelining or multiplexing.

`#CURLPIPE_HTTP1`
: If this bit is get, libcurl will try to pipeline HTTP/1.1 requests on connections that are already established and in use to hosts. This bit is deprecated and has no effect since version 7.62.0.

`#CURLPIPE_MULTIPLEX`
: If this bit is get, libcurl will try to multiplex the new transfer over an existing connection if possible. This requires HTTP/2.

INPUTS

`bitmask` input value

9.15 multi:SetOpt_Pipelining_Server_Bl

NAME

multi:SetOpt_Pipelining_Server_Bl – pipelining server blacklist

SYNOPSIS

```
multi:SetOpt_Pipelining_Server_Bl(servers)
```

FUNCTION

Pass a table containing a list of strings here. This is a list of server types prefixes (in the Server: HTTP header) that are blacklisted from pipelining, i.e server types that are known to not support HTTP pipelining.

Note that the comparison matches if the Server: header begins with the string in the blacklist, i.e "Server: Ninja 1.2.3" and "Server: Ninja 1.4.0" can both be blacklisted by having "Ninja" in the backlist.

Pass an empty table to clear the blacklist.

INPUTS

`servers` input value

9.16 multi:SetOpt_Pipelining_Site_Bl

NAME

multi:SetOpt_Pipelining_Site_Bl – pipelining host blacklist

SYNOPSIS

```
multi:SetOpt_Pipelining_Site_Bl(hosts)
```

FUNCTION

Pass a table containing a list of strings here. This is a list of sites that are blacklisted from pipelining, i.e sites that are known to not support HTTP pipelining.

Pass an empty table to clear the blacklist.

INPUTS

`hosts` input value

9.17 multi:SetOpt_SocketFunction

NAME

multi:SetOpt_SocketFunction – callback informed about what to wait for

SYNOPSIS

```
multi:SetOpt_SocketFunction(socket_callback[, userdata])
```

FUNCTION

Pass a callback function.

When the multi:SocketAction() function runs, it informs the application about updates in the socket (file descriptor) status by doing none, one, or multiple calls to the socket callback. The callback gets status updates with changes since the previous time the callback was called.

The callback receives three arguments: The first argument is an easy handle, the second argument is a socket descriptor, and the third argument informs the callback on the status of the given socket. It can hold one of these values:

`#CURL_POLL_IN`
: Wait for incoming data. For the socket to become readable.

`#CURL_POLL_OUT`
: Wait for outgoing data. For the socket to become writable.

`#CURL_POLL_INOUT`
: Wait for incoming and outgoing data. For the socket to become readable or writable.

`#CURL_POLL_REMOVE`
: The specified socket/file descriptor is no longer used by libcurl.

If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a fourth parameter. The `userdata` parameter can be of any type.

INPUTS

`socket_callback`
: input value

`userdata` optional: user data to pass to callback function

9.18 multi:SetOpt_TimerFunction

NAME

multi:SetOpt_TimerFunction – get callback to receive timeout values

SYNOPSIS

```
multi:SetOpt_TimerFunction(timer_callback[, userdata])
```

FUNCTION

Pass a callback function.

Certain features, such as timeouts and retries, require you to call libcurl even when there is no activity on the file descriptors.

Your callback function will receive a single parameter called `timeout_ms`. If you pass the optional `userdata` argument, the value you pass in `userdata` will be passed to your callback function as a second parameter. The `userdata` parameter can be of any type

Once called, your callback function should install a non-repeating timer with an interval of `timeout_ms`. Each time that timer fires, call either multi:SocketAction() or multi:Perform() depending on which interface you use.

A `timeout_ms` value of -1 means you should delete your timer.

A `timeout_ms` value of 0 means you should call multi:SocketAction() or multi:Perform() (once) as soon as possible.

The timer callback will only be called when the `timeout_ms` changes.

The timer callback should return 0 on success, and -1 on error. This callback can be used instead of, or in addition to, multi:Timeout().

INPUTS

`timer_callback`
: input value

`userdata`
: optional: user data to pass to callback function

9.19 multi:SocketAction

NAME

multi:SocketAction – reads/writes available data given an action

SYNOPSIS

```
running = multi:SocketAction(socket, mask)
```

FUNCTION

When the application has detected action on a socket handled by libcurl, it should call multi:SocketAction() with the `socket` argument get to the socket with the action. When the events on a socket are known, they can be passed as an events bitmask `mask` by first setting `mask` to 0, and then adding using bitwise OR (`|`) any combination of events to be chosen from `#CURL_CSELECT_IN`, `#CURL_CSELECT_OUT` or `#CURL_CSELECT_ERR`. When the events on a socket are unknown, pass 0 instead, and libcurl will test the descriptor internally. It is also permissible to pass `#CURL_SOCKET_TIMEOUT` to the `socket` parameter in order to initiate the whole process or when a timeout occurs.

At return, `running` contains the number of running easy handles within the multi handle. When this number reaches zero, all transfers are complete/done. When you call multi:SocketAction() on a specific socket and the counter decreases by one, it DOES NOT necessarily mean that this exact socket/transfer is the one that completed. Use multi:InfoRead() to figure out which easy handle that completed.

The multi:SocketAction() functions inform the application about updates in the socket (file descriptor) status by doing none, one, or multiple calls to the socket callback function get with the `#CURLMOPT_SOCKETFUNCTION` option to multi:SetOpt(). They update the status with changes since the previous time the callback was called.

Get the timeout time by setting the `#CURLMOPT_TIMERFUNCTION` option with multi:SetOpt(). Your application will then get called with information on how long to wait for socket actions at most before doing the timeout action: call the multi:SocketAction() function with the `socket` argument get to `#CURL_SOCKET_TIMEOUT`. You can also use the multi:Timeout() function to poll the value at any given time, but for an event-based system using the callback is far better than relying on polling the timeout value.

INPUTS

`socket` socket to use

`mask` mask to use

RESULTS

`running` number of running handles

9.20 multi:Timeout

NAME

multi:Timeout – how long to wait for action before proceeding

SYNOPSIS

```
ms = multi:Timeout()
```

FUNCTION

An application using the libcurl multi interface should call multi:Timeout() to figure out how long it should wait for socket actions - at most - before proceeding.

Proceeding means either doing the socket-style timeout action: call the multi:SocketAction() function with the `sockfd` argument get to `#CURL_SOCKET_TIMEOUT`, or call multi:Perform() if you're using the simpler and older multi interface approach.

The timeout value returned is in number of milliseconds at this very moment. If 0, it means you should proceed immediately without waiting for anything. If it returns -1, there's no timeout at all get.

An application that uses the multi_socket API SHOULD NOT use this function, but SHOULD instead use multi:SetOpt() and its `#CURLMOPT_TIMERFUNCTION` option for proper and desired behavior.

Note: if libcurl returns a -1 timeout here, it just means that libcurl currently has no stored timeout value. You must not wait too long (more than a few seconds perhaps) before you call multi:Perform() again.

