Writing your own protocol with sockets

If you need to implement your own protocol, you can do so using LiveCode's socket support.

To understand this chapter it is assumed you understand the basics of how the Internet works, including the concepts of sockets, IP addresses and ports. More information on these concepts can be found in Wikipedia.

Tip: The standard protocols that LiveCode support such as http and ftp, discussed earlier in this chapter, have all been implemented as a scripted library with LiveCode's socket support.

You can examine this library by running `edit script of stack "revlibURL"` in the Message Box.

Beware, this library is not for the faint of heart. If you change anything, LiveCode's Internet commands may cease to operate.

Opening a connection

To open a connection use the **open socket** command.

The following command opens a connection to the IP address specified in the `tIPAddress`variable and the port specified in the `tPort`variable.

It specifies that LiveCode should send the message "`chatConnected`" when a connection has been established.

```
open socket (tIPAddress & ":" & tPort) with message "chatConnected"
```

To open a secure socket, use the **open secure socket** variant of the command.

To open a UDP datagram socket, use the **open datagram socket** variant of the command. For more information on these variants, see the *LiveCode Dictionary*.

Looking up a host name or IP address

You may look up an IP address from a host name with the **hostNameToAddress** function.

For example, to get the IP address for the livecode.com server:

```
put hostNameToAddress("www.livecode.com") into tIPAddress
```

To get the host name of the local machine, use the **hostName** function.

To look up the name from an IP address, use the **hostAddressToName** function.

Reading and writing data

Once LiveCode opens a connection, it will send a `chatConnected`message.

To receive data, use the **read from socket** command. The following message reads data from the socket and sends a `chatReceived`message when reading is completed.

```
on chatConnected pSocket
    read from socket pSocket with message chatReceived
```

```
end chatConnected
```

Once reading from the socket is completed the `chatReceived` message can be used to process or display the data.

It can then specify that it should continue to read from the socket until more data is received, sending another `chatReceived` message when done.

```
on chatReceived pSocket, pData
    put pData after field "chat output"
    read from socket pSocket with message "chatReceived"
end chatReceived
```

To write data to the socket, use the **write** command:

```
write field "chat text" to socket tSocket
```

Disconnecting

To disconnect, use the **close socket** command.

You should store a variable with details of any open sockets and close them when you have finished using them or when your stack closes.

```
close socket (tIDAddress & ":" & tPort)
```

Listening for and accepting incoming connections

To accept incoming connections on a given port, use the **accept connections** command.

The following example tells LiveCode to listen for connections on port 1987 and send the message `chatConnected`if a connection is established.

You can then start to read data from the socket in the `chatConnected`handler.

```
accept connections on port 1987 with message chatConnected
```

Handling errors

If there is an error, LiveCode will send a **socketError** message with the address of the socket and the error message.

If a socket is closed a **socketClosed** message will be sent.

If a socket times out waiting for data a **socketTimeout** message will be sent.

To get a list of sockets that are open, use the **openSockets** function.

You can set the default timeout interval by setting the **socketTimeOutInterval** property.

For more details on all of these features, see the *LiveCode Dictionary*.

Tip: You can see a complete implementation of a basic client server "chat" application by navigating to Documentation -> Getting Started -> Sample Projects -> Internet Chat – creating a custom protocol using sockets -> Launch.

Most of the scripts for the "server" stack are in the "start server" button. Most of the scripts for the client are in the stack script for the "chat client" stack.

In stack "chat server"
stack script

```
on preOpenStack
  open stack "chat client" of this stack
end preOpenStack
```

No Card Scripts

button "Start server"

```
-- declare a variable here to make it available to the entire script
local lChatterArray

-- when the mouse is clicked
on mouseUp
  -- provide visual feedback the server has been started
  disable me
  -- make it possible to stop the server
  enable button "Stop server"
  -- start accepting incoming connections
  -- the port has been chosen randomly, a high number
  -- is unlikely to be in use by anything else
  -- when a connection is received, send the message "chatConnected"
  accept connections on port 1987 with message chatConnected
end mouseUp

-- when a connection is recevied (this is first set up by mouseUp, above)
-- the "s" variable contains the address and port of the computer
-- that is connecting
on chatConnected s
```

```
  -- read in one line of data from the socket identified in the "s" variable
  read from socket s for 1 line
  -- remove any trailing return character
  put line 1 of it into tChatMessage
  -- add this new connection to the array containing a list of connections
  put tChatMessage into lChatterArray[s]
  -- call a handler to send a message to all clients informing them of the
  -- new connection
  broadcastToClients "*" & tChatMessage & " has joined the chat"
  -- put details of the new connection and a new line into the main field
  put tChatMessage && "connected" & return after field "serverstatus"
  -- start reading from the new connection contained in the "s" variable
  -- each time more data is received, call the chatMessage handler
  read from socket s with message chatMessage
end chatConnected

-- this handler is called when new data is received from a client
-- it is first set up by the chatConnected handler above
-- the variable "s" contains the host and port of the computer sending
-- the variable "data" contains the text that they sent
on chatMessage s,data
  -- put the chat message and a new line after the main field
  put data & return after field "serverstatus"
  -- send the chat message to all clients
  broadCastToClients data
  -- when more data is received from this client, send this message again
  read from socket s with message chatmessage
end chatMessage

-- this handler is called by the two handlers above
-- it sends the data contained in the "message" variable to all
-- the currently connected clients
on broadcasttoclients message
  -- get a list of all currently connected clients
  -- we add each client to this array when they connect in the handler above
  put keys(lChatterArray) into tChatterList
  -- cycle through all of the currently connected clients
  -- placing the host and port for each one into the variable "tSocket"
  repeat for each line tSocket in tChatterList
    -- send the data contained in the message variable to the client
    write message to socket tSocket
  end repeat
end broadcasttoclients

-- this message is sent when a client disconnects
-- the "s" variable contains the host and port of the client that disconnected
```

```
on socketClosed s
  -- look up the status of this client in the array we stored earlier
  put lChatterArray[s] into tChatter
  -- display this client disconnected to the main field
  put tChatter && "disconnected" & return after field "serverstatus"
  -- delete the reference to this client in the clients list array
  delete lChatterArray[s]
  -- tell all the remaining clients that this client has disconnected
  broadCastToClients "*" & tChatter && "has left"
end socketClosed
```

In Button “Stop server”

```
on mouseUp
  -- provide visual feedback that the server is stopped
  disable me
  -- make it possible to start the server again
  enable button "Start server"
  -- the openSockets contains a list of all socket connections that are open
  -- cycle through that list, putting each item in it into the variable "a"
  -- each time we go around the loop
  repeat for each line a in the opensockets
    -- close the connection contained in the variable "a"
    close socket a
  end repeat
end mouseUp
```

In button “Clear”

```
on mouseUp
  -- clear the text in the main field
  put empty into field "serverstatus"
end mouseUp
```

IN STACK “Chat Client”

```
-- declaring a variable here will make it available to the entire script
```

```
-- the lChatSocket variable contains the host and port for the connection
local lChatSocket

-- this handler is called by the mouseUp handler in the
-- script of the connect button
-- it starts the connection to the chat server
on chatConnect
  -- clear the responses field
  put empty into field "responses"
  -- prevent the user from typing while waiting for the connection to open
  disable group 1
  -- open a connection to the host address specified in the host field
  -- using port 1987, a number chosen randomly.  a high port number
  -- is unlikely to conflict with another application
  -- send a message "chatConnected" when sucessfully connected to this host
  open socket field "host" & ":1987" with message "chatConnected"
end chatConnect

-- this handler is called by the mouseUp handler in the
-- script of the connect button
-- it stops the connection to the chat server
on chatDisconnect
  -- close the connection to the host and port stored in the lChatSocket variable
  close socket lChatSocket
  -- prevent the user from typing as the connection is now closed
  disable group 1
  -- change the connect button to show we are disconnected and to allow
connecting
  set the label of button "connect" to "Connect"
end chatDisconnect

-- this message is sent when the stack is closed
on closeStack
  -- call the disconnection handler (above)
  chatDisconnect
end closeStack

-- this message handler is set up in the chatConnect handler above
-- it is called when a connection is established
-- the "s" variable contains the host and port of the server we
-- are now connected to
on chatConnected s
  -- activate the controls in group 1 so the user can type
  enable group 1
  -- change the connect button to show we are successfully
  -- connected and to allow disconnecting
```

```
  set the label of button "connect" to "Disconnect"
  -- store the host and port of the server we are now connected to
  put s into lChatSocket
  -- send the user name to the chat server so it can broadcast
  -- this to other chat clients
  write field "username" & return to socket lChatSocket
  -- specify the message to be sent whenever any data is received from
  -- the chat server connection
  read from socket s with message chatReceived
end chatConnected

-- this message is called when data is received from the chat server
-- it is first set up in the handler chatConnected above
-- the variable "s" contains the host that connected
-- the variable "data" contains the data that was sent
on chatReceived s,data
  -- display the data that was sent
  put data & return after field "responses"
  -- specify that this message is to be sent again when more data is received
  read from socket s with message chatReceived
end chatReceived

-- this message is sent automatically in the event of an error
-- the "s" variable contains the host and port connected
-- the data variable contains the error message
on socketerror s,data
  -- prevent the user typing
  disable group 1
  -- show we are disconnected now and make it possible to start
  -- a new connection
  set the label of button "connect" to "Connect"
  -- display a dialog on the screen with the error message
  answer data
end socketerror

-- this message handler is called in the mouseUp handler of the
-- send button.  the "data" variable contains the message to send
-- it sends that data to the chat server
on chatMessage data
  -- send the user name followed by the data to the chat server
  -- connection is stored in the lChatSocket variable
  write field "username" & ":" & data to socket lChatSocket
end chatMessage

NO CARD SCRIPT
```

In Button “Connect” on Chat Client

```
on mouseUp
  if the label of me is "Connect" then
    chatConnect
  else
    chatDisconnect
  end if
end mouseUp
```

in field “Chatmessage”

```
on returnInField
  -- send a mouseUp message to the send button
  -- we use "click at" instead of "send mouseUp" so that
  -- we get the visual feedback associated with clicking on the button
  click at the location of button "Send"
end returnInField

on enterInField
  -- activate the handler above
  -- this is short hand for writing out the handler again, but would save
  -- time if we ever made the handler above more complex
  returnInField
end enterInField
```

in Button “Send”

```
on mouseUp
  -- chatMessage is a message handler in the stack script
  -- send this message together with the contents of the field
  -- the user typed in
  chatMessage field "chatmessage"
  -- clear the field so the user can type another message
  put empty into field "chatmessage"
end mouseUp
```

```
edit script of stack "revlibURL"`
```