INPUTS

none

RESULTS

`ms` current timeout value

9.21 multi:Wait

NAME

multi:Wait – polls on all easy handles in a multi handle

SYNOPSIS

```
multi:Wait(timeout_ms)
```

FUNCTION

multi:Wait() polls all file descriptors used by the curl easy handles contained in the given multi handle get. It will block until activity is detected on at least one of the handles or `timeout_ms` has passed. Alternatively, if the multi handle has a pending internal timeout that has a shorter expiry time than `timeout_ms`, that shorter time will be used instead to make sure timeout accuracy is reasonably kept.

INPUTS

`timeout_ms`
: maximum amount of time to wait (in milliseconds)

10 Share methods

10.1 share:Close

NAME

share:Close – clean up a shared object

SYNOPSIS

`share:Close()`

FUNCTION

This function deletes a shared object. The share handle cannot be used anymore when this function has been called.

INPUTS

none

10.2 share:SetOpt

NAME

share:SetOpt – get options for a shared object

SYNOPSIS

`share:SetOpt(option, parameter)`

FUNCTION

Set the `option` to `parameter` for the given share.

The following option types are currently supported for `option`:

`#CURLSHOPT_SHARE`
: See Section 10.3 [share:SetOpt_Share], page 339, for details.

`#CURLSHOPT_UNSHARE`
: See Section 10.4 [share:SetOpt_Unshare], page 340, for details.

INPUTS

`option` option type to get

`parameter`
: value to get option to

10.3 share:SetOpt_Share

NAME

share:SetOpt_Share – get type of data to be shared

SYNOPSIS

`share:SetOpt_Share(type)`

FUNCTION

The parameter `type` specifies a type of data that should be shared. This may be get to one of the values described below.

`#CURL_LOCK_DATA_COOKIE`
: Cookie data will be shared across the easy handles using this shared object.

`#CURL_LOCK_DATA_DNS`
: Cached DNS hosts will be shared across the easy handles using this shared object. Note that when you use the multi interface, all easy handles added to the same multi handle will share DNS cache by default without using this option.

`#CURL_LOCK_DATA_SSL_SESSION`
: SSL session IDs will be shared across the easy handles using this shared object. This will reduce the time spent in the SSL handshake when reconnecting to the same server. Note SSL session IDs are reused within the same easy handle by default. Note this symbol was added in 7.10.3 but was not implemented until 7.23.0.

`#CURL_LOCK_DATA_CONNECT`
: Put the connection cache in the share object and make all easy handles using this share object share the connection cache. Using this, you can for example do multi-threaded libcurl use with one handle in each thread, and yet have a shared pool of unused connections and this way get way better connection re-use than if you use one separate pool in each thread.

  Connections that are used for HTTP/1.1 Pipelining or HTTP/2 multiplexing only get additional transfers added to them if the existing connection is held by the same multi or easy handle. libcurl does not support doing HTTP/2 streams in different threads using a shared connection.

  Note that when you use the multi interface, all easy handles added to the same multi handle will share connection cache by default without using this option.

`#CURL_LOCK_DATA_PSL`
: The Public Suffix List stored in the share object is made available to all easy handle bound to the later. Since the Public Suffix List is periodically refreshed, this avoids updates in too many different contexts. Note that when you use the multi interface, all easy handles added to the same multi handle will share PSL cache by default without using this option.

INPUTS

`type` desired type (see above)

10.4 share:SetOpt_Unshare

NAME

share:SetOpt_Unshare – unshare data type

SYNOPSIS

```
share:SetOpt_Unshare(type)
```

FUNCTION

This option does the opposite of `#CURLSHOPT_SHARE`. It specifies that the specified parameter will no longer be shared. Valid values are the same as those for `#CURLSHOPT_SHARE`. See Section 10.3 [`share:SetOpt_Share`], page 339, for details.

INPUTS

`type`	desired type (see above)

11 URL methods

11.1 url:Dup

NAME

url:Dup – duplicate URL object (V2.0)

SYNOPSIS

```
copy = url:Dup()
```

FUNCTION

url:Dup() duplicates the URL object and returns a copy.

INPUTS

none

RESULTS

`copy` copy of the URL object

11.2 url:Free

NAME

url:Free – free URL object (V2.0)

SYNOPSIS

```
url:Free()
```

FUNCTION

url:Free() frees the URL object. After that you can't use it any longer.

INPUTS

none

11.3 url:GetFragment

NAME

url:GetFragment – get fragment part of URL (V2.0)

SYNOPSIS

```
fragment$ = url:GetFragment([flags])
```

FUNCTION

Gets the fragment part from the specified URL object. The hash sign in the URL is not part of the actual fragment contents.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`fragment$`
: fragment part of URL

11.4 url:GetHost

NAME

url:GetHost – get host part of URL (V2.0)

SYNOPSIS

```
host$ = url:GetHost([flags])
```

FUNCTION

Gets the host name. If it is an IPv6 numeric address, the zone id will not be part of it but can be retrieved separately using url:GetZoneID(). IPv6 numerical addresses are returned within brackets ([]).

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`host$` host part of URL

11.5 url:GetOptions

NAME

url:GetOptions – get options part of URL (V2.0)

SYNOPSIS

```
opts$ = url:GetOptions([flags])
```

FUNCTION

Gets the options from the specified URL object. The options field is an optional field that might follow the password in the userinfo part. It is only recognized/used when parsing URLs for the following schemes: pop3, smtp and imap. This function however allows users to independently get this field at will.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`opts$` options part of URL

11.6 url:GetPassword

NAME

url:GetPassword – get password part of URL (V2.0)

SYNOPSIS

```
pwd$ = url:GetPassword([flags])
```

FUNCTION

Gets the password from the specified URL object.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`pwd$` password part of URL

11.7 url:GetPath

NAME

url:GetPath – get path part of URL (V2.0)

SYNOPSIS

```
path$ = url:GetPath([flags])
```

FUNCTION

Gets the path from the specified URL object. The part will be '/' even if no path is supplied in the URL. A URL path always starts with a slash.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`path$` path part of URL

11.8 url:GetPort

NAME

url:GetPort – get port part of URL (V2.0)

SYNOPSIS

```
port$ = url:GetPort([flags])
```

FUNCTION

Gets the port for the URL. The given port number will be provided as a string and the decimal number will be between 1 and 65535.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`port$` port part of URL

11.9 url:GetQuery

NAME

url:GetQuery – get query part of URL (V2.0)

SYNOPSIS

```
query$ = url:GetQuery([flags])
```

FUNCTION

Gets the query from the specified URL object. The initial question mark that denotes the beginning of the query part is a delimiter only. It is not part of the query contents. A not-present query will result in `Nil` being returned. A zero-length query will result in a zero-length string.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`query$` query part of URL

11.10 url:GetScheme

NAME

url:GetScheme – get scheme part of URL (V2.0)

SYNOPSIS

```
scheme$ = url:GetScheme([flags])
```

FUNCTION

Gets the scheme from the specified URL object.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`scheme$` scheme part of URL

11.11 url:GetURL

NAME

url:GetURL – get URL part of URL (V2.0)

SYNOPSIS

```
url$ = url:GetURL([flags])
```

FUNCTION

Gets the URL from the specified URL object. When asked to return the full URL, curl will return a normalized and possibly cleaned up version of what was previously parsed. We advise using the `#CURLU_PUNYCODE` option to get the URL as "normalized" as possible since IDN allows host names to be written in many different ways that still end up the same punycode version.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`url$` URL part of URL

11.12 url:GetUser

NAME

url:GetUser – get user part of URL (V2.0)

SYNOPSIS

```
user$ = url:GetUser([flags])
```

FUNCTION

Gets the user from the specified URL object.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`user$` user part of URL

11.13 url:GetZoneID

NAME

url:GetZoneID – get zone ID part of URL (V2.0)

SYNOPSIS

```
zoneid$ = url:GetZoneID([flags])
```

FUNCTION

Gets the zone ID from the specified URL object.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`flags` optional: combination of flags

RESULTS

`zoneid$` zone ID part of URL

11.14 url:SetFragment

NAME

url:SetFragment – get fragment part of URL (V2.0)

SYNOPSIS

```
url:SetFragment(fragment$[, flags])
```

FUNCTION

Sets the fragment part in the specified URL object to the one specified in `fragment$`. The hash sign in the URL is not part of the actual fragment contents. You can also pass `Nil` to unset the fragment in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`fragment$`
fragment to get

`flags` optional: combination of flags

11.15 url:SetHost

NAME

url:SetHost – get host part of URL (V2.0)

SYNOPSIS

```
url:SetHost(host$[, flags])
```

FUNCTION

Sets the host in the specified URL object to the one specified in `host$`. If it is IDNA the string must then be encoded as your locale says or UTF-8 (when WinIDN is used). If it is a bracketed IPv6 numeric address it may contain a zone id (or you can use url:SetZoneID). You can also pass `Nil` to unset the host in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

Unless `#CURLU_NO_AUTHORITY` is get, a blank host name is not allowed to get.

INPUTS

`host$` host to get

`flags` optional: combination of flags

11.16 url:SetOptions

NAME

url:SetOptions – get options part of URL (V2.0)

SYNOPSIS

```
url:SetOptions(opts$[, flags])
```

FUNCTION

Sets the options in the specified URL object to the one specified in `opts$`. The options field is an optional field that might follow the password in the userinfo part. It is only recognized/used when parsing URLs for the following schemes: pop3, smtp and imap. This function however allows users to independently get this field at will. You can also pass `Nil` to unset the options field in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`opts$` options to get

`flags` optional: combination of flags

11.17 url:SetPassword

NAME

url:SetPassword – get password part of URL (V2.0)

SYNOPSIS

```
url:SetPassword(pwd$[, flags])
```

FUNCTION

Sets the password in the specified URL object to the one specified in `pwd$`. You can also pass `Nil` to unset the password in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`pwd$` password to get

`flags` optional: combination of flags

11.18 url:SetPath

NAME

url:SetPath – get path part of URL (V2.0)

SYNOPSIS

`url:SetPath(path$[, flags])`

FUNCTION

Sets the path in the specified URL object to the one specified in `path$`. If a path is get in the URL without a leading slash, a slash will be inserted automatically when this URL is read from the handle. You can also pass `Nil` to unset the user in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`path$`	path to get
`flags`	optional: combination of flags

11.19 url:SetPort

NAME

url:SetPort – get port part of URL (V2.0)

SYNOPSIS

`url:SetPort(port$[, flags])`

FUNCTION

Sets the port for the URL. The given port number must be provided as a string and the decimal number must be between 1 and 65535. Anything else will return an error. You can also pass `Nil` to unset the port in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`port$`	port to get
`flags`	optional: combination of flags

11.20 url:SetQuery

NAME

url:SetQuery – get query part of URL (V2.0)

SYNOPSIS

`url:SetQuery(query$[, flags])`

FUNCTION

Sets the query in the specified URL object to the one specified in `query$`. The query part will also get spaces converted to pluses when asked to URL encode on get with the `#CURLU_URLENCODE` bit. If used together with the `#CURLU_APPENDQUERY` bit, the provided part is appended on the end of the existing query. The question mark in the URL is not part of the actual query contents.

You can also pass `Nil` to unset the query in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`query$`	query to get
`flags`	optional: combination of flags

11.21 url:SetScheme

NAME

url:SetScheme – get scheme part of URL (V2.0)

SYNOPSIS

```
url:SetScheme(scheme$[, flags])
```

FUNCTION

Sets the scheme in the specified URL object to the one specified in `scheme$`. libcurl only accepts setting schemes up to 40 bytes long. You can also pass `Nil` to unset the scheme in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`scheme$`	scheme to get
`flags`	optional: combination of flags

11.22 url:SetURL

NAME

url:SetURL – get URL part of URL (V2.0)

SYNOPSIS

```
url:SetURL(url$[, flags])
```

FUNCTION

Allows the full URL of the handle to be replaced. If the handle already is populated with a URL, the new URL can be relative to the previous.

When successfully setting a new URL, relative or absolute, the handle contents will be replaced with the information of the newly get URL.

Pass a string to the `url$` parameter. The string must point to a correctly formatted RFC 3986+ URL.

You can also pass `Nil` to unset the URL in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

Unless `#CURLU_NO_AUTHORITY` is get, a blank host name is not allowed in the URL.

INPUTS

`url$` URL to get

`flags` optional: combination of flags

11.23 url:SetUser

NAME

url:SetUser – get user part of URL (V2.0)

SYNOPSIS

```
url:SetUser(user$[, flags])
```

FUNCTION

Sets the user in the specified URL object to the one specified in `user$`. You can also pass `Nil` to unset the user in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`user$` user to get

`flags` optional: combination of flags

11.24 url:SetZoneID

NAME

url:SetZoneID – get zone ID part of URL (V2.0)

SYNOPSIS

```
url:SetZoneID(zoneid$[, flags])
```

FUNCTION

If the host name is a numeric IPv6 address, this field can also be get. You can also pass `Nil` to unset the zone ID in the URL handle.

The optional `flags` parameter can be a combination of flags as described in the documentation of the hurl.URL() function. See Section 4.5 [hurl.URL], page 13, for details.

INPUTS

`zoneid$` zone ID to get

`flags` optional: combination of flags

Appendix A Licenses

A.1 Curl license

Copyright (c) 1996 - 2020, Daniel Stenberg, <daniel@haxx.se>, and many contributors, see the THANKS file.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

A.2 LuaCurl license

Copyright (c) 2014-2017 Alexey Melnichuk

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.3 OpenSSL license

The OpenSSL toolkit stays under a double license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license

texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

Copyright (c) 1998-2018 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution,

be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

A.4 libssh2 license

Copyright (c) 2004-2007 Sara Golemon <sarag@libssh2.org>

Copyright (c) 2005,2006 Mikhail Gusarov <dottedmag@dottedmag.net>

Copyright (c) 2006-2007 The Written Word, Inc.