##libUrl v1.2.0 2010-09-16

# #

```
on extensionInitialize
  if the target is me then
    # MW-2015-01-28: [[ ScriptifyLibURL ]] Use the scriptified stack.
    insert the script of stack "revLibUrl" into back
    // AL-2015-01-29: [[ Scriptify revLibUrl ]] Initialise custom properties when
library is loaded
    initialiseCustomProps
  end if
end extensionInitialize

on extensionFinalize
  if the target is me then
    remove the script of stack "revLibUrl" from back
    -- CW-2016-06-11: [[ External driver support ]] Ensure external driver is
unloaded properly.
    # remove external library if in use
    unloadExternalDriver true
  end if
end extensionFinalize
-----------------------------------------------------------------------
##shared locals
local lvCount,lvBlockingUrl,lvBlockBypass,lvAuthBlockBypass,lvLogField, lvTickle
local lvJumpOut ##used by libUrlResetAll to make sure "wait for messages" loops
exit cleanly

local laLoadReq,laLoadedUrls,laStatus,laUrl,laLength,laData
local laAction,laUrlLoadStatus,laUrlErrorStatus,laLoadQ, laLoadingUrls
local laUser,laPasswd,laAuth,laBytes,laLongFileName,laHost
local laMessg,laPostData,laTemp
local laCancelled,lvStatusCallback
local laFile,laReadBytes,laWriteBytes
local laConnectHost, laConnectID, laSocketUser
local laUrlFormat

-- MM-2014-02-27: [[ HTTPS Proxy ]] Used to cache the proxy for the given URL
(rather than use global httpproxy property)
local laUrlProxy
```

```
local lvSocketToken, lvSocketOpenStart, lvSocketOpenMessageID ##for socket opening
local laSocketClosedByScript

-- MM-2014-02-27: [[ HTTPS Proxy ]] Stores if a given socket is secured.
local laSocketSecured

-- MM-2014-02-27: [[ PAC Support ]]
local lvProxyInitialized       -- If we have already attempted to extract the systems proxy settings
local lvUsePACFileForProxy     -- If we should use the configured PAC file for proxy settings
local lvHTTPProxy              -- The proxy server (if any) configured by the system
local lvProxyBypassList        -- URLs that bypass any proxy settings

-- CW-2016-06-11: [[ External driver support ]] Add support for using an external library for network functions.
local lvExtDriver

-----------------------------------------------------------------------
##http locals
local laConn,laCode,laChunk,laRhHeader, laHaveHeader, laNeedChunk
local laStatusCode, laStatusMessage, laNewLoc
local laLineNum,laTmpData
local laHttpDataDone
local laCurrentHttpHeaders
local laPostLength, laPostBytes
local lvAuthCallbacks, laServerAuthTokens, laProxyAuthTokens
local laMaxPostWithoutExpect
local laAuthRecursCount, lvNewSocketForAuth
local lvSSLVerification, laCurrentSSLVerify
local lvFollowHttpRedirects
-----------------------------------------------------------------------
##ftp locals
local lvFtpMode, lvFtpStopTime, lvDataPortCount, lvFtpListCommand
local laFtpDataDone
local laControlXDataMap ##control sockets keyed by data sockets
local laHome,lvNeedDir
local laControlXLocalMap,laTransPasvIP,laTransActvIP,laMode
local laStopUnit, laStopSec
local laFTPCommandStatus
local lvFtpCommandSocket ##socket used by libUrlFtpCommand
local laUrlByFile ##used by libUrlUploadFile to track which files go to which url

-----------------------------------------------
on libraryStack
```

```
  if the target is not me then pass libraryStack

  // AL-2015-01-29: [[ Scriptify revLibUrl ]] Initialise custom properties when stack becomes in use
  initialiseCustomProps
end libraryStack
----------------------------------------------
-- CW-2016-06-11: [[ External driver support ]] Add function to select external driver for use with libUrl.
## Set driver to use for libUrl
on libUrlSetDriver pDriver
  # Do nothing if we are setting the driver to the currently used one.
  if lvExtDriver is not pDriver then
    libUrlResetAll

    unloadExternalDriver false
    put pDriver into lvExtDriver

    if pDriver is not empty then
      return loadExternalDriver()
    else
      send "libUrlSetBehavior" to me in 0
      return empty
    end if
  else
    return empty
  end if
end libUrlSetDriver

command libUrlSetBehavior
  if the behavior of me is not lvExtDriver then
    set the behavior of me to lvExtDriver
  end if
end libUrlSetBehavior

-- CW-2016-06-11: [[ External driver support ]] Call driver specific init/remove commands if external driver is in use.
## Send initialise command to external library
private function loadExternalDriver
  local tResult

  # Add the external driver as a behavior for libURL, the script must already be loaded
  set the behavior of me to lvExtDriver

  put ulExtInitDriver() into tResult
```

```
  if tResult is not empty then
    # External driver initialisation failed, so don't use it as a behavior for libURL
    put empty into lvExtDriver
    send "libUrlSetBehavior" to me in 0
  end if

  return tResult
end loadExternalDriver

## Send shutdown command to external library
private command unloadExternalDriver pSetBehavior
  if lvExtDriver is not empty then
    ulExtRemoveDriver
    put empty into lvExtDriver
  end if

  if pSetBehavior then
    # Stop using the external driver as a behavior for libURL
    set the behavior of me to empty
  end if
end unloadExternalDriver
-----------------------------------------------
on initialiseCustomProps
  # initialise ftp codes prop set
  local tFtpCodes
  put "ABOR:225,226|Connect:220|CWD:250|DELE:250|LIST:125,150|MKD:257|
MODE:200|NLST:125,150|PASS:230,202|PASV:227|PORT:200|PWD:257|QUIT:221|
RETR:125,150|RMD:250|SIZE:213|STOR:125,150|transferComplete:226|TYPE:200|
USER:230,331" into tFtpCodes
  split tFtpCodes by "|" and ":"
  set the customProperties["cFtpGoodCodes"] of me to tFtpCodes

  # initialise default header
  set the cDefaultHeader of me to "METHOD --- HTTP/1.1" & return & \
    "Host: " & return & "User-Agent: "

  # initialise version
  set the cVersion of me to "1.2.0"

  # initalise PAC support javascript
  local tPacSupport
  put "function isPlainHostName(e){return e.indexOf('.')==-1?true:false}function
dnsDomainIs(e,t){var n=e.toLowerCase();var r=t.toLowerCase();var
i=n.substring(n.length-r.length,n.length);if(i==r)return true;return false}function
localHostOrDomainIs(e,t){var n=e.toLowerCase();var r=t.toLowerCase();return n==r||
isPlainHostName(n)&!isPlainHostName(r)?true:false}function isResolvable(e){var
```