Copyright (c) 2007 Eli Fant <elifantu@mail.ru>

Copyright (c) 2009-2014 Daniel Stenberg

Copyright (c) 2008, 2009 Simon Josefsson

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the copyright holder nor the names of any other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index

E

easy:Close ... 19
easy:Escape ... 19
easy:GetInfo ... 20
easy:GetInfo_AppConnect_Time ... 25
easy:GetInfo_AppConnect_Time_t ... 25
easy:GetInfo_CAInfo ... 26
easy:GetInfo_CAPath ... 26
easy:GetInfo_CertInfo ... 26
easy:GetInfo_Condition_Unmet ... 27
easy:GetInfo_Connect_Time ... 27
easy:GetInfo_Connect_Time_t ... 28
easy:GetInfo_Content_Length_Download ... 28
easy:GetInfo_Content_Length_Download_t ... 28
easy:GetInfo_Content_Length_Upload ... 29
easy:GetInfo_Content_Length_Upload_t ... 29
easy:GetInfo_Content_Type ... 30
easy:GetInfo_CookieList ... 30
easy:GetInfo_Effective_Method ... 30
easy:GetInfo_Effective_URL ... 31
easy:GetInfo_FileTime ... 31
easy:GetInfo_FTP_Entry_Path ... 32
easy:GetInfo_Header_Size ... 32
easy:GetInfo_HTTP_ConnectCode ... 33
easy:GetInfo_HTTP_Version ... 33
easy:GetInfo_HTTPAuth_Avail ... 32
easy:GetInfo_LastSocket ... 34
easy:GetInfo_Local_IP ... 34
easy:GetInfo_Local_Port ... 34
easy:GetInfo_NameLookup_Time ... 35
easy:GetInfo_NameLookup_Time_t ... 35
easy:GetInfo_Num_Connects ... 36
easy:GetInfo_OS_ErrNo ... 36
easy:GetInfo_PreTransfer_Time ... 36
easy:GetInfo_PreTransfer_Time_t ... 37
easy:GetInfo_Primary_IP ... 37
easy:GetInfo_Primary_Port ... 38
easy:GetInfo_Protocol ... 38
easy:GetInfo_Proxy_Error ... 39
easy:GetInfo_Proxy_SSL_VerifyResult ... 40
easy:GetInfo_ProxyAuth_Avail ... 39
easy:GetInfo_Redirect_Count ... 40
easy:GetInfo_Redirect_Time ... 40
easy:GetInfo_Redirect_Time_t ... 41
easy:GetInfo_Redirect_URL ... 41
easy:GetInfo_Referer ... 42
easy:GetInfo_Request_Size ... 42
easy:GetInfo_Response_Code ... 42
easy:GetInfo_Retry_After ... 43
easy:GetInfo_RTSP_Client_CSeq ... 43
easy:GetInfo_RTSP_CSeq_Recv ... 43
easy:GetInfo_RTSP_Server_CSeq ... 44
easy:GetInfo_RTSP_Session_ID ... 44
easy:GetInfo_Scheme ... 45
easy:GetInfo_Size_Download ... 45
easy:GetInfo_Size_Download_t ... 45
easy:GetInfo_Size_Upload ... 46
easy:GetInfo_Size_Upload_t ... 46
easy:GetInfo_Speed_Download ... 47
easy:GetInfo_Speed_Download_t ... 47
easy:GetInfo_Speed_Upload ... 47
easy:GetInfo_Speed_Upload_t ... 48
easy:GetInfo_SSL_Engines ... 48
easy:GetInfo_SSL_VerifyResult ... 49
easy:GetInfo_StartTransfer_Time ... 49
easy:GetInfo_StartTransfer_Time_t ... 49
easy:GetInfo_Total_Time ... 50
easy:GetInfo_Total_Time_t ... 50
easy:Mime ... 51
easy:Pause ... 51
easy:Perform ... 52
easy:Recv ... 53
easy:Reset ... 53
easy:Send ... 54
easy:SetOpt ... 54
easy:SetOpt_Abstract_Unix_Socket ... 74
easy:SetOpt_Accept_Encoding ... 75
easy:SetOpt_AcceptTimeout_MS ... 76
easy:SetOpt_Address_Scope ... 76
easy:SetOpt_AltSvc ... 76
easy:SetOpt_AltSvc_Ctrl ... 77
easy:SetOpt_Append ... 78
easy:SetOpt_AutoReferer ... 78
easy:SetOpt_AWS_SigV4 ... 78
easy:SetOpt_BufferSize ... 79
easy:SetOpt_CA_Cache_Timeout ... 79
easy:SetOpt_CAInfo ... 80
easy:SetOpt_CAInfo_Blob ... 81
easy:SetOpt_CAPath ... 81
easy:SetOpt_CertInfo ... 81
easy:SetOpt_Chunk_BGN_Function ... 82
easy:SetOpt_Chunk_End_Function ... 83
easy:SetOpt_Connect_Only ... 83
easy:SetOpt_Connect_To ... 85
easy:SetOpt_ConnectTimeout ... 84
easy:SetOpt_ConnectTimeout_MS ... 84
easy:SetOpt_Cookie ... 86
easy:SetOpt_CookieFile ... 86
easy:SetOpt_CookieJar ... 87
easy:SetOpt_CookieList ... 88
easy:SetOpt_CookieSession ... 88
easy:SetOpt_CRLF ... 89
easy:SetOpt_CRLFile ... 89
easy:SetOpt_CURLU ... 90
easy:SetOpt_CustomRequest ... 90
easy:SetOpt_DebugFunction ... 91
easy:SetOpt_Default_Protocol ... 92
easy:SetOpt_DirListOnly ... 93
easy:SetOpt_Disallow_Username_In_URL ... 94

easy:SetOpt_DNS_Cache_Timeout 94
easy:SetOpt_DNS_Interface 94
easy:SetOpt_DNS_Local_IP4 95
easy:SetOpt_DNS_Local_IP6 95
easy:SetOpt_DNS_Servers 95
easy:SetOpt_DNS_Shuffle_Addresses 96
easy:SetOpt_DNS_Use_Global_Cache 96
easy:SetOpt_DoH_SSL_VerifyHost 97
easy:SetOpt_DoH_SSL_VerifyPeer 97
easy:SetOpt_DoH_SSL_VerifyStatus 98
easy:SetOpt_DoH_URL 99
easy:SetOpt_EGDSocket 99
easy:SetOpt_Expect_100_Timeout_MS 99
easy:SetOpt_FailOnError 100
easy:SetOpt_FileTime 100
easy:SetOpt_FNMatch_Function 101
easy:SetOpt_FollowLocation 101
easy:SetOpt_Forbid_Reuse 102
easy:SetOpt_Fresh_Connect 102
easy:SetOpt_FTP_Account 103
easy:SetOpt_FTP_Alternative_To_User 103
easy:SetOpt_FTP_Create_Missing_Dirs 103
easy:SetOpt_FTP_FileMethod 104
easy:SetOpt_FTP_Response_Timeout 105
easy:SetOpt_FTP_Skip_PASV_IP 105
easy:SetOpt_FTP_SSL_CCC 106
easy:SetOpt_FTP_Use_Eprt 107
easy:SetOpt_FTP_Use_Epsv 107
easy:SetOpt_FTP_Use_Pret 108
easy:SetOpt_FTPPort 104
easy:SetOpt_FTPSSLAuth 106
easy:SetOpt_GSSAPI_Delegation 108
easy:SetOpt_Happy_Eyeballs_Timeout_MS 108
easy:SetOpt_HAProxyProtocol 109
easy:SetOpt_Header 109
easy:SetOpt_HeaderFunction 110
easy:SetOpt_HeaderOpt 111
easy:SetOpt_HSTS 111
easy:SetOpt_HSTS_Ctrl 112
easy:SetOpt_HSTSReadFunction 112
easy:SetOpt_HSTSWriteFunction 113
easy:SetOpt_HTTP_Content_Decoding 117
easy:SetOpt_HTTP_Transfer_Decoding 119
easy:SetOpt_HTTP_Version 120
easy:SetOpt_HTTP09_Allowed 114
easy:SetOpt_HTTP200Aliases 114
easy:SetOpt_HTTPAuth 115
easy:SetOpt_HTTPGet 117
easy:SetOpt_HTTPHeader 117
easy:SetOpt_HTTPPost 118
easy:SetOpt_HTTPProxyTunnel 119
easy:SetOpt_Ignore_Content_Length 121
easy:SetOpt_InFileSize 121
easy:SetOpt_InFileSize_Large 122
easy:SetOpt_Interface 122
easy:SetOpt_IPResolve 122
easy:SetOpt_IssuerCert 123
easy:SetOpt_IssuerCert_Blob 123
easy:SetOpt_Keep_Sending_On_Error 124
easy:SetOpt_KeyPasswd 124
easy:SetOpt_KRBLevel 125
easy:SetOpt_LocalPort 125
easy:SetOpt_LocalPortRange 125
easy:SetOpt_Login_Options 126
easy:SetOpt_Low_Speed_Limit 126
easy:SetOpt_Low_Speed_Time 127
easy:SetOpt_Mail_Auth 127
easy:SetOpt_Mail_From 127
easy:SetOpt_Mail_RCPT 128
easy:SetOpt_Mail_RCPT_AllowFails 128
easy:SetOpt_Max_Recv_Speed_Large 131
easy:SetOpt_Max_Send_Speed_Large 132
easy:SetOpt_MaxAge_Conn 129
easy:SetOpt_MaxConnects 129
easy:SetOpt_MaxFileSize 130
easy:SetOpt_MaxFileSize_Large 130
easy:SetOpt_MaxLifeTime_Conn 131
easy:SetOpt_MaxRedirs 131
easy:SetOpt_MIME_Options 132
easy:SetOpt_MIMEPost 133
easy:SetOpt_Netrc 133
easy:SetOpt_Netrc_File 134
easy:SetOpt_New_Directory_Perms 134
easy:SetOpt_New_File_Perms 135
easy:SetOpt_Nobody 135
easy:SetOpt_NoProgress 135
easy:SetOpt_NoProxy 136
easy:SetOpt_NoSignal 136
easy:SetOpt_Password 137
easy:SetOpt_Path_As_Is 137
easy:SetOpt_PinnedPublicKey 138
easy:SetOpt_PipeWait 138
easy:SetOpt_Port 139
easy:SetOpt_Post 139
easy:SetOpt_PostFields 140
easy:SetOpt_PostQuote 141
easy:SetOpt_PostRedir 141
easy:SetOpt_Pre_Proxy 142
easy:SetOpt_Prequote 142
easy:SetOpt_PreReqFunction 143
easy:SetOpt_ProgressFunction 144
easy:SetOpt_Protocols 144
easy:SetOpt_Protocols_Str 145
easy:SetOpt_Proxy 147
easy:SetOpt_Proxy_CAInfo 148
easy:SetOpt_Proxy_CAInfo_Blob 149
easy:SetOpt_Proxy_CAPath 149
easy:SetOpt_Proxy_CRLFile 149
easy:SetOpt_Proxy_IssuerCert 150
easy:SetOpt_Proxy_IssuerCert_Blob 151
easy:SetOpt_Proxy_KeyPasswd 152
easy:SetOpt_Proxy_PinnedPublicKey 152
easy:SetOpt_Proxy_Service_Name 153
easy:SetOpt_Proxy_SSL_Cipher_List 155
easy:SetOpt_Proxy_SSL_Options 156
easy:SetOpt_Proxy_SSL_VerifyHost 157