t=dnsResolve(e);return typeof t=='string'&&t.length?true:false}function isInNet(e,t,n){var r=dnsResolve(e);if(r){var i=t.split('.');var s=n.split('.');var o=r.split('.');if(i.length==s.length&&s.length==o.length){for(var u=0;u<i.length;u++){if((i[u]&s[u])!=(s[u]&o[u]))return false}return true}}return false}function dnsResolve(e){var t;t=__dnsResolve(e);var n;n=t.split('.');if(n.length==4)return t;return null}function myIpAddress(){var e;e=__myIpAddress();var t;t=e.split('.');if(t.length==4)return e;return null}function dnsDomainLevels(e){var t=e.split('.');return t.length-1}function shExpMatch(e,t){if(typeof e!='string'||typeof t!='string')return false;if(t=='*')return true;if(e==''&&t=='')return true;e=e.toLowerCase();t=t.toLowerCase();var n=e.length;var r=t.split('*');var i=0;for(var s=0;s<r.length;s++){if(r[s]=='')continue;if(i>n)return false;i=e.indexOf(r[s]);if(i==-1)return false;i+=r[s].length;e=e.substring(i,n);n=e.length}s--;if(r[s]==''||e=='')return true;return false}function weekdayRange(e,t,n){var r=new Date;var i='SUNMONTUEWEDTHUFRISAT';e=e.toUpperCase();if(t==undefined)t=e;else t=t.toUpperCase();var s=i.indexOf(e);var o=i.indexOf(t);if(o==-1&&t=='GMT'){n=t;o=s}if(s==-1||o==-1)return false;s=s/3;o=o/3;if(n=='GMT')r=r.getUTCDay();else r=r.getDay();if(s<=o&&r>=s&&r<=o)return true;if(o<s&&(r<=o||r>=s))return true;return false}function dateRange(){var e=new Date;var t=arguments.length;var n=arguments[t-1];if(typeof n!='string')n=false;else{n=n.toUpperCase();if(n!='GMT')n=false;else{n=true;t--}}if(!t||t>6)return false;var r=0;var i=0;var s=0;var o=0;var u=0;var a=0;for(var f=0;f<t;f++){var l=arguments[f];if(typeof l=='number'){if(l>31){if(!u)u=l;else if(!a)a=l;else return false}else if(!l)return false;else if(!r)r=l;else if(!i)i=l;else return false}else if(typeof l=='string'){var c='JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC';l=l.toUpperCase();l=c.indexOf(l);if(l==-1)return false;l/=3;l+=1;if(!s)s=l;else if(!o)o=l;else return false}else return false}if(!u)u=n?e.getUTCFullYear():e.getFullYear();if(!a)a=u;if(!s)s=(n?e.getUTCMonth():e.getMonth())+1;if(!o)o=s;if(!r)r=n?e.getUTCDate():e.getDate();if(!i)i=r;var h;var p;if(n){h=Date.UTC(u,s-1,r,0,0,0,0);p=Date.UTC(a,o-1,i,23,59,59,999)}else{h=Date(u,s-1,r,0,0,0,0).valueOf();p=Date(a,o-1,i,23,59,59,999).valueOf()}e=e.valueOf();return h<=e&&e<=p}function timeRange(){var e=0;var t=new Date;var n=new Date;var r=new Date;var i=arguments.length;var s=arguments[i-1];if(typeof s!='string')s=false;else{s=s.toUpperCase();if(s!='GMT')s=false;else{s=true;i--}}if(!i||i>6||i%2&&i!=1)return false;t.setMinutes(0);t.setSeconds(0);t.setMilliseconds(0);r.setMinutes(59);r.setSeconds(59);r.setMilliseconds(999);for(e=0;e<i/2;e++){var o=arguments[e];if(s){switch(e){case 0:t.setUTCHours(o);r.setUTCHours(o);break;case 1:t.setUTCMinutes(o);r.setUTCMinutes(o);break;case 2:t.setUTCSeconds(o);r.setUTCSeconds(o);break}}else{switch(e){case 0:t.setHours(o);r.setHours(o);break;case 1:t.setMinutes(o);r.setMinutes(o);break;case 2:t.setSeconds(o);r.setSeconds(o);break}}}if(i!=1){r.setMinutes(0);r.setSeconds(0);r.setMilliseconds(0);for(e=0;e<i/2;e++){var o=arguments[i/2+e];if(s){switch(e){case 0:r.setUTCHours(o);break;case 1:r.setUTCMinutes(o);break;case 2:r.setUTCSeconds(o);break}}else{switch(e){case 0:r.setHours(o);break;case 1:r.setMinutes(o);break;case 2:r.setSeconds(o);break}}}}n=n.valueOf();t=t.valueOf();r=r.valueOf();if(r<t)r+=864e5;return t<=n&&n<=r}function __FindProxyForURL(e,t){return FindProxyForURL(e,t)}" into tPacSupport

```
   set the cPACSupport of me to tPacSupport

   # initialise top level domains
   local tTopLevelDomains
   put
"biz,com,info,name,net,org,pro,aero,asia,cat,coop,edu,gov,int,jobs,mil,mobi,museum,tel,
travel,arpa,nato,example,invalid,localhost,test,bitnet,csnet,local,root,uucp,onion,exit,berl
in,lat,nyc,bzh,cym,gal,sco,geo,mail,kids,post,shop,web,xxx,ac,ad,ae,af,ag,ai,al,am,an,a
o,aq,ar,as,at,au,aq,ax,az,ac,ad,ae,af,ag,ai,al,am,an,ao,aq,ar,as,at,au,aw,ax,az,ba,bb,bd
,be,bf,bg,bh,bi,bj,bm,bn,bo,br,bs,bt,bw,by,bz,ca,cc,cd,cf,cg,ch,ci,ck,cl,cm,cn,co,cr,cu,cv,
cx,cy,cz,de,dj,dk,dm,do,dz,ec,ee,eg,er,es,et,eu,fi,fj,fk,fm,fo,fr,ga,gd,ge,gf,gg,gh,gi,gl,gm,
gn,gp,gq,gr,gs,gt,gu,gw,gy,hk,hm,hn,hr,ht,hu,id,ie,il,im,in,io,iq,ir,is,it,je,jm,jo,jp,ke,kg,kh,k
i,km,kn,kp,kr,kw,ky,kz,la,lb,lc,li,lk,lr,ls,lt,lu,lv,ly,ma,mc,md,me,mg,mh,mk,ml,mm,mn,mo,
mp,mq,mr,ms,mt,mu,mv,mw,mx,my,mz,na,nc,ne,nf,ng,ni,nl,no,np,nr,nu,nz,om,pa,pe,pf,p
g,ph,pk,pl,pn,pr,ps,pt,pw,py,qa,re,ro,rs,ru,rw,sa,sb,sc,sd,se,sg,sh,si,sk,sl,sm,sn,sr,st,su,
sv,sy,sz,tc,td,tf,tg,th,tj,tk,tl,tm,tn,to,tr,tt,tv,tw,tz,ua,ug,uk,us,uy,uz,va,vc,ve,vg,vi,vn,vu,wf,
ws,ye,za,zm,zw,um,bl,eh,mf,bv,gb,pm,sj,so,yt,tp,yu,cs,zr" into tTopLevelDomains
   replace comma with return in tTopLevelDomains
   set the cTopLevelDomains of me to tTopLevelDomains
end initialiseCustomProps

## for debugging ##
private function md5 pString
   local tHex
   get binaryDecode("h*", md5digest(pString), tHex)
   return tHex
end md5
##

#######Engine Calls#############################################
#######  load url  #############################################
-----------------------------------------------
on loadUrl x,y
   local newUrl
   put false into lvJumpOut
   put ulStripUrl(x) into newUrl
   if lvCount is empty then
      put "6923" into lvCount
   end if
   switch
      case newUrl is among the keys of laLoadingUrls
         ##don't allow loads if the same url is waiting to load
         return "error  URL is currently loading" ##with empty
         break
      case newUrl is not among the keys of laLoadedUrls OR laUrlLoadStatus[newUrl] is
not "cached"
```

```
        ##  put the long id of the target &","& item 2 of the params into
laMessg[newUrl]
        ##dc 081104
        put the long id of the target &","& item -1 of the params into laMessg[newUrl]
        put true into laLoadReq[newUrl]
        put 1 into laLoadingUrls[newUrl] #for tracking
        put "getData" into laAction[newUrl]
        put empty into laUrlErrorStatus[newUrl]
        put empty into laUrlLoadStatus[newUrl]
        put empty into laLoadedUrls[newUrl]

        ulGetFormat newUrl,lvCount

        -- SN 2014-02-21 Decode the URL read according to the encoding specified
in the head part
        ulDecodeData newUrl

        if laUrlLoadStatus[newUrl] is "error" and not laCancelled[newurl] then
          ulSendMessage newUrl ##send message now only if error occurred befoe
block point
          return "error"
          ----------------------
        else if laCancelled[newUrl] then
          ##user cancelled after starting but before blocking point
          delete local laLoadedUrls[newUrl]
          delete local laUrlLoadStatus[newUrl]
          delete local laUrlErrorStatus[newUrl]
          delete local laStatus[newUrl]
          delete local laCancelled[newUrl]
          delete local laCurrentHttpHeaders[newUrl]
        else
          return empty
        end if
        ----------------------
        break
      case newUrl is among the keys of laLoadedUrls and laUrlLoadStatus[newUrl] is
"cached" #url is in cache

        ## put the long id of the target &","& item 2 of the params into
laMessg[newUrl]
        ##dc 081104
        put the long id of the target &","& item -1 of the params into laMessg[newUrl]
        ulSendMessage newUrl ##send message
        return empty
    end switch
```

```
end loadUrl

############unload url############

on unloadUrl pUrl
  put ulStripUrl(pUrl) into pUrl
  ##need to check if it is loading or in loadQ
  if pUrl is among the keys of laLoadingUrls then
    delete local laLoadingUrls[pUrl]
    ulCancelRequest pUrl ##stop any current downloads
    delete local laData[pUrl] ##in case download hasn't started
    return empty
  else if pUrl is among the keys of laUrlLoadStatus then
    delete local laLoadedUrls[pUrl]
    delete local laUrlLoadStatus[pUrl]
    delete local laUrlErrorStatus[pUrl]
    delete local laStatus[pUrl]
    return empty
  else ##not loaded
    return "can't find url"
  end if
end unloadUrl

############ get url ############

on getUrl x
  local newUrl,tRetResult,tRetData
  put false into lvJumpOut
  put ulStripUrl(x) into newUrl
  if newUrl is among the keys of laLoadedUrls and laUrlLoadStatus[newUrl] is "cached"
then
    return empty  with urlResult laLoadedUrls[newUrl]
  end if
  if newUrl is among the keys of laLoadingUrls and (lvAuthBlockBypass is not true)
then
    return "error  URL is currently loading" with urlResult empty
  end if
  if lvBlockingUrl is empty or lvBlockBypass is true or lvAuthBlockBypass is true or
(lvExtDriver is not empty and ulExtIsBlocked() is false) then
    put newUrl into lvBlockingUrl
    if lvCount is empty then
      put "6923" into lvCount
    end if

    put empty into laUrlErrorStatus[newUrl]
    put "getData" into laAction[newUrl]
```

```
    put empty into laData[newUrl]
    ulGetFormat newUrl,lvCount # convert url to components

    -- SN 2014-02-21 Decode the URL read according to the encoding specified in
the head part
    ulDecodeData newUrl
    ##final clean up here

    if laAuthRecursCount[newUrl] < 1 then
      delete local laStatus[newUrl] ##
      delete local laAuthRecursCount[newUrl] ##

      put laUrlErrorStatus[newUrl] into tRetResult
      delete local  laUrlErrorStatus[newUrl]

      put laData[newUrl] into tRetData ##swap data before deleting laData
      delete local laData[newUrl]
      put empty into lvBlockingUrl ##clear
      return tRetResult with urlResult tRetData
    else
      return 1
    end if
  else ##blocked by previous request
    return "error Previous request not completed" with urlResult empty
  end if
end getUrl

############### post Url ###################

on postUrl y,x
  local newUrl,tRetResult,tRetData
  put textEncode(y, "native") into y
  put false into lvJumpOut
  put ulStripUrl(x) into newUrl
  if newUrl is among the lines of keys(laLoadingUrls) then
    return "error  URL is currently loading" with urlResult empty
  end if
  if lvBlockingUrl is empty  or lvBlockBypass or lvAuthBlockBypass is true or
(lvExtDriver is not empty and ulExtIsBlocked() is false) then
    put newUrl into lvBlockingUrl
    if lvCount is empty then
      put "6923" into lvCount
    end if
    put y into laPostData[newUrl]
    put length(y) into laPostLength[newUrl]
    put 0 into laPostBytes[newUrl]
```

```
      put "postData" into laAction[newUrl]
      put empty into laUrlErrorStatus[newUrl]
      put empty into laData[newUrl]
      ulGetFormat newUrl,lvCount # convert url to components

      ---------------------------------
      ##final clean up here
      if laAuthRecursCount[newUrl] < 1 then
        delete local laStatus[newUrl]
        delete local laAuthRecursCount[newUrl]

        put laUrlErrorStatus[newUrl] into tRetResult
        delete local  laUrlErrorStatus[newUrl]

        put laData[newUrl] into tRetData
        delete local laData[newUrl]
        put empty into lvBlockingUrl ##clear
        return tRetResult with urlResult tRetData
      else
        return 1
      end if
      ---------------------------------
    else ##blocked by previous request
      put "error Previous request not completed" into tRetResult
      return tRetResult with urlResult empty
    end if
end postUrl

################ put x into url ################

on putUrl y,x
    local newUrl,tRetResult,tRetData
    put textEncode(y, "native") into y
    put false into lvJumpOut
    put ulStripUrl(x) into newUrl
    if newUrl is among the lines of keys(laLoadingUrls) then
      return "error  URL is currently loading" with urlResult empty
    end if
    if lvBlockingUrl is empty  or lvBlockBypass or lvAuthBlockBypass is true or
(lvExtDriver is not empty and ulExtIsBlocked() is false) then
      put newUrl into lvBlockingUrl
      if lvCount is empty then
        put "6923" into lvCount
      end if
      put y into laPostData[newUrl]
      put length(y) into laPostLength[newUrl]
```

```
      put 0 into laPostBytes[newUrl]

      put "putData" into laAction[newUrl]
      put empty into laUrlErrorStatus[newUrl]
      put empty into laData[newUrl]
      ulGetFormat newUrl,lvCount # convert url to components
      ---------------------------------
      ##final clean up here
      if laAuthRecursCount[newUrl] < 1 then
        delete local laStatus[newUrl]
        delete local laAuthRecursCount[newUrl]

        put laUrlErrorStatus[newUrl] into tRetResult
        delete local  laUrlErrorStatus[newUrl]

        put laData[newUrl] into tRetData
        delete local laData[newUrl]
        put empty into lvBlockingUrl ##clear
        return tRetResult with urlResult tRetData
      else
        return 1
      end if
      ---------------------------------

    else ##blocked by previous request
      put "error Previous request not completed" into tRetResult
      return tRetResult with urlResult empty
    end if
end putUrl

###############delete url#########################

on deleteUrl x
    local newUrl,tRetResult,tRetData
    put false into lvJumpOut
    put ulStripUrl(x) into newUrl
    if newUrl is among the lines of keys(laLoadingUrls) then
      return "error  URL is currently loading" with urlResult empty
    end if
    if lvBlockingUrl is empty  or lvBlockBypass is true or (lvExtDriver is not empty and
ulExtIsBlocked() is false) then
      put newUrl into lvBlockingUrl
      if lvCount is empty then
        put "6923" into lvCount
      end if
```

```
      put "deleteData" into laAction[newUrl]
      put empty into laUrlErrorStatus[newUrl]
      ulGetFormat newUrl,lvCount # convert url to components

      ---------------------------------
      ##final clean up here
      delete local laStatus[newUrl]
      put laUrlErrorStatus[newUrl] into tRetResult
      delete local  laUrlErrorStatus[newUrl]

      put laData[newUrl] into tRetData
      delete local laData[newUrl]
      put empty into lvBlockingUrl ##clear
      return tRetResult with urlResult tRetData