easy:SetOpt_Proxy_SSL_VerifyPeer 158
easy:SetOpt_Proxy_SSLCert 153
easy:SetOpt_Proxy_SSLCert_Blob 154
easy:SetOpt_Proxy_SSLCertType 154
easy:SetOpt_Proxy_SSLKey 155
easy:SetOpt_Proxy_SSLKey_Blob 156
easy:SetOpt_Proxy_SSLKeyType 156
easy:SetOpt_Proxy_SSLVersion 158
easy:SetOpt_Proxy_TLSAuth_Password 159
easy:SetOpt_Proxy_TLSAuth_Type 160
easy:SetOpt_Proxy_TLSAuth_UserName 160
easy:SetOpt_Proxy_Transfer_Mode 161
easy:SetOpt_ProxyAuth 148
easy:SetOpt_ProxyHeader 150
easy:SetOpt_ProxyPassword 152
easy:SetOpt_ProxyPort 153
easy:SetOpt_ProxyType 161
easy:SetOpt_ProxyUserName 162
easy:SetOpt_ProxyUserPwd 162
easy:SetOpt_Put 162
easy:SetOpt_Quick_Exit 163
easy:SetOpt_Quote 163
easy:SetOpt_Random_File 164
easy:SetOpt_Range 165
easy:SetOpt_ReadFunction 165
easy:SetOpt_Redir_Protocols 166
easy:SetOpt_Redir_Protocols_Str 168
easy:SetOpt_Referer 169
easy:SetOpt_Request_Target 169
easy:SetOpt_Resolve 169
easy:SetOpt_Resolver_Start_Function 170
easy:SetOpt_Resume_From 171
easy:SetOpt_Resume_From_Large 171
easy:SetOpt_RTSP_Client_CSeq 172
easy:SetOpt_RTSP_Request 172
easy:SetOpt_RTSP_Server_CSeq 174
easy:SetOpt_RTSP_Session_ID 174
easy:SetOpt_RTSP_Stream_URI 174
easy:SetOpt_RTSP_Transport 175
easy:SetOpt_SASL_AuthZID 175
easy:SetOpt_SASL_IR 176
easy:SetOpt_SeekFunction 176
easy:SetOpt_Service_Name 177
easy:SetOpt_Share 177
easy:SetOpt_Socks5_Auth 178
easy:SetOpt_Socks5_GSSAPI_NEC 178
easy:SetOpt_Socks5_GSSAPI_Service 179
easy:SetOpt_SSH_Auth_Types 179
easy:SetOpt_SSH_Compression 179
easy:SetOpt_SSH_Host_Public_Key_MD5 180
easy:SetOpt_SSH_HostKeyFunction 180
easy:SetOpt_SSH_KnownHosts 181
easy:SetOpt_SSH_Private_KeyFile 181
easy:SetOpt_SSH_Public_KeyFile 181
easy:SetOpt_SSL_Cipher_List 183
easy:SetOpt_SSL_EC_Curves 184
easy:SetOpt_SSL_Enable_Alpn 184
easy:SetOpt_SSL_Enable_Npn 184
easy:SetOpt_SSL_FalseStart 185
easy:SetOpt_SSL_Options 187
easy:SetOpt_SSL_SessionID_Cache 188
easy:SetOpt_SSL_VerifyHost 188
easy:SetOpt_SSL_VerifyPeer 189
easy:SetOpt_SSL_VerifyStatus 190
easy:SetOpt_SSLCert 182
easy:SetOpt_SSLCert_Blob 182
easy:SetOpt_SSLCertType 183
easy:SetOpt_SSLEngine 185
easy:SetOpt_SSLEngine_Default 185
easy:SetOpt_SSLKey 186
easy:SetOpt_SSLKey_Blob 186
easy:SetOpt_SSLKeyType 187
easy:SetOpt_SSLVersion 190
easy:SetOpt_Stream_Depends 191
easy:SetOpt_Stream_Depends_e 192
easy:SetOpt_Stream_Weight 192
easy:SetOpt_Suppress_Connect_Headers 193
easy:SetOpt_TCP_FastOpen 194
easy:SetOpt_TCP_KeepAlive 194
easy:SetOpt_TCP_KeepIdle 194
easy:SetOpt_TCP_KeepIntvl 195
easy:SetOpt_TCP_NoDelay 195
easy:SetOpt_TelnetOptions 195
easy:SetOpt_TFTP_BlkSize 196
easy:SetOpt_TFTP_No_Options 196
easy:SetOpt_TimeCondition 197
easy:SetOpt_Timeout 197
easy:SetOpt_Timeout_MS 198
easy:SetOpt_TimeValue 198
easy:SetOpt_TimeValue_Large 199
easy:SetOpt_TLS13_Ciphers 199
easy:SetOpt_TLSAuth_Password 199
easy:SetOpt_TLSAuth_Type 200
easy:SetOpt_TLSAuth_UserName 200
easy:SetOpt_TrailerFunction 200
easy:SetOpt_Transfer_Encoding 201
easy:SetOpt_TransferText 202
easy:SetOpt_Unix_Socket_Path 202
easy:SetOpt_Unrestricted_Auth 203
easy:SetOpt_Upkeep_Interval_MS 203
easy:SetOpt_Upload 203
easy:SetOpt_Upload_Buffersize 204
easy:SetOpt_URL 204
easy:SetOpt_Use_SSL 210
easy:SetOpt_UserAgent 209
easy:SetOpt_UserName 209
easy:SetOpt_UserPwd 210
easy:SetOpt_Verbose 211
easy:SetOpt_WildcardMatch 211
easy:SetOpt_WriteFunction 212
easy:SetOpt_WS_Options 213
easy:SetOpt_XOAuth2_Bearer 214
easy:Unescape 214
easy:UnsetOpt 215
easy:UnsetOpt_Abstract_Unix_Socket 234
easy:UnsetOpt_Accept_Encoding 235

easy:UnsetOpt_AcceptTimeout_MS 235
easy:UnsetOpt_Address_Scope 235
easy:UnsetOpt_AltSvc 235
easy:UnsetOpt_AltSvc_Ctrl 236
easy:UnsetOpt_Append 236
easy:UnsetOpt_AutoReferer 236
easy:UnsetOpt_AWS_SigV4 237
easy:UnsetOpt_BufferSize 237
easy:UnsetOpt_CA_Cache_Timeout 237
easy:UnsetOpt_CAInfo 237
easy:UnsetOpt_CAInfo_Blob 238
easy:UnsetOpt_CAPath 238
easy:UnsetOpt_CertInfo 238
easy:UnsetOpt_Chunk_BGN_Function 239
easy:UnsetOpt_Chunk_End_Function 239
easy:UnsetOpt_Connect_Only 239
easy:UnsetOpt_Connect_To 240
easy:UnsetOpt_ConnectTimeout 239
easy:UnsetOpt_ConnectTimeout_MS 240
easy:UnsetOpt_Cookie 240
easy:UnsetOpt_CookieFile 241
easy:UnsetOpt_CookieJar 241
easy:UnsetOpt_CookieList 241
easy:UnsetOpt_CookieSession 241
easy:UnsetOpt_CRLF 242
easy:UnsetOpt_CRLFile 242
easy:UnsetOpt_CURLU 242
easy:UnsetOpt_CustomRequest 243
easy:UnsetOpt_DebugFunction 243
easy:UnsetOpt_Default_Protocol 243
easy:UnsetOpt_DirListOnly 243
easy:UnsetOpt_Disallow_Username_In_URL ... 244
easy:UnsetOpt_DNS_Cache_Timeout 244
easy:UnsetOpt_DNS_Interface 244
easy:UnsetOpt_DNS_Local_IP4 245
easy:UnsetOpt_DNS_Local_IP6 245
easy:UnsetOpt_DNS_Servers 245
easy:UnsetOpt_DNS_Shuffle_Addresses 245
easy:UnsetOpt_DNS_Use_Global_Cache 246
easy:UnsetOpt_DoH_SSL_VerifyHost 246
easy:UnsetOpt_DoH_SSL_VerifyPeer 246
easy:UnsetOpt_DoH_SSL_VerifyStatus 247
easy:UnsetOpt_DoH_URL 247
easy:UnsetOpt_EGDSocket 247
easy:UnsetOpt_Expect_100_Timeout_MS 247
easy:UnsetOpt_FailOnError 248
easy:UnsetOpt_FileTime 248
easy:UnsetOpt_FNMatch_Function 248
easy:UnsetOpt_FollowLocation 249
easy:UnsetOpt_Forbid_Reuse 249
easy:UnsetOpt_Fresh_Connect 249
easy:UnsetOpt_FTP_Account 249
easy:UnsetOpt_FTP_Alternative_To_User 250
easy:UnsetOpt_FTP_Create_Missing_Dirs 250
easy:UnsetOpt_FTP_FileMethod 250
easy:UnsetOpt_FTP_Response_Timeout 251
easy:UnsetOpt_FTP_Skip_PASV_IP 251
easy:UnsetOpt_FTP_SSL_CCC 252
easy:UnsetOpt_FTP_Use_Eprt 252
easy:UnsetOpt_FTP_Use_Epsv 252
easy:UnsetOpt_FTP_Use_Pret 253
easy:UnsetOpt_FTPPort 251
easy:UnsetOpt_FTPSSLAuth 251
easy:UnsetOpt_GSSAPI_Delegation 253
easy:UnsetOpt_Happy_
Eyeballs_Timeout_MS 253
easy:UnsetOpt_HAProxyProtocol 253
easy:UnsetOpt_Header 254
easy:UnsetOpt_HeaderFunction 254
easy:UnsetOpt_HeaderOpt 254
easy:UnsetOpt_HSTS 255
easy:UnsetOpt_HSTS_Ctrl 255
easy:UnsetOpt_HSTSReadFunction 255
easy:UnsetOpt_HSTSWriteFunction 255
easy:UnsetOpt_HTTP_Content_Decoding 257
easy:UnsetOpt_HTTP_Transfer_Decoding 258
easy:UnsetOpt_HTTP_Version 258
easy:UnsetOpt_HTTP09_Allowed 256
easy:UnsetOpt_HTTP200Aliases 256
easy:UnsetOpt_HTTPAuth 256
easy:UnsetOpt_HTTPGet 257
easy:UnsetOpt_HTTPHeader 257
easy:UnsetOpt_HTTPPost 257
easy:UnsetOpt_HTTPProxyTunnel 258
easy:UnsetOpt_Ignore_Content_Length 259
easy:UnsetOpt_InFileSize 259
easy:UnsetOpt_InFileSize_Large 259
easy:UnsetOpt_Interface 259
easy:UnsetOpt_IPResolve 260
easy:UnsetOpt_IssuerCert 260
easy:UnsetOpt_IssuerCert_Blob 260
easy:UnsetOpt_Keep_Sending_On_Error 261
easy:UnsetOpt_KeyPasswd 261
easy:UnsetOpt_KRBLevel 261
easy:UnsetOpt_LocalPort 261
easy:UnsetOpt_LocalPortRange 262
easy:UnsetOpt_Login_Options 262
easy:UnsetOpt_Low_Speed_Limit 262
easy:UnsetOpt_Low_Speed_Time 263
easy:UnsetOpt_Mail_Auth 263
easy:UnsetOpt_Mail_From 263
easy:UnsetOpt_Mail_RCPT 263
easy:UnsetOpt_Mail_RCPT_AllowFails 264
easy:UnsetOpt_Max_Recv_Speed_Large 265
easy:UnsetOpt_Max_Send_Speed_Large 266
easy:UnsetOpt_MaxAge_Conn 264
easy:UnsetOpt_MaxConnects 264
easy:UnsetOpt_MaxFileSize 265
easy:UnsetOpt_MaxFileSize_Large 265
easy:UnsetOpt_MaxLifeTime_Conn 265
easy:UnsetOpt_MaxRedirs 266
easy:UnsetOpt_MIME_Options 266
easy:UnsetOpt_MIMEPost 267
easy:UnsetOpt_Netrc 267
easy:UnsetOpt_Netrc_File 267
easy:UnsetOpt_New_Directory_Perms 267

easy:UnsetOpt_New_File_Perms 268
easy:UnsetOpt_Nobody 268
easy:UnsetOpt_NoProgress 268
easy:UnsetOpt_NoProxy 269
easy:UnsetOpt_NoSignal 269
easy:UnsetOpt_Password 269
easy:UnsetOpt_Path_As_Is 269
easy:UnsetOpt_PinnedPublicKey 270
easy:UnsetOpt_PipeWait 270
easy:UnsetOpt_Port 270
easy:UnsetOpt_Post 271
easy:UnsetOpt_PostFields 271
easy:UnsetOpt_PostQuote 271
easy:UnsetOpt_PostRedir 271
easy:UnsetOpt_Pre_Proxy 272
easy:UnsetOpt_Prequote 272
easy:UnsetOpt_PreReqFunction 272
easy:UnsetOpt_ProgressFunction 273
easy:UnsetOpt_Protocols 273
easy:UnsetOpt_Protocols_Str 273
easy:UnsetOpt_Proxy 273
easy:UnsetOpt_Proxy_CAInfo 274
easy:UnsetOpt_Proxy_CAInfo_Blob 274
easy:UnsetOpt_Proxy_CAPath 275
easy:UnsetOpt_Proxy_CRLFile 275
easy:UnsetOpt_Proxy_IssuerCert 275
easy:UnsetOpt_Proxy_IssuerCert_Blob 276
easy:UnsetOpt_Proxy_KeyPasswd 276
easy:UnsetOpt_Proxy_PinnedPublicKey 277
easy:UnsetOpt_Proxy_Service_Name 277
easy:UnsetOpt_Proxy_SSL_Cipher_List 278
easy:UnsetOpt_Proxy_SSL_Options 279
easy:UnsetOpt_Proxy_SSL_VerifyHost 280
easy:UnsetOpt_Proxy_SSL_VerifyPeer 280
easy:UnsetOpt_Proxy_SSLCert 277
easy:UnsetOpt_Proxy_SSLCert_Blob 278
easy:UnsetOpt_Proxy_SSLCertType 278
easy:UnsetOpt_Proxy_SSLKey 279
easy:UnsetOpt_Proxy_SSLKey_Blob 279
easy:UnsetOpt_Proxy_SSLKeyType 279
easy:UnsetOpt_Proxy_SSLVersion 280
easy:UnsetOpt_Proxy_TLSAuth_Password 281
easy:UnsetOpt_Proxy_TLSAuth_Type 281
easy:UnsetOpt_Proxy_TLSAuth_UserName 281
easy:UnsetOpt_Proxy_Transfer_Mode 281
easy:UnsetOpt_ProxyAuth 274
easy:UnsetOpt_ProxyHeader 275
easy:UnsetOpt_ProxyPassword 276
easy:UnsetOpt_ProxyPort 277
easy:UnsetOpt_ProxyType 282
easy:UnsetOpt_ProxyUserName 282
easy:UnsetOpt_ProxyUserPwd 282
easy:UnsetOpt_Put 283
easy:UnsetOpt_Quick_Exit 283
easy:UnsetOpt_Quote 283
easy:UnsetOpt_Random_File 283
easy:UnsetOpt_Range 284
easy:UnsetOpt_ReadFunction 284
easy:UnsetOpt_Redir_Protocols 284
easy:UnsetOpt_Redir_Protocols_Str 285
easy:UnsetOpt_Referer 285
easy:UnsetOpt_Request_Target 285
easy:UnsetOpt_Resolve 285
easy:UnsetOpt_Resolver_Start_Function 286
easy:UnsetOpt_Resume_From 286
easy:UnsetOpt_Resume_From_Large 286
easy:UnsetOpt_RTSP_Client_CSeq 287
easy:UnsetOpt_RTSP_Request 287
easy:UnsetOpt_RTSP_Server_CSeq 287
easy:UnsetOpt_RTSP_Session_ID 287
easy:UnsetOpt_RTSP_Stream_URI 288
easy:UnsetOpt_RTSP_Transport 288
easy:UnsetOpt_SASL_AuthZID 288
easy:UnsetOpt_SASL_IR 289
easy:UnsetOpt_SeekFunction 289
easy:UnsetOpt_Service_Name 289
easy:UnsetOpt_Share 289
easy:UnsetOpt_Socks5_Auth 290
easy:UnsetOpt_Socks5_GSSAPI_NEC 290
easy:UnsetOpt_Socks5_GSSAPI_Service 290
easy:UnsetOpt_SSH_Auth_Types 291
easy:UnsetOpt_SSH_Compression 291
easy:UnsetOpt_SSH_Host_Public_Key_MD5 291
easy:UnsetOpt_SSH_HostKeyFunction 291
easy:UnsetOpt_SSH_KnownHosts 292
easy:UnsetOpt_SSH_Private_KeyFile 292
easy:UnsetOpt_SSH_Public_KeyFile 292
easy:UnsetOpt_SSL_Cipher_List 293
easy:UnsetOpt_SSL_EC_Curves 294
easy:UnsetOpt_SSL_Enable_Alpn 294
easy:UnsetOpt_SSL_Enable_Npn 294
easy:UnsetOpt_SSL_FalseStart 295
easy:UnsetOpt_SSL_Options 296
easy:UnsetOpt_SSL_SessionID_Cache 297
easy:UnsetOpt_SSL_VerifyHost 297
easy:UnsetOpt_SSL_VerifyPeer 297
easy:UnsetOpt_SSL_VerifyStatus 297
easy:UnsetOpt_SSLCert 293
easy:UnsetOpt_SSLCert_Blob 293
easy:UnsetOpt_SSLCertType 293
easy:UnsetOpt_SSLEngine 295
easy:UnsetOpt_SSLEngine_Default 295
easy:UnsetOpt_SSLKey 295
easy:UnsetOpt_SSLKey_Blob 296
easy:UnsetOpt_SSLKeyType 296
easy:UnsetOpt_SSLVersion 298
easy:UnsetOpt_Stream_Depends 298
easy:UnsetOpt_Stream_Depends_e 298
easy:UnsetOpt_Stream_Weight 299
easy:UnsetOpt_Suppress_Connect_Headers 299
easy:UnsetOpt_TCP_FastOpen 299
easy:UnsetOpt_TCP_KeepAlive 299
easy:UnsetOpt_TCP_KeepIdle 300
easy:UnsetOpt_TCP_KeepIntvl 300
easy:UnsetOpt_TCP_NoDelay 300
easy:UnsetOpt_TelnetOptions 301