      ---------------------------------

    else ##blocked by previous request
      put "error Previous request not completed" into tRetResult
      return tRetResult with urlResult empty
    end if
end deleteUrl

######## cachedUrls ################################################
on getCachedUrls
    local tKey,tLoadedKeys,tRes
    #ensure url has "cached" status
    #there may be urls with data but with "error" status
    #for example, with a 404 error, the "courtesy page" may appear in the data
    put keys(laUrlLoadStatus) into tLoadedKeys
    repeat for each line tKey in tLoadedKeys
      if laUrlLoadStatus[tKey] is "cached" then
        put tKey & cr after tRes
      end if
    end repeat
    if char -1 of tRes is cr then delete char -1 of tRes
    return tRes
end getCachedUrls

######## UrlStatus ##################################################
on getUrlStatus x #x is url
    put ulStripUrl(x) into x
    return laUrlLoadStatus[x]
end getUrlStatus

####################breaks down the url into
components####################
```

```
on ulGetFormat pUrl,pCount
  local tString,tOff,tURLHost,tUrlPort,tSavedDel
  local tConnectHost,tBadAddress
  local tPre,tUser,tPass,tHost,tPort,tFilename,tHostblock

  try
    ## url parsing method changed for 1.0.13
    ## we use offset and not just matchText to keep compatible with older engines
    ## that can't handle non-greedy regex

    put pUrl into tString
    put offset("://", tString) into tOff
    if tOff > 0 then
      put char 1 to tOff -1 of tString into tPre
      delete char 1 to tOff + 2 of tString
      put offset("/",tString) into tOff
      if tOff > 0 then
        put char 1 to tOff -1 of tString into tHostblock
        put char tOff  to -1 of tString into tFilename
      else if tOff = 0 and length(tString) > 0 then
        put tString into tHostblock
        put empty into tFilename
      else
        throw "error"
      end if

    else
      throw "error"
    end if

    if  matchText(tHostBlock, "(.+):(.+)@([^:]*)(.*)", tUser,tPass,tHost,tPort) then

    else if  matchText(tHostBlock, "([^:]*)(.*)", tHost,tPort) then

    else
      throw "error"
    end if

    if tPort is not empty and char 2 to -1 of tPort is not a integer then
      throw "error"
    end if


  catch pErr
    put "invalid URL: " & quote & pUrl & quote into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "error" into laUrlLoadStatus[pUrl]
```

```
    exit "ulGetFormat"
  end try

  if tPre is "https" and char 1 to 3 of the version < 2.6  then
    put "https protocol not supported in this version" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then
      put "error" into laUrlLoadStatus[pUrl]
      delete local laLoadingUrls[pUrl]
    end if
    exit "ulGetFormat"
  end if

  ##set connection IP
  switch tPre
    case "http"
    case "https"

      -- MM-2014-02-27: [[ PAC Support ]] Extract any proxy for the given URL.
      put _proxyForURL(pURL) into laUrlProxy[pURL]
      ulLogIt "Proxy for URL:" && laUrlProxy[pURL] & cr

      -- CW-2016-08-14: [[ External driver support ]] Bypass URL/Proxy
manipulation and async check for external drivers.
      if lvExtDriver is empty then
        -- CW-2016-08-14: If we are not using an external driver, asynchronous
HTTP uploads are not supported.
        if laAction[pUrl] is "putData" and laLoadReq[pUrl] is true then
          put "HTTP async uploads not supported in this version" into
laUrlErrorStatus[pUrl]
          put "error" into laUrlLoadStatus[pUrl]
          delete local laLoadingUrls[pUrl]
          exit "ulGetFormat"
        end if

        -- MM-2014-02-27: [[ HTTPS Proxy ]] We now support fetching https URLs
through a proxy.
        if laUrlProxy[pURL] is not empty then
          ## for now don't allow https connections through proxies
          ## will look at this again later
          put tHost into tURLHost
          put laUrlProxy[pURL] into tHost
          replace "http://" with "" in tHost ##not sure if this is possible but just in
case
          if tPort <> empty then
            put tPort into tUrlPort ##save this for setting laFilename below
          else if tPre is "https" then
```

```
            put ":443" into tUrlPort
          else
            put ":80" into tUrlPort
          end if
          #get the proxy port
          put the itemdel into tSavedDel
          set the itemdel to ":"
          if the number of items of tHost > 1 and item -1 of tHost is a number then
            put ":" &  item -1 of tHost into tPort
            delete item -1 of tHost #remove port for now
          else
            put ":80" into tPort
          end if
          set the itemDel to tSavedDel
        else if tPort is empty then
          if tPre is "https" then
            put ":443" into tPort
          else
            put ":80" into tPort
          end if
        end if
      end if

      break
    case "sftp"
      -- CW-2016-06-11: [[ External driver support ]] SFTP is only supported by
external drivers.
      if lvExtDriver is empty then
        put "sftp protocol not supported in this version" into laUrlErrorStatus[pUrl]
        if laLoadReq[pUrl] then put "error" into laUrlLoadStatus[pUrl]
        exit "ulGetFormat"
      end if
    case "ftp"
      -- CW-2016-06-11: [[ External driver support ]] This processing is only
needed by the internal driver.
      if lvExtDriver is empty then
        if tPort is empty then put ":21" into tPort

        if tUser is empty then
          put "anonymous" into tUser
          put "guest" into tPass
        end if
      end if

      if tFilename is not empty then
        break
```

```
      end if
    default
      put "invalid URL: " & quote & pUrl & quote into laUrlErrorStatus[pUrl]
      if laLoadReq[pUrl] then put "error" into laUrlLoadStatus[pUrl]
      exit "ulGetFormat"
      break
  end switch

  ##store current state of httpHeaders ad use when request is actually processed
  put the httpHeaders into laCurrentHttpHeaders[pURL]

  if laUrlFormat[pUrl]["references"] > 0 then
    add 1 to laUrlFormat[pUrl]["references"]
  else
    put 1 into laUrlFormat[pUrl]["references"]
    put tPre into laUrlFormat[pUrl]["protocol"]
    put tHost into laURLFormat[pURL]["host"]
    put tPort into laURLFormat[pURL]["port"]
    put tUser into laURLFormat[pURL]["user"]
    put tPass into laURLFormat[pURL]["pass"]
    put tFilename into laURLFormat[pURL]["filename"]
    # if using a proxy, these contain the host & port of the requested URL
    put tURLPort into laURLFormat[pURL]["urlPort"]
    put tURLHost into laURLFormat[pURL]["urlhost"]
  end if

  -- CW-2016-06-11: [[ External driver support ]] Call external library to handle
request rather than process directly.
  if lvExtDriver is not empty then
    ulExtHandleRequest pUrl, laAction[pUrl], laCurrentHttpHeaders[pUrl],
laPostData[pUrl], laUrlProxy[pUrl]
  else
    local tResolveHost
    put merge("[[tHost]]l[[pURL]]") into tResolveHost
    # clear status before resolving hostname
    put empty into laStatus[pUrl]

    get hostNameToAddress(tResolveHost, "ulHostnameToAddressCallback")
    ## I.M. 2010-03-11  Need to check result and handle immediate failure where
    ##  no callback message will be sent
    if the result is not empty then
      ulHostNameToAddressCallback tResolveHost, empty, the result
    end if

    if laLoadReq[pUrl] is empty or  laAuthRecursCount[pUrl] > 0 then ##dc 170205
      repeat while laStatus[pUrl] is empty
```

```
        if lvJumpOut then exit to top
        wait for messages
      end repeat
    end if
  end if
end ulGetFormat

##########################################################################
##########

on ulHostNameToAddressCallback pHostname, pIP, pOptionalError
  local tURL, tOffset
  local tPre, tHost, tPort, tUser, tPass, tFilename, tURLHost, tURLPort
  local tConnectHost, tBadAddress
  put offset("l", pHostname) into tOffset
  put char (tOffset + 1) to -1 of pHostname into tURL

  put laURLFormat[tURL]["protocol"] into tPre
  put laURLFormat[tURL]["host"] into tHost
  put laURLFormat[tURL]["port"] into tPort
  put laURLFormat[tURL]["user"] into tUser
  put laURLFormat[tURL]["pass"] into tPass
  put laURLFormat[tURL]["filename"] into tFilename
  put laURLFormat[tURL]["urlPort"] into tURLPort
  put laURLFormat[tURL]["urlHost"] into tURLHost

  if pIP is not empty then
    put line 1 of pIP & tPort into tConnectHost
    ##TEMP CHECK
    if tPre is "https" then
      put tHost & tPort into tConnectHost
    end if
    ##
  else
    put true into tBadAddress
  end if

  if tBadAddress then
    if pOptionalError is not empty then
      put pOptionalError into laUrlErrorStatus[tURL]
    else
      put "invalid host address" into laUrlErrorStatus[tURL]
    end if
    if laLoadReq[tURL] then
      put "error" into laUrlLoadStatus[tURL]
      delete local laLoadingUrls[tURL]
```

```
      delete local laAction[tURL]
      ulSendMessage tURL
    end if
    put false into laStatus[tUrl]
    ##need better clean up here
  else

    ##dc 080702
    ##need to keep separate reference for ftp IPs by user
    ## in case we need to connect to two accounts simultaneously
    ##so just keep user + host ref for all  urls
    ##so laConnectHost has format host:portluser
    put tConnectHost &  "|" & tUser into laConnectHost[tURL]

    if tUser is not empty then put urlDecode(tUser) into laUser[tURL]
    if tPass is not empty  then
      put urlDecode(tPass) into laPasswd[tURL]
      put "true" into laAuth[tURL]
    end if
    put tHost into laHost[tURL]
    if tFileName is empty then put "/" into tFileName
    -- MM-2014-02-27: [[ HTTPS Proxy ]] We now support fetching HTTPS URLS
through a proxy.
    if laUrlProxy[tURL] <> empty then
      ----changed for 1.0.10 Remove username:password from url when sending
request through a proxy
      if tPre is "http" then
        put tPre & "://" & tURLHost & tUrlPort & tFilename into laLongFileName[tURL]
##check later about proxies!!!
      else
        put tFileName into laLongFileName[tURL]
      end if
      put tURLHost into laHost[tURL]
    else
      put tFileName into laLongFileName[tURL]
    end if
    put "booked" into laUrlErrorStatus[tURL]
    if tPre = "http" or tPre = "https" then
      if tPre = "https" then
        put lvSSLVerification into laCurrentSSLVerify[tURL]
      end if
      ulHttpRequest tURL
    else
      ulFtpRequest tURL
    end if
  end if
```

```
   ## AL-2013-07-30: [[ Bug 10796 ]] HTTP "get URL" omits port number from
HOST header
   ## delete this array at the end so that 'get url' can use the port and protocol
data.
   subtract 1 from laURLFormat[tURL]["references"]
   if laURLFormat[tURL]["references"] is 0 then
      delete local laURLFormat[tURL]
   end if
end ulHostNameToAddressCallback

##################choose http method####################

on ulHttpRequest pUrl
   switch
      case laAction[pUrl] is "getData"
         if(laLoadReq[pUrl]) is true and laAuthRecursCount[pUrl] < 1 then
            ulHttpLoad pUrl
         else
            ulGetHttp pUrl
         end if
         break
      case laAction[pUrl] is "deleteData"
         ulDeleteHttp pUrl
         break
      case laAction[pUrl] is "putData"
         ulPutHttp pUrl
         break
      case laAction[pUrl] is "postData"
         ulPostHttp pUrl
   end switch
end ulHttpRequest
-------------------------------------------------
##set up queue for http load requests
on ulHttpLoad pUrl
   local tIP,tLoadingKeys,tHaveConnection

   put laConnectHost[pUrl] into tIP
   put keys(laLoadQ) into tLoadingKeys
   if tIP is among the lines of tLoadingKeys then
      put true into tHaveConnection
   else
      put false into tHaveConnection
   end if
   put pUrl & cr after  laLoadQ[tIP]
   put "queued" into laUrlLoadStatus[pUrl]
```

```
  if not tHaveConnection then
    ulNextHttpLoadRequest tIP
  end if
end ulHttpLoad
----------------------------------------------
##dispatch next load request
on ulNextHttpLoadRequest pIP
  local tUrl

  put line 1 of laLoadQ[pIP] into tUrl
  if tUrl <> empty then
    if tUrl = lvBlockingUrl then ##the same URL is being requested in a blocking call
      repeat until lvBlockingUrl <> tUrl
        if lvJumpOut then exit to top
        wait for messages
      end repeat
    end if
    ##in case url was "unloaded" during any wait, check that it's still in the queue
    if tUrl is among the lines of keys(laLoadingUrls) then
      delete line 1 of laLoadQ[pIP] ##added in 1.0.8r4

      ulGetHttp tUrl
    else
      ##modified dc 00202 Delete current rquest if not in laLoadingUrls
      ----------------------------------
      ## CLEAN UP POINT if user cancelled while in queue
      delete line 1 of laLoadQ[pIP] ##delete this item
      ulCleanUpHttpLocals tUrl
      delete local laLoadReq[tUrl] ##added dc 210702
      delete local laLoadedUrls[tUrl]##added dc 210702
      delete local laMessg[tUrl]##added dc 210702

      delete local laUrlErrorStatus[tUrl]
      delete local laUrlLoadStatus[tUrl]
      delete local laCancelled[tUrl]
      if the number of lines of laLoadQ[pIP] = 0 then
        delete local laLoadQ[pIP]
        delete local laConnectID[pIP]
      else
        ##use send to ensure this thread  finishes before next request
        send "ulNextHttpLoadRequest" && quote & pIP & quote to me in 1 milliseconds
      end if
      ------------------------------
    end if
  end if
```

```
end ulNextHttpLoadRequest

##############method GET#########################

on ulGetHttp pUrl
  local tSocket,tRequest

  try
    put empty into laStatus[pUrl] ##set wait flag here
    put "started" into laUrlErrorStatus[pUrl]
    put ulWhichSocket(pUrl) into tSocket

    put pUrl into laUrl[tSocket] #ref the url to the used socket##KEY REFERENCE
    if tSocket is not among the lines of the openSockets then
      ulStartTickle ##safeguard routine
      get ulOpenSocket(tSocket)
      if not it then throw it ##error opening socket
    end if

    put "contacted" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "contacted" into laUrlLoadStatus[pUrl]
    ulSendCallback pUrl,"contacted"  ##CALLBACK FEATURE

    ##ADD file opening HERE for libUrlDownloadToFile
    if laFile[pUrl] <> empty then
      open file laFile[pUrl] for binary write
      if the result is not empty then
        throw the result
      end if
    end if

    put ulBuildHttpRequest(pUrl) into tRequest
    ulLogit  tRequest & cr ##LOG
    write tRequest & crlf  to socket tSocket  with message "ulStartRead"
    ----------
    if the result is not empty then
      throw the result  #early exit
    end if
    ----------
    ##blocking point "get url"
    #If we got here by "load url" then we don't block, otherwise we do

    if laLoadReq[pUrl] is empty or  laAuthRecursCount[pUrl] > 0 then ##dc 170205
      repeat while laStatus[pUrl] is empty
        if lvJumpOut then exit to top
        wait for messages
```

```
      end repeat

    end if
  catch pErr ##clean up point
    ulHttpEarlyCleanUp tSocket,pUrl,pErr
    exit ulGetHttp
  end try
end ulGetHttp
---------------------------------
on ulHttpEarlyCleanUp x, pUrl, pErr
  local tLoadReq,tConnectHost

  put "error" && pErr into laUrlErrorStatus[pUrl]
  if laLoadReq[pUrl] then put "error" into laUrlLoadStatus[pUrl]
  put false into laStatus[pUrl] ##to unblock waits
  put laLoadReq[pUrl] into tLoadReq ##holder
  put laConnectHost[pUrl] into tConnectHost
  ulCleanUpHttp x

  # OK-2009-03-25 : Bug 7837 - Error code 624 means user interrupt. In this case
we unblock and exit to top,
  # to allow users to abort loops containing locking url accesses.
  if item 1 of pErr = 624 then
    put empty into lvBlockingUrl
    ulExitToTop
  end if

  if tLoadReq then
    ulSendMessage pUrl ##added 091002
    ##use send to ensure this thread finishes before next request starts
    send "ulNextHttpLoadRequest" && quote & tConnectHost & quote to me in 1
milliseconds
  end if
end ulHttpEarlyCleanUp

private command ulExitToTop
  exit to top
end ulExitToTop
-----------------------------------------------------------
on ulStartRead x,y
  local tUrlHolder

  if laUrl[x] <> empty then ##carry on
    ---------------------------------
    put "requested" into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put "requested" into laUrlLoadStatus[laUrl[x]]
```

```
ulSendCallback laUrl[x],"requested"  ##CALLBACK FEATURE
if laLoadReq[laUrl[x]] then
  put empty into laLoadedUrls[laUrl[x]]
else
  put empty into  laData[laUrl[x]]
end if
put empty into laTmpData[laUrl[x]]
put empty into laTemp[laUrl[x]]

read from socket x  with message "ulReadmore"
--------------------------------
if the result <> empty then
  put "error" && the result into laUrlErrorStatus[laUrl[x]]
  if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
  put false into laStatus[laUrl[x]] ##to unblock wait
end if
--------------------------------
##The wait in the getHttp handler allows "load" requests to pass
## we wait here for ALL requests so we can clean up
---------------------------------------
put laUrl[x] into tUrlHolder
repeat while laStatus[tUrlHolder] is empty
  if lvJumpOut then exit to top
  wait for messages
end repeat
---------------------------------------
## CLEAN UP POINT
####auth handling here

##  if not laLoadReq[laUrl[x]] then ##removed to allow load with authcallback
ulHandleAuth x
##    end if

if laStatus[laUrl[x]] is false and x is among the lines of the opensockets then
  close socket x ##if user unloaded
else if laConn[laUrl[x]] is "close" and x is among the lines of the opensockets then
  close socket x ##
end if
if word 1 of laUrlErrorStatus[laUrl[x]] is "error" then
  ulSendCallback laUrl[x], "error"  ##send CALLBACK here if error
end if
if laAuthRecursCount[laUrl[x]] < 1 then
  ulHttpLateCleanUp x
  if lvNewSocketForAuth > 0 then
    ulRemoveAuthSocketRefs tUrlHolder
```

```
      end if
    end if

  end if

end ulStartRead
-------------------------------------
on ulRemoveAuthSocketRefs pUrl
  ##removes any other socket references to laUrl
  ##that may have been used in authorisation recursion
  ## when new sockest were used
  put empty into lvNewSocketForAuth
  local tKeys
  put keys(laUrl) into tKeys
  repeat for each line tSocket in tKeys
    if laUrl[tSocket] = pUrl then

      delete local laUrl[tSocket]
    end if
  end repeat
end ulRemoveAuthSocketRefs
-------------------------------------------------------------
on ulHandleAuth x
  local tSkip,tOff,tAuth,i,tLine,tCloseSocket,tRes, tNewSocketForAuth

  if laStatusCode[laUrl[x]] is 407 then ##proxy authentication
    put 0 into tSkip
    repeat
      put lineOffset("Proxy-Authenticate:",laRhHeader[laUrl[x]],tSkip) into tOff
      if tOff = 0 then exit repeat
      put line (tOff + tSkip) of laRhHeader[laUrl[x]] into tAuth[word 2 of line (tOff + tSkip)
of laRhHeader[laUrl[x]]]
      add tOff to tSkip
    end repeat

    repeat with i = the number of lines of lvAuthCallbacks down to 1
      put line i of lvAuthCallbacks into tLine
      if item 1 of tLine is among the lines of keys(tAuth) then
        put laConn[laUrl[x]] is "close" into tCloseSocket ##

        add 1 to laAuthRecursCount[laUrl[x]]
        if tCloseSocket then
          put true into laSocketClosedByScript[x] ##for OS X
          close socket x ##
          add 1 to lvNewSocketForAuth
        end if
```

```
            ulSendAuthMessage item 1 of tLine, laUrl[x], tAuth[item 1 of tLine]
            put the result into tRes
            subtract 1 from laAuthRecursCount[laUrl[x]]

            exit repeat
          end if
        end repeat
      else if laStatusCode[laUrl[x]] is 401 then ###www-authentication
        put 0 into tSkip
        repeat
          put lineOffset("WWW-Authenticate:",laRhHeader[laUrl[x]],tSkip) into tOff
          if tOff = 0 then exit repeat
          put line (tOff + tSkip) of laRhHeader[laUrl[x]] into tAuth[word 2 of line (tOff + tSkip)
of laRhHeader[laUrl[x]]]
          add tOff to tSkip
        end repeat

        repeat with i = the number of lines of lvAuthCallbacks down to 1
          put line i of lvAuthCallbacks into tLine
          if item 1 of tLine is among the lines of keys(tAuth) then
            put laConn[laUrl[x]] is "close" into tCloseSocket ##
            add 1 to laAuthRecursCount[laUrl[x]]
            if tCloseSocket then
              put true into laSocketClosedByScript[x] ##for OS X
              close socket x ##
              add 1 to lvNewSocketForAuth

            end if

            ulSendAuthMessage item 1 of tLine, laUrl[x], tAuth[item 1 of tLine]
            put the result into tRes

            subtract 1 from laAuthRecursCount[laUrl[x]]

            exit repeat

          end if
        end repeat


      end if

    end ulHandleAuth
    ----------------------------------
    on ulParseHeaders x
```

```
   local tLocLine,tLenLine,tContentLine,tCodeLine,tConnectionLine,ptConnLine

   set the lastRhHeaders of me to laRhHeader[laUrl[x]] ##set property
   // SN-2015-06-04: [[ Bug 15456 ]] Apply fix proposed by Dave Cragg
   //  see http://runtime-revolution.278305.n4.nabble.com/OT-More-on-false-
timeouts-and-headers-td4692465.html
   filter lines of laRhHeader[laUrl[x]]  without  "*Content-Transfer-Encoding:*"

   put lineOffset("Location:",laRhHeader[laUrl[x]]) into tLocLine
   put lineOffset("Content-Length:",laRhHeader[laUrl[x]]) into tLenLine
   put lineOffset("Content-Type:",laRhHeader[laUrl[x]]) into tContentLine
   put lineOffset("Transfer-Encoding:",laRhHeader[laUrl[x]]) into tCodeLine
   put lineOffset("Connection:",laRhHeader[laUrl[x]]) into tConnectionLine
   put lineOffset("Proxy-Connection:",laRhHeader[laUrl[x]]) into ptConnLine

   #get status code
   put word 2 of line 1 of laRhHeader[laUrl[x]] into laStatusCode[laUrl[x]]
   #get status message for error results
   put word 2 to -1 of line 1 of laRhHeader[laUrl[x]] into laStatusMessage[laUrl[x]]

   if laStatusCode[laUrl[x]] is not a number then ##no point hanging around
      return "Unable to resolve server response."
      # put "error" && "Unable to resolve server response." into
laUrlErrorStatus[laUrl[x]]
      # if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
      # put false into laStatus[laUrl[x]] ##to unblock wait
      # exit "uIDoProcess"
   end if

   # MW-2011-04-23: [[ Bug 9520 ]] HTTP headers don't necessarily have a space
   #     after the ':'.
   set the itemDelimiter to ":"

   if tConnectionLine is not "0" then
      put the last word of item 2 to -1 of line tConnectionLine of laRhHeader[laUrl[x]] into
laConn[laUrl[x]]
   else
      put empty into laConn[laUrl[x]]
   end if
   if ptConnLine is not "0" then
      put the last word of item 2 to -1 of line ptConnLine of laRhHeader[laUrl[x]] into
laConn[laUrl[x]]
   end if
   if tLenLine is not "0" and the last word of item 2 to -1 of (line tLenLine of
laRhHeader[laUrl[x]]) is a number then
```

```
    put the last word of item 2 to -1 of (line tLenLine of laRhHeader[laUrl[x]]) into
laLength[laUrl[x]]
  else
    put empty into laLength[laUrl[x]]
    put empty into laHttpDataDone[laUrl[x]]
  end if
  if tCodeLine <> "0" then
    put the last word of item 2 to -1 of line tCodeLine of laRhHeader[laUrl[x]] into
laCode[laUrl[x]]
  end if
  if tLocLine <> "0" then
    put the last word of item 2 to -1 of line tLocLine of laRhHeader[laUrl[x]] into
laNewLoc[laUrl[x]]
  end if

  return empty
end ulParseHeaders
-------------------------------------
on ulHttpLateCleanUp x
  local tLoadReq,tUrlHolder,tConnectHost

  if laFile[laUrl[x]] <> empty then
    close file laFile[laUrl[x]] ##close here??
  end if

  put laLoadReq[laUrl[x]] into tLoadReq ##holder
  put laUrl[x] into tUrlHolder #so we can delete in cleanUp
  put laConnectHost[laUrl[x]] into tConnectHost #holder so we can delete in clean up
  ulCleanUpHttp x
  if tLoadReq and laCancelled[tUrlHolder] then
    delete local laLoadedUrls[tUrlHolder]
    delete local laUrlLoadStatus[tUrlHolder]
    delete local laUrlErrorStatus[tUrlHolder]
    delete local laStatus[tUrlHolder]
  end if
  if not laCancelled[tUrlHolder] then
    ulSendMessage tUrlHolder
  else
    delete local laMessg[tUrlHolder]
  end if
  delete local laFile[tUrlHolder]
  delete local laCancelled[tUrlHolder]
  if tLoadReq then
    if lvNewSocketForAuth > 0 then
      ##need todelete these when cleaning upafter recursive auth calls
      delete local laData[tUrlHolder]
```

```
      delete local laAuthRecursCount[tUrlHolder]
      put empty into lvBlockingUrl
    end if
    ##use send to ensure current request finishes completely
    send "ulNextHttpLoadRequest" && quote & tConnectHost & quote to me in 1
milliseconds
  end if

end ulHttpLateCleanUp
----------------------------------------------------------------
on ulReadmore x,y
  local tHeaderOffSet
  #separate the header from body

  put false into laHaveHeader[laUrl[x]]
  put y after laTmpData[laUrl[x]]
  put y after laTemp[laUrl[x]]

  put lineOffset(crlf & crlf, laTmpData[laUrl[x]]) into tHeaderOffSet ##proper header
structure

  ##added to catch irregularly formed headers 1.0.7b1
  if tHeaderOffset is 0 then ##for irregularly formed headers
    put lineOffset(cr & crlf, laTmpData[laUrl[x]]) into tHeaderOffSet
    if tHeaderOffset is 0 then
      put lineOffset(cr & cr, laTmpData[laUrl[x]]) into tHeaderOffSet
    end if
  end if

  if tHeaderOffSet is not 0 then#1
    put tHeaderOffSet into laLineNum[laUrl[x]]
    put line 1 to laLineNum[laUrl[x]] of laTmpData[laUrl[x]] into laRhHeader[laUrl[x]]
    repeat
      ##be sure we have a header
      if char 1 to 4 of laRhHeader[laUrl[x]] = "HTTP" then exit repeat
      delete line 1 of laRhHeader[laUrl[x]]
      if laRhHeader[laUrl[x]] is empty then ##we don't have a header
        put "error" && "No header received" into laUrlErrorStatus[laUrl[x]]
        if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock wait
        exit "ulReadmore"
      end if
    end repeat

    switch
      case word 2 of line 1 of laRhHeader[laUrl[x]] is  "100"
```

```
          ##Is this handled right??
          ulLogit line 1 to laLineNum[laUrl[x]]+1 of laTmpData[laUrl[x]] & cr ##LOG
          delete line 1 to laLineNum[laUrl[x]]+1 of laTmpData[laUrl[x]]
          delete line 1 to laLineNum[laUrl[x]]+1 of laTemp[laUrl[x]]

          get lineOffset(crlf & crlf, laTmpData[laUrl[x]])
          if it is not "0" then
            put it into laLineNum[laUrl[x]]
            put line 1 to laLineNum[laUrl[x]] of laTmpData[laUrl[x]] into
laRhHeader[laUrl[x]]
            ulDoProcess x,y
          else

            read from socket x  with message "ulReadmore"
          end if
          break
        case word 2 of line 1 of laRhHeader[laUrl[x]] is  not "100"
          ulDoProcess x,y
      end switch

    else#1
      if laStatus[laUrl[x]] is empty then
        read from socket x  with message "ulReadmore"
        ##how often should we do this reading for a header??
        ----------------------------------
        if the result <> empty then
          put "error" && the result into laUrlErrorStatus[laUrl[x]]
          if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
          put false into laStatus[laUrl[x]] ##to unblock wait below
        end if
        ----------------------------------
      end if
    end if#1
end ulReadmore
----------------------------------------
on ulDoProcess x,y

    local tCloseSocket,tNewLoc,tRedData,tUrlHolder,tStatus
    local tData

    #handles reading data depending on whether transfer method is streamed,
chunked or "until socket closes"

    if not laHaveHeader[laUrl[x]] then ##pick up header first time only
      ulLogIt laRhHeader[laUrl[x]] & cr & cr --LOG
```

```
    ulParseHeaders x
    if the result <> empty then
      put "error" && the result into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
      put false into laStatus[laUrl[x]] ##to unblock wait
      exit "ulDoProcess"

    end if
    delete line 1 to laLineNum[laUrl[x]]+1 in laTmpData[laUrl[x]]