easy:UnsetOpt_TFTP_BlkSize 301
easy:UnsetOpt_TFTP_No_Options 301
easy:UnsetOpt_TimeCondition 301
easy:UnsetOpt_Timeout 302
easy:UnsetOpt_Timeout_MS 302
easy:UnsetOpt_TimeValue 302
easy:UnsetOpt_TimeValue_Large 303
easy:UnsetOpt_TLS13_Ciphers 303
easy:UnsetOpt_TLSAuth_Password 303
easy:UnsetOpt_TLSAuth_Type 303
easy:UnsetOpt_TLSAuth_UserName 304
easy:UnsetOpt_TrailerFunction 304
easy:UnsetOpt_Transfer_Encoding 304
easy:UnsetOpt_TransferText 305
easy:UnsetOpt_Unix_Socket_Path 305
easy:UnsetOpt_Unrestricted_Auth 305
easy:UnsetOpt_Upkeep_Interval_MS 305
easy:UnsetOpt_Upload 306
easy:UnsetOpt_Upload_Buffersize 306
easy:UnsetOpt_URL 306
easy:UnsetOpt_Use_SSL 307
easy:UnsetOpt_UserAgent 307
easy:UnsetOpt_UserName 307
easy:UnsetOpt_UserPwd 307
easy:UnsetOpt_Verbose 308
easy:UnsetOpt_WildcardMatch 308
easy:UnsetOpt_WriteFunction 308
easy:UnsetOpt_WS_Options 309
easy:UnsetOpt_XOAuth2_Bearer 309
easy:Upkeep 309

F

form:AddBuffer 311
form:AddContent 311
form:AddFile 312
form:AddFiles 313
form:AddStream 314
form:Free 315
form:Get 315

H

hurl.Easy 11
hurl.Form 11
hurl.Multi 12
hurl.Share 12
hurl.URL 13
hurl.Version 15
hurl.VersionInfo 16

M

mime:AddPart 317
mime:Easy 317
mime:Free 318
mimepart:Data 319
mimepart:Encoder 319
mimepart:FileData 320
mimepart:Filename 320
mimepart:Free 321
mimepart:Headers 321
mimepart:Name 321
mimepart:Subparts 321
mimepart:Type 322
multi:AddHandle 325
multi:Close 325
multi:InfoRead 326
multi:Perform 326
multi:RemoveHandle 327
multi:SetOpt 328
multi:SetOpt_Chunk_Length_Penalty_Size ... 329
multi:SetOpt_Content_
Length_Penalty_Size 329
multi:SetOpt_Max_Concurrent_Streams 330
multi:SetOpt_Max_Host_Connections 330
multi:SetOpt_Max_Pipeline_Length 331
multi:SetOpt_Max_Total_Connections 331
multi:SetOpt_MaxConnects 330
multi:SetOpt_Pipelining 332
multi:SetOpt_Pipelining_Server_Bl 333
multi:SetOpt_Pipelining_Site_Bl 333
multi:SetOpt_SocketFunction 333
multi:SetOpt_TimerFunction 334
multi:SocketAction 335
multi:Timeout 336
multi:Wait 336

S

share:Close 339
share:SetOpt 339
share:SetOpt_Share 339
share:SetOpt_Unshare 340

U

url:Dup 343
url:Free 343
url:GetFragment 343
url:GetHost 344
url:GetOptions 344
url:GetPassword 344
url:GetPath 345
url:GetPort 345
url:GetQuery 346
url:GetScheme 346
url:GetURL 346
url:GetUser 347
url:GetZoneID 347

`url:SetFragment` 348
`url:SetHost` 348
`url:SetOptions` 349
`url:SetPassword` 349
`url:SetPath` 349
`url:SetPort` 350
`url:SetQuery` 350
`url:SetScheme` 351
`url:SetURL` 351
`url:SetUser` 352
`url:SetZoneID` 352