    put true into laHaveHeader[laUrl[x]]

  else
    put y after laTmpData[laUrl[x]]
  end if
  ------------------------------------

  ------------------------------------
  switch
    case laStatusCode[laUrl[x]] is among the items of "301,302,307" and
laAction[laUrl[x]] is "getdata" and lvFollowHttpRedirects is not false

      # OK-2008-09-19 : fixed unquoted literal
      --case aStatusCode[laUrl[x]] is "303" and lvFollowHttpRedirects is not false
    case laStatusCode[laUrl[x]] is "303" and lvFollowHttpRedirects is not false
      #we are redirected to a different url
      if laNewLoc[laUrl[x]] is not empty then
        put laNewLoc[laUrl[x]] into tNewLoc
        ------------------------------
        put laConn[laUrl[x]] is "close" into tCloseSocket ##
        if tCloseSocket then close socket x
        put true into lvBlockBypass ##to allow another blocking call
        add 1 to laAuthRecursCount[laUrl[x]]

        put laUrl[x] into tUrlHolder

        local tResult
        put url tNewLoc into tRedData
        put the result into tResult

        put tUrlHolder into laUrl[x]

        if tResult is empty then
          ulStoreData laUrl[x],tRedData

          put empty into laUrlErrorStatus[laUrl[x]]
          if laFile[laUrl[x]] is empty then
```

```
            put "cached" into tStatus
          else
            put "downloaded" into tStatus
          end if
          if laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
          if laAction[laUrl[x]] <> "putData" then
            ulSendCallback laUrl[x],"downloaded"  ##CALLBACK
          else
            ulSendCallback laUrl[x],"uploaded" ##CALLBACK
          end if
        else
          -- MM-2014-08-12: [[ Bug 12798 ]] Report the actual reason for the
redirect failing.
          put "error" && "Redirect failed" && tNewLoc && "-" && tResult into
laUrlErrorStatus[laUrl[x]]
          if laLoadReq[laUrl[x]] is "true" then
            put "error" into laUrlLoadStatus[laUrl[x]]
            put empty into laLoadedUrls[laUrl[x]]
          else
            put empty into laData[laUrl[x]]
          end if
        end if
        subtract 1 from laAuthRecursCount[laUrl[x]]
        put false into lvBlockBypass##to disallow another blocking call
        ----------------------
        put true into laStatus[laUrl[x]]
      end if
      break

      ## added in 1.0,15b9
    case laStatusCode[laUrl[x]] is among the items of "100,304"
      ## responses that return no message entity
      ## and may not close the connection
      ## so we need to exit here
      put true into laStatus[laUrl[x]]
      put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] is "true" then
        put "error" into laUrlLoadStatus[laUrl[x]]
      end if

      delete local laHaveHeader[laUrl[x]]

      break

    case laStatusCode[laUrl[x]] is "200" and laURLFormat[laUrl[x]]["protocol"] is "https"
and laUrlProxy[laUrl[x]] is not empty and laSocketSecured[x] is false
```

```
        -- MM-2014-02-28: [[ HTTPS Proxy ]] A status code of 200 recieved when
fetching a HTTPS URL through a proxy
        --  indicates that the proxy has succesfully connected to the URL. We can
now continue communicating with the
        --  proxy as if talking to the URL directly. Secure the socket and send the
HTTP request as if the proxy was not there.
        local tURL
        put laURL[x] into tURL
        -- Deleting the URL logged against this socket means we will reuse this
(newly secured) socket when sending the request
        -- rather than createing a brand new one (see ulWhichSocket).
        delete variable laURL[x]
        put false into laHaveHeader[tURL]
        if laCurrentSSLVerify[tURL] is false then
          secure socket x without verification
        else
          secure socket x with verification for host _extractHost(tURL)
        end if
        put true into laSocketSecured[x]
        ulHttpRequest tURL
        break

        ##normal case
      case laLength[laUrl[x]] is not empty
      case laStatusCode[laUrl[x]] is "204"  -- When a 204 is returned no data is
returned, just the length of the data in the header.

        put  laTmpData[laUrl[x]] into tData
        ulStoreData laUrl[x],tData
        put length(tData) into laReadBytes[laUrl[x]]
        if laStatus[laUrl[x]]  is empty then ulDoProcessLength x

        break
        #### chunked ######################################
      case laCode[laUrl[x]] is "chunked"
        put true into laNeedChunk[laUrl[x]]
        put empty into laReadBytes[laUrl[x]]
        ulDoProcessChunked x
        break
        ##########No length header  ##typically from CGI request
        #Handle both cases together
      case laLength[laUrl[x]] is empty and laConn[laUrl[x]] is "close"
      case laLength[laUrl[x]] is empty and laConn[laUrl[x]] is empty
        put laTmpData[laUrl[x]] into tData
        ulStoreData laUrl[x],tData
        put length(tData) into laReadBytes[laUrl[x]]
```

```
      put empty into laTmpData[laUrl[x]] ##clear buffer
      ulDoProcessNoLength x
      break
  end switch
end ulDoProcess
-----------------------------------------------------------------
on ulDoProcessLength x,y
  local tStatus
  ##normal http case
  if y <> empty then
    ulStoreData laUrl[x],y
    add length(y) to laReadBytes[laUrl[x]]
  end if

  if laReadBytes[laUrl[x]] >= laLength[laUrl[x]] then
    put "true" into laStatus[laUrl[x]]

    if char 1 of laStatusCode[laUrl[x]] = 2 then ##in 200 range--OK
      if laFile[laUrl[x]] is empty then
        put "cached" into tStatus
      else
        put "downloaded" into tStatus
      end if
      put empty into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
      if laAction[laUrl[x]] <> "putData" then
        ulSendCallback laUrl[x],"downloaded"  ##CALLBACK
      else
        ulSendCallback laUrl[x],"uploaded" ##CALLBACK
      end if

    else
      put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] is "true" then
        put "error" into laUrlLoadStatus[laUrl[x]]
      end if
    end if

    delete local laHaveHeader[laUrl[x]]
    ------------------------
  else ##need more data
    put "loading," & laReadBytes[laUrl[x]] & "," & laLength[laUrl[x]] into tStatus
    put tStatus  into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put tStatus  into laUrlLoadStatus[laUrl[x]]
    ulSendCallback laUrl[x],tStatus  ##CALLBACK FEATURE
    if laStatus[laUrl[x]] is empty then
```

```
      read from socket x  with message "ulDoProcessLength"
      ----------
      if the result <> empty then
        put "error" && the result into laUrlErrorStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock waits above
        if laLoadReq[laUrl[x]] then
          put "error" into laUrlLoadStatus[laUrl[x]]
          put empty into laLoadedUrls[laUrl[x]] ##empty any data here
        else
          put empty into laData[laUrl[x]] ##empty any data here
        end if
      end if
      exit "ulDoProcessLength"
      ----------
    end if
  end if

end ulDoProcessLength
----------
on ulDoProcessChunked x,y
  local wOffset,tRead,tStatus,tData

  if y <> empty then
    put y after laTmpData[laUrl[x]]
  end if

  repeat while  laStatus[laUrl[x]] is empty
    if laNeedChunk[laUrl[x]] then
      get the number of chars of line 1 of laTmpData[laUrl[x]]
      if not ((char it of laTmpData[laUrl[x]] is numtochar(13)) and\
      (char it+1 of laTmpData[laUrl[x]] is numtochar(10))) then
        read from socket x with message "ulDoProcessChunked"
        ----------
        if the result <> empty then
          put "error" && the result into laUrlErrorStatus[laUrl[x]]
          put false into laStatus[laUrl[x]] ##to unblock wait above
          if laLoadReq[laUrl[x]] then
            put "error" into laUrlLoadStatus[laUrl[x]]
            put empty into laLoadedUrls[laUrl[x]] ##empty any data here
          else
            put empty into laData[laUrl[x]] ##empty any data here
          end if
        end if
        exit "ulDoProcessChunked"
      end if
```

```
  put offset(numtochar(13), laTmpData[laUrl[x]]) into wOffset[laUrl[x]]
  ----get chunk size value (store in laChunk)
  put char 1 to wOffset[laUrl[x]]-1 of laTmpData[laUrl[x]] into tRead[laUrl[x]]
  set the itemDel to ";"
  put item 1 of tRead[laUrl[x]] into tRead[laUrl[x]] ##remove any chunk extension
  set the itemdel to comma
  replace space with empty in tRead[laUrl[x]]

  put baseConvert(tRead[laUrl[x]],16,10) into laChunk[laUrl[x]]
  ---------
  delete char 1 to wOffset[laUrl[x]]+1 of laTmpData[laUrl[x]]
  if tRead[laUrl[x]] is "0" then ##completed
    delete local laNeedChunk[laUrl[x]]
    delete local laHaveHeader[laUrl[x]]
    put "true" into laStatus[laUrl[x]] ##to break out of wait
    if char 1 of laStatusCode[laUrl[x]] = 2 then ##in 200 range --OK
      if laFile[laUrl[x]] is empty then
        put "cached" into tStatus
      else
        put "downloaded" into tStatus
      end if
      put empty into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
      if laAction[laUrl[x]] <> "putData" then
        ulSendCallback laUrl[x],"downloaded"  ##CALLBACK
      else
        ulSendCallback laUrl[x],"uploaded" ##CALLBACK
      end if

    else
      put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] is "true" then
        put "error" into laUrlLoadStatus[laUrl[x]]
      end if
    end if
    exit "ulDoProcessChunked"
  end if
end if

switch
  case length(laTmpData[laUrl[x]])=laChunk[laUrl[x]] + 2
    put char 1 to laChunk[laUrl[x]] of laTmpData[laUrl[x]] into tData
    ulStoreData laUrl[x],tData
    add length(tData) to laReadBytes[laUrl[x]]
    put "loading," & laReadBytes[laUrl[x]] & "," into tStatus
    put tStatus  into laUrlErrorStatus[laUrl[x]]
```

```
    if laLoadReq[laUrl[x]] then put tStatus  into laUrlLoadStatus[laUrl[x]]
    ulSendCallback laUrl[x],tStatus  ##CALLBACK FEATURE
    delete char 1 to laChunk[laUrl[x]] + 2 of laTmpData[laUrl[x]]
    put true into laNeedChunk[laUrl[x]]
    if laStatus[laUrl[x]] is empty then
      read from socket x with message "ulDoProcessChunked"
      --------------------
      if the result <> empty then
        put "error" && the result into laUrlErrorStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock wait above
        if laLoadReq[laUrl[x]] then
          put "error" into laUrlLoadStatus[laUrl[x]]
          put empty into laLoadedUrls[laUrl[x]] ##empty any data here
        else
          put empty into laData[laUrl[x]] ##empty any data here
        end if
      end if
      exit "ulDoProcessChunked"
      ----------------------
    end if
    break
  case length(laTmpData[laUrl[x]])>laChunk[laUrl[x]] + 2
    put char 1 to laChunk[laUrl[x]] of laTmpData[laUrl[x]] into tData
    ulStoreData laUrl[x],tData
    add length(tData) to laReadBytes[laUrl[x]]
    put "loading," & laReadBytes[laUrl[x]] & "," into tStatus
    put tStatus  into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put tStatus  into laUrlLoadStatus[laUrl[x]]
    ulSendCallback laUrl[x],tStatus  ##CALLBACK FEATURE
    delete char 1 to laChunk[laUrl[x]] + 2 of laTmpData[laUrl[x]]
    put true into laNeedChunk[laUrl[x]]
    next repeat
    break
  case length(laTmpData[laUrl[x]]) < laChunk[laUrl[x]] + 2
    put false into laNeedChunk[laUrl[x]]
    if laStatus[laUrl[x]] is empty then
      read from socket x  with message "ulDoProcessChunked"
      -------------------------
      if the result <> empty then
        put "error" && the result into laUrlErrorStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock wait above
        if laLoadReq[laUrl[x]] then
          put "error" into laUrlLoadStatus[laUrl[x]]
          put empty into laLoadedUrls[laUrl[x]] ##empty any data here
        else
          put empty into laData[laUrl[x]] ##empty any data here
```

```
            end if
          end if
          ---------------------------
        end if
        exit "ulDoProcessChunked"
    end switch
  end repeat
end ulDoProcessChunked
---------------------------------------
on ulDoProcessNoLength x,y
  local tResult,tStatus

  if y <> empty then
    ulStoreData laUrl[x],y
    add length(y) to laReadBytes[laUrl[x]]
  end if

  if x is among the lines of the openSockets then ##test for closure here
    if laStatus[laUrl[x]] is empty and laHttpDataDone[laUrl[x]] is empty then
      read from socket x  with message "ulDoProcessNoLength"
      ---------------------------------------
      if the result <> empty then

        put the result into tResult
        if tResult is not "socket is not open" then

          put "error" && the result into laUrlErrorStatus[laUrl[x]]
          put false into laStatus[laUrl[x]] ##to unblock wait
          if laLoadReq[laUrl[x]] then
            put "error" into laUrlLoadStatus[laUrl[x]]##not likely
            put empty into laLoadedUrls[laUrl[x]] ##empty any data here
          else
            put empty into laData[laUrl[x]] ##empty any data here
          end if
        else ##assume that we've got all the data ##treat as "completed" as below

          put "true" into laStatus[laUrl[x]]
          if char 1 of laStatusCode[laUrl[x]] = 2 then
            if laFile[laUrl[x]] is empty then
              put "cached" into tStatus
            else
              put "downloaded" into tStatus
            end if
            put empty into laUrlErrorStatus[laUrl[x]]
            if  laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
            if laAction[laUrl[x]] <> "putData" then
```

```
            ulSendCallback laUrl[x],"downloaded"  ##CALLBACK
          else
            ulSendCallback laUrl[x],"uploaded" ##CALLBACK
          end if

        else
          put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
          if  laLoadReq[laUrl[x]] is "true" then put "error" into laUrlLoadStatus[laUrl[x]]
        end if
        delete local laHaveHeader[laUrl[x]]
      end if
    else

    end if
    ----------------------------------------
  end if
else ##completed

  put "true" into laStatus[laUrl[x]]
  if char 1 of laStatusCode[laUrl[x]] = 2 then
    if laFile[laUrl[x]] is empty then
      put "cached" into tStatus
    else
      put "downloaded" into tStatus
    end if
    put empty into laUrlErrorStatus[laUrl[x]]
    if  laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
    if laAction[laUrl[x]] <> "putData" then
      ulSendCallback laUrl[x],"downloaded"  ##CALLBACK
    else
      ulSendCallback laUrl[x],"uploaded" ##CALLBACK
    end if

  else
    put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
    if  laLoadReq[laUrl[x]] is "true" then put "error" into laUrlLoadStatus[laUrl[x]]
  end if
  delete local laHaveHeader[laUrl[x]]
end if
end ulDoProcessNoLength
##################HTTP DELETE################################

on ulDeleteHttp pUrl
  local tSocket,tRequest

  try
```

```
    put empty into laStatus[pUrl]
    put ulWhichSocket(pUrl) into tSocket
    put pUrl into laUrl[tSocket] #ref the url to the used socket
    if tSocket is not among the lines of the openSockets then
      ulStartTickle ##safeguard routine
      get ulOpenSocket(tSocket)
      if not it then throw it ##error opening socket
    end if
    put "contacted" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "contacted" into laUrlLoadStatus[pUrl]
    ulSendCallback pUrl,"contacted"  ##CALLBACK FEATURE
    put ulBuildHttpRequest(pUrl) into tRequest
    ulLogit  tRequest & cr ##LOG
    write tRequest & crlf  to socket tSocket
    if  the result is not empty then
      throw the result
    end if
    put "requested" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "requested" into laUrlLoadStatus[pUrl]
    ulSendCallback pUrl,"requested"  ##CALLBACK FEATURE
    read from socket tSocket   until crlf & crlf  with message "ulReadMore"
    if  the result is not empty then throw the result

    repeat while laStatus[pUrl] is empty
      if lvJumpOut then exit to top
      wait for messages
    end repeat
    ----------------------------------
    ulHttpLateCleanUp tSocket
    -------------------------
  catch pErr
    ulHttpEarlyCleanUp tSocket,pUrl,pErr
    exit ulDeleteHttp
  end try
end ulDeleteHttp

########################HTTP POST#########################

on ulPostHttp pUrl
  local tSocket,tRequest
  try
    put empty into laStatus[pUrl] ##set wait flag here
    put ulWhichSocket(pUrl) into tSocket
    put pUrl into laUrl[tSocket] #ref the url to the used socket

    if tSocket is not among the lines of the openSockets then
```

```
    ulStartTickle ##safeguard routine
    get ulOpenSocket(tSocket)
    if not it then throw it ##error opening socket
  end if
  put "contacted" into laUrlErrorStatus[pUrl]
  if laLoadReq[pUrl] then put "contacted" into laUrlLoadStatus[pUrl]
  ulSendCallback pUrl,"contacted"  ##CALLBACK FEATURE
  put ulBuildHttpRequest(pUrl) into tRequest
  put empty into  laData[pUrl]
  put empty into laTmpData[pUrl]
  ulLogit tRequest  & cr #LOG
  if "Expect: 100-continue" is in tRequest then
    write tRequest  & crlf   to socket tSocket with message "ulPostResponse"
  else
    write tRequest  & crlf   to socket tSocket with message "ulWriteSome"
  end if
  -------------------------
  if the result is not empty then
    throw the result
  end if
  -------------------------

  ## blocking point
  repeat while laStatus[pUrl] is empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat

  ulHandleAuth tSocket

  ## added for 1.0.11
  if laStatus[pUrl] is false and tSocket is among the lines of the opensockets then
    close socket tSocket
  else if laConn[pUrl] is "close" and tSocket is among the lines of the opensockets
then
    ##must close socket if remote host headers have "close" header
    close socket tSocket
  end if
  if word 1 of laUrlErrorStatus[pUrl] is "error" then
    ulSendCallback pUrl, "error"  ##send CALLBACK here if error
  end if

  if laAuthRecursCount[pUrl] < 1 then
    ulHttpLateCleanUp tSocket
    if lvNewSocketForAuth > 0 then
      ulRemoveAuthSocketRefs pUrl
```

```
      end if
    end if


  catch pErr
    ulHttpEarlyCleanUp tSocket,pUrl,pErr
    exit ulPostHttp
  end try
end ulPostHttp
-----------------------------------------------
on ulPostResponse x
  read from socket x with message "ulReadPostResponse"
end ulPostResponse
-----------------------------------------------
on ulReadPostResponse  x,y
  local tHeader,tResponseCode,tConn,tSkip,tOff,tAuth,tLine,tRes

  put empty into tHeader
  ulGetHttpHeader y, tHeader
  if the result is empty then
    put word 2 of line 1 of tHeader into tResponseCode
    put ulConnection(tHeader) into tConn
    switch tResponseCode
      case 100
        ulWriteSome x
        break

      default
        put tHeader into laRhHeader[laUrl[x]]
        ulParseHeaders x
        put "error" && word 2 to -1 of line 1 of tHeader into laUrlErrorStatus[laUrl[x]]

        if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock wait
        exit "ulReadPostResponse"
        break
    end switch
  else
    put "error" && the result into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
    put false into laStatus[laUrl[x]] ##to unblock wait
    exit "ulReadPostResponse"

  end if

end ulReadPostResponse
```

```
----------------------
function ulConnection pHeaders
  local tConnectionLine,ptConnLine,tConn

  put lineOffset("Connection:",pHeaders) into tConnectionLine
  put lineOffset("Proxy-Connection:",pHeaders) into ptConnLine
  if tConnectionLine is not "0" then
    put last word of line tConnectionLine of pHeaders into tConn
  end if
  if ptConnLine is not "0" then
    put last word of line ptConnLine of pHeaders into tConn
  end if
  return tConn

end ulConnection
-----------------------------------------------
-- Ensure the encoding has no space or hyphen included
function ulNormaliseEncoding pEncoding
  replace "-" with empty in pEncoding
  replace " " with empty in pEncoding
  return toLower(pEncoding)
end ulNormaliseEncoding

-- Convert the data according to the encoding specified
command ulDecodeData pUrl

  -- MM-2014-04-08: [[ Bug 12006 ]] Ignore the content type encoding for pre 7.0
versions of LiveCode.
  if the buildNumber < 10000 then
    return empty
  end if

  local tEncoding
  put empty into tEncoding

  local tHeadSize
  --   Only look for the encoding in the head part
  put offset("</head>", laData[pUrl]) into tHeadSize

  if tHeadSize is 0 then exit "ulDecodeData"

  local tHead
  put char 1 to tHeadSize of laData[pUrl] into tHead

  local tCharsetRegex
```

```
   put "<meta .*charset=" & quote & "?([a-zA-Z0-9 -]+)" & quote & ".*>" into
tCharsetRegex

   local tCharset
   if matchtext(tHead, tCharsetRegex, tCharset) then
      switch ulNormaliseEncoding(tCharset)
         case "utf8"
            put "UTF8" into tEncoding
            break
         case "utf16"
            put "UTF16" into tEncoding
            break
         case "iso88591"
            put "ANSI" into tEncoding
            break
      end switch

      if tEncoding is not empty then
         local tDecoded
         put uniEncode(laData[pUrl], tEncoding) into tDecoded
         put uniDecode(tDecoded) into laData[pUrl]
      end if
   end if
end ulDecodeData

-------------------------------------------------

on ulGetHttpHeader @pBuffer, @pHeader
   local tHeaderOffSet,tHeader
   ## added to allow 100 responses to post commands
   ## but must also accommodate 401,407 responses (authentication)
   put lineOffset(crlf & crlf, pBuffer) into tHeaderOffSet ##proper header structure

   ##added to catch irregularly formed headers 1.0.7b1
   if tHeaderOffset is 0 then ##for irregularly formed headers
      put lineOffset(cr & crlf, pBuffer) into tHeaderOffSet
      if tHeaderOffset is 0 then
         put lineOffset(cr & cr, pBuffer) into tHeaderOffSet
      end if
   end if

   if tHeaderOffSet is not 0 then
      put line 1 to tHeaderOffSet of pBuffer into tHeader
      repeat
         ##be sure we have a header
         if char 1 to 4 of tHeader = "HTTP" then exit repeat
```

```
      delete line 1 of tHeader
      if tHeader is empty then ##we don't have a header
        put empty into pHeader
        return "error bad header" ##BAD EXIT
      end if
    end repeat
    put tHeader into pHeader
    ulLogit pHeader & return ##LOG
    return empty ##GOOD EXIT
  else#1
    put empty into pHeader
    return "error no end of header" ##BAD EXIT
  end if
end  ulGetHttpHeader
----------------------------------------------------
on ulWriteSome x
  local tBlockSize,tOffset,tRemaining,tBytes,tChunk,tStatusString
  put 4096 into tBlockSize
  if laUrl[x] <> empty then ##in case an error was encountered when writing
    put laPostBytes[laUrl[x]] + 1 into tOffset
    put laPostLength[laUrl[x]] - laPostBytes[laUrl[x]] into tRemaining
    if tRemaining <= tBlockSize then ##don't forget to get tBytes
      get char tOffset to -1 of laPostData[laUrl[x]]
      # MW-2010-10-12: [[ Bug 8655 ]] Applying patch suggested in bug report.
      write it to socket x with message "ulWriteOver"
      ---------------------------------
      if the result <> empty then
        put "error" && the result into laUrlErrorStatus[laUrl[x]]
        if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
        put false into laStatus[laUrl[x]] ##to unblock wait above
      end if
      ---------------------------------
      repeat while laStatus[laUrl[x]] is empty
        if lvJumpOut then exit to top
        wait for messages
      end repeat
    else
      put char tOffset to (tOffset + tBlockSize -1) of laPostData[laUrl[x]] into tChunk
      ##delete char 1 to 4096 in laPostData[laUrl[x]]
      add tBlockSize to laPostBytes[laUrl[x]]
      put "uploading," & laPostBytes[laUrl[x]] & "," & laPostLength[laUrl[x]] into
tStatusString
      ulSendCallback laUrl[x],tStatusString  ##CALLBACK FEATURE
      write tChunk to socket x with message "ulWriteSome"

    end if
```

```
  end if
end ulWriteSome
----------------------------------
on ulWriteOver x
  put "requested" into laUrlErrorStatus[laUrl[x]]
  read from socket x  with message "ulReadmore"
end ulWriteOver
###################HTTP PUT############################

on ulPutHttp pUrl
  local tSocket,tRequest

  try
    put empty into laStatus[pUrl]##flag
    put ulWhichSocket(pUrl) into tSocket
    put pUrl into laUrl[tSocket] #ref the url to the used socket
    if tSocket is not among the lines of the openSockets then
      ulStartTickle ##safeguard routine
      get ulOpenSocket(tSocket)
      if not it then throw it ##error opening socket
    end if
    put "contacted" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "contacted" into laUrlLoadStatus[pUrl]
    ulSendCallback pUrl,"contacted"  ##CALLBACK FEATURE
    put ulBuildHttpRequest(pUrl) into tRequest
    put empty into  laData[pUrl]
    put empty into laTmpData[pUrl]
    ulLogit  tRequest & cr ##LOG

    ##just the same as ulPostHttp from this point
    if "Expect: 100-continue" is in tRequest then
      write tRequest  & crlf   to socket tSocket with message "ulPostResponse"
    else
      write tRequest  & crlf   to socket tSocket with message "ulWriteSome"
    end if
    -------------------------
    if the result is not empty then
      throw the result
    end if
    -------------------------

    ##blocking point
    repeat while laStatus[pUrl] is empty
      if lvJumpOut then exit to top
      wait for messages
```

```
      end repeat

      ulHandleAuth tSocket

      ## added for 1.0.11
      if laStatus[pUrl] is false and tSocket is among the lines of the opensockets then
        close socket tSocket
      else if laConn[pUrl] is "close" and tSocket is among the lines of the opensockets
then
        ##must close socket if remote host headers have "close" header
        close socket tSocket
      end if
      if word 1 of laUrlErrorStatus[pUrl] is "error" then
        ulSendCallback pUrl, "error"  ##send CALLBACK here if error
      end if

      if laAuthRecursCount[pUrl] < 1 then
        ulHttpLateCleanUp tSocket
        if lvNewSocketForAuth > 0 then
          ulRemoveAuthSocketRefs pUrl
        end if
      end if
    catch pErr
      ulHttpEarlyCleanUp tSocket,pUrl,pErr
      exit ulPutHttp
    end try
end ulPutHttp
----------------------------------------
function ulWhichSocket pUrl
    local tConnectHost,tConnID,tIsFtp,tSocket

    ##build socket ref including Connection ID number
    ## re-use sockets for blocking requests if open
    ##load request sequences will always use same socket
    ##new sequence gets new socket
    put laConnectHost[pUrl] into tConnectHost
    set the itemDel to "|"
    if laLoadReq[pUrl] then ##fixed typo here 1,1,6b1
      if laConnectID[tConnectHost] <> empty then
        put  laConnectID[tConnectHost] into tConnID
      else
        add 1 to lvCount
        put  lvCount into tConnID
        put lvCount into laConnectID[tConnectHost]
      end if
    else ##blocking request
```

```
      put empty into tConnID
      put char 1 to 4 of pUrl is "ftp:" into tIsFtp
      repeat for each line i in the openSockets
        if tIsFtp then
          ##makes sure we have the same username before re-using an  FTP socket
          if laSocketUser[i] <> laUser[pUrl] then
            next repeat
          end if
        end if
        if item 1 of tConnectHost is item 1 of i and (i is not among the lines of keys(laUrl)
or lvAuthBlockBypass is true or lvBlockBypass is true) then #OK to use
          if laConn[pUrl] <> "close" then
            put last item of i into tConnID

            exit repeat
          end if
        end if
      end repeat
      if tConnID is empty then ##need new socket ref
        add 1 to lvCount
        put lvCount into tConnID

      end if
    end if
    ##swap out user name for connection ID
    put item 1 of tConnectHost & "|" & tConnID into tSocket
    if laUser[pUrl] <> empty then
      ##for ftp sockets, we need to keep reference to user name
      put laUser[pUrl] into laSocketUser[tSocket] ## here or when connection is
made??
    end if
    set the itemDel to comma
    ulLogit "socket selected:" && tSocket & cr ##LOG
    delete local laSocketClosedByScript[tSocket]
    return tSocket
end ulWhichSocket

-----------------------------------------

private command _AddProxyAuthToRequest pUrl, @pRequest
    set the itemDel to "|"
    local tConnectHost
    put item 1 of laConnectHost[pUrl] into tConnectHost
    set the itemDel to comma

    local tFilter
```

```
   put tConnectHost & ",*" into tFilter

   local tAuthKeys
   put keys(laProxyAuthTokens) into tAuthKeys
   filter tAuthKeys with tFilter
   if tAuthKeys <> empty then
      local tKey
      put line 1 of tAuthKeys into tKey

      local tToken
      put item 2 of laProxyAuthTokens[tKey] into tToken

      local tUseAgain
      put item 1 of laProxyAuthTokens[tKey] into tUseAgain

      local tHeader
      put "Proxy-Authorization:" && tToken into tHeader

      put return & tHeader after pRequest

      if tUseAgain is not true then ##once only
         delete local laProxyAuthTokens[tKey]
      end if
   end if
end _AddProxyAuthToRequest

function ulBuildHttpRequest pUrl
   local tRequest,tAction,tMethod,tAgent,tDataSize,tLogin,tHaveServerAuth
   local tAuthKeys,tAuthKey,tFilter,tKey,tToken,tUseAgain,tHeader,tHeaderLine
   local tConnectHost,tRequest2

   ##build the httpRequest including
   ##request line and basic headers

   if the customHTTPHeaders of me <> empty then#a
      put the customHTTPHeaders of me into tRequest
      set the customHTTPHeaders of me to empty
   else if laUrlProxy[pUrl] is not empty and laUrlFormat[pUrl]["protocol"] is "https" then
      -- MM-2014-02-27: [[ HTTPS Proxy ]] We are fetching a HTTPS URL through a
proxy.
      --  If we have yet to secure the socket we are using for this URL, we must first
      --  tell the proxy we want to tunnel through (using the CONNECT command).
      --  If the socket is already secured, we can assume we are talking to the
desired URL
      --  directly (by tunneling through the proxy) and build the request as normal.
      local tSocket
```

```
      repeat for each key tSock in laUrl
        if laURL[tSock] is pURL then
          put tSock into tSocket
        end if
      end repeat
      if laSocketSecured[tSocket] is false then
        put "CONNECT" &&  laUrlFormat[pUrl]["urlHost"] & laUrlFormat[pUrl]["urlPort"] &&
"HTTP/1.1" & cr & "Host:" && laUrlFormat[pUrl]["urlHost"] & laUrlFormat[pUrl]["urlPort"]
into tRequest

        -- MM-2014-06-05: [[ Bug 12566 ]] Make sure we authenticate the proxy.
        if laUrlProxy[pUrl] <> empty then
          _AddProxyAuthToRequest pUrl, tRequest
        end if
      end if
    end if

    if tRequest is empty then
      ##get template
      put the cDefaultHeader of me into tRequest
      ##get method
      put laAction[pUrl] into tAction
      switch tAction
        case "getData"
          put "GET" into tMethod
          break
        case  "deleteData"
          put "DELETE" into tMethod
          break
        case "putData"
          put "PUT" into tMethod
          break
        case "postData"
          put "POST" into tMethod
          break
      end switch

      replace "METHOD" with tMethod in tRequest
      ----------------------------
      ##fill in url resource
      put  laLongFileName[pUrl] into word 2 of line 1 of tRequest
      ## fill in host
      put  laHost[pUrl]  after line 2 of tRequest
      ## AL-2013-07-30: [[ Bug 10796 ]] ¬†HTTP "get URL" omits port number from
HOST header
      ## add port if not default
```

```
    local tPort
    put laUrlFormat[pUrl]["port"] into tPort

    local tProtocol
    put laUrlFormat[pUrl]["protocol"] into tProtocol

    -- MM-2014-02-27: [[ HTTPS Proxy ]] We now support HTTPS URLs over a
proxy, so ammend port accordingly.
    if tProtocol is "https" and laUrlProxy[pUrl] is not empty and laUrlFormat[pUrl]
["urlPort"] is not ":443" then
      put laUrlFormat[pUrl]["urlPort"] after line 2 of tRequest
    else if (tProtocol is "https" and tPort is not ":443" and laUrlProxy[pUrl] is empty) or
(tProtocol is "http" and tPort is not ":80") then
      put tPort after line 2 of tRequest
    end if
    ## fill in User-Agent
    if "rev" is in the short name of me then
      put "LiveCode" into tAgent
    else
      put "Metacard" into tAgent
    end if
    put tAgent && "(" & the platform & ")" after line 3 of tRequest
    if tMethod is among the items of "PUT,POST" then
      put length(laPostdata[pUrl]) into tDataSize

      put cr & "Content-Length:" && tDataSize after tRequest
      put cr & "Content-Type:  application/x-www-form-urlencoded" after tRequest
      if tDataSize > laMaxPostWithoutExpect and laMaxPostWithoutExpect <> empty
then
        put cr & "Expect: 100-continue" after tRequest ##ADDED 1.0.15b3
      end if
    end if
    --    if laAuth[pUrl] is not empty then
    --      put base64Encode(laUser[pUrl] & ":" & laPasswd[pUrl]) into tLogin
    --      put cr & "Authorization: Basic" && tLogin after tRequest
    --      put true into tHaveServerAuth ##flag for whether to set authoriation
below or not
    --    else
    --      put false into tHaveServerAuth
    --    end if
    if laAuth[pUrl] is not empty then
      put base64Encode(laUser[pUrl] & ":" & laPasswd[pUrl]) into tLogin
      -- MM-2013-01-28: [[ Bug 10009 ]] Make sure auth string is all on a single line.
      replace return with empty in tLogin
      put cr & "Authorization: Basic" && tLogin after tRequest
```

```
      put true into tHaveServerAuth ##flag for whether to set authoriation below or
not
    else
      put false into tHaveServerAuth
    end if
    ----------------------------------
    #### new authorization headers 1.015b2
    ##proxy first
    -- MM-2014-02-27: [[ PAC Support ]] Updated to use new proxy for URL
variable rather than global property.
    -- MM-2014-06-05: [[ Bug 12566 ]] Centralised proxy authetication.
    if laUrlProxy[pUrl] <> empty then
      _AddProxyAuthToRequest pUrl, tRequest
    end if

    ##now server authorization
    if not tHaveServerAuth then
      put keys(laServerAuthTokens) into tAuthKeys
      repeat for each line tAuthKey in tAuthKeys
        if item 1 of tAuthKey is in pUrl then
          put item 2 of laServerAuthTokens[tAuthKey] into tToken
          put item 1 of laServerAuthTokens[tAuthKey] into tUseAgain
          put "Authorization:" && tToken into tHeader

          put return & tHeader after tRequest

          if tUseAgain is not true then ##once only
            delete local laServerAuthTokens[tAuthKey]
          end if
          exit repeat
        end if
      end repeat
    end if
    ----------------------------------

    ##customize according to httpHeaders
    if laCurrentHttpHeaders[pUrl] is not empty then

      repeat for each line tHeader in laCurrentHttpHeaders[pUrl]
        put lineOffset((word 1 of tHeader),tRequest) into tHeaderLine
        if tHeaderLine is not 0 and word 1 of tHeader = (word 1 of line tHeaderLine of
tRequest) then
          ##replace header value
          put tHeader into line tHeaderLine of tRequest
        else
          ##add new field
```

```
            put cr & tHeader after tRequest
          end if
        end repeat
      end if

    end if
    ##set the lastHTTPHeaders##
    set the lastHTTPHeaders of me to tRequest
    ##separate lines with crlf in header
    repeat for each line i in tRequest
      put  i & crlf after tRequest2
    end repeat
    return tRequest2

end ulBuildHttpRequest
##################################################
################ FTP LOGIN####################
########################################
on ulFtpRequest pUrl
  if laLoadReq[pUrl] then
    ulFtpLoad pUrl
  else if laAction[pUrl] = "postdata" then ##don't handle this
    put "error Post command not handled for FTP" into laUrlErrorStatus[pUrl]
    ulCleanUpFtpLocals pUrl

    exit "ulFtpRequest"
  else
    ulFtpSocket pUrl
  end if
end ulFtpRequest
----------------------------
##set up queue for http load requests
on ulFtpLoad pUrl
  local tIP,tLoadingKeys,tHaveConnection

  put laConnectHost[pUrl] into tIP
  put keys(laLoadQ) into tLoadingKeys
  if tIP is among the lines of tLoadingKeys then
    put true into tHaveConnection
  else
    put false into tHaveConnection
  end if
  put pUrl & cr after  laLoadQ[tIP]
  put "queued" into laUrlLoadStatus[pUrl]

  if not tHaveConnection then
```

```
    ulNextFtpLoadRequest tIP
  end if

end ulFtpLoad
-----------------------------------------------
##dispatch next request
on ulNextFtpLoadRequest pIP
  local tUrl
  put line 1 of laLoadQ[pIP] into tUrl
  if tUrl <> empty then
    if tUrl = lvBlockingUrl then ##the same URL is being requested in a blocking call
      repeat until lvBlockingUrl <> tUrl
        if lvJumpOut then exit to top
        wait for messages
      end repeat
    end if

    ##in case url was "unloaded" during any wait, check that it's still in the queue
    if tUrl is among the lines of keys(laLoadingUrls) then
      delete line 1 of laLoadQ[pIP] ##delete this item ##added for 1.0.8r4
      ulFtpSocket tUrl
    else
      ##Delete current rquest if not in laLoadingUrls
      ----------------------------------
      ## CLEAN UP POINT if user cancelled while in queue
      delete line 1 of laLoadQ[pIP] ##delete this item
      ulCleanUpFtpLocals tUrl
      delete local laLoadReq[tUrl] ##added dc 210702
      delete local laLoadedUrls[tUrl]##added dc 210702
      delete local laMessg[tUrl]##added dc 210702
      delete local laUrlErrorStatus[tUrl]
      delete local laUrlLoadStatus[tUrl]
      delete local laCancelled[tUrl]
      if the number of lines of laLoadQ[pIP] = 0 then
        delete local laLoadQ[pIP]
        delete local laConnectID[pIP]
      else
        ##use send .. in
        send "ulNextFtpLoadRequest" && quote & pIP & quote to me in 1 milliseconds
      end if
      ----------------------------------
    end if
  end if
end ulNextFtpLoadRequest

-------ulFtpSocket-----------------------------------------------------
```

```
-------Establishes connection for all ftp calls------------------
on ulFtpSocket pUrl
  local tSocket,tReply,tCmd,tErr,tUrlHolder

  put empty into laStatus[pUrl] ##set main wait flag here
  put ulWhichSocket(pUrl) into tSocket
  put pUrl into laUrl[tSocket] ##reference the url to the used socket
  put "0" into laStopUnit[tSocket]
  put "0" into laStopSec[tSocket]

  if tSocket is not among the lines of the openSockets then
    ulStartTickle ##safeguard routine
    get ulOpenSocket(tSocket)
    if not it then
      ulFtpEarlyExit tSocket,pUrl,it
      exit ulFtpSocket
    end if
    ----------------------get server response (220)
    put ulFtpWaitResponse(tSocket) into tReply
    if not ulFtpGoodReply(tReply, "connect") then
      ulFtpEarlyExit tSocket,pUrl,tReply
      exit "ulFtpSocket"
    end if
    ---------------
    put "connecting" into laUrlErrorStatus[pUrl]
    if laLoadReq[pUrl] then put "connecting" into laUrlLoadStatus[pUrl]
    ulSendCallback pUrl,"connecting"  ##CALLBACK FEATURE
    ---------------
    put "USER " & laUser[pUrl] into tCmd
    put ulFtpCommand(tCmd,tSocket) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      ulFtpEarlyExit tSocket,pUrl,tReply
      exit  "ulFtpSocket"
    end if
    ----------------------
    put  "PASS " & laPasswd[pUrl] into tCmd
    put ulFtpCommand(tCmd,tSocket) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      ulFtpEarlyExit tSocket,pUrl,tReply
      exit  "ulFtpSocket"
    end if

  end if
  ---------------------------
  write "PWD"  & crlf to socket tSocket with message "ulFtpStartPoint" ##BRANCH TO
ALLOW NON BLOCKING load
```

```
  if the result <> empty then
    put the result into tErr
    ulFtpEarlyExit tSocket,pUrl,tErr
    exit ulFtpSocket
  end if
  ----------------------------------------
  ##let non-blocking requests exit
  put pUrl into tUrlHolder ##so we can delete laUrl and  in ulFtpStartPoint on return
  if laLoadReq[tUrlHolder] is empty then
    repeat until laStatus[tUrlHolder] is not empty
      if lvJumpOut then exit to top
      wait for messages
    end repeat
  end if
end ulFtpSocket
----------------------------------------
on ulFtpEarlyExit pSocket,pUrl,pErr
  ####
  ##clean up when exiting before first blocking point
  ####
  local tConnectHost,tLoadReq

  put laConnectHost[pUrl] into tConnectHost
  replace "ftpErr," with empty in pErr
  put "error" &&  pErr into laUrlErrorStatus[pUrl]
  if laLoadReq[pUrl] then put "error" into laUrlLoadStatus[pUrl]
  put false into laStatus[pUrl]
  close socket pSocket
  delete local laSocketUser[pSocket]
  delete local laStopUnit[pSocket]
  delete local laStopSec[pSocket]

  put laLoadReq[pUrl] into tLoadReq ##holder
  ulCleanUpFtp pSocket
  if tLoadReq then
    ulSendMessage pUrl ##added 091002
    send "ulNextFtpLoadRequest" && quote & tConnectHost & quote to me in 1
milliseconds
  end if
end ulFtpEarlyExit
-------------------------
on ulFtpSetError x, pErr
  replace "ftpErr," with empty in pErr
  put "error " && pErr into laUrlErrorStatus[laUrl[x]]
  if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
  put false into laStatus[laUrl[x]] ##set flag to get past waits
```

```
end ulFtpSetError

-------ulFtpStartPoint----------------------------------------------------
-------Continues after load calls have passed on-------------------
on ulFtpStartPoint x
  local tNum,tReply,tLoadStatus,tLoadReq,tUrlHolder,tConnectHost

  set the itemdel to "|"
  put item -1 of x into tNum
  set the itemdel to comma
  put ulFtpWaitResponse(x) into tReply
  replace "ftpErr," with empty in tReply
  if not ulFtpGoodReply(tReply, "PWD") then ##command sent in ftpSocket
    ulFtpSetError x,tReply
    close socket x
    delete local laSocketUser[x]
  else
    ##051202 next 5 lines repair bug introduced in 1.0.8a1
    ## and ensure home directory is only set once per session
    ##otherwise CWD calls put us out of kilter
    if laHome[laUrl[x]] is empty then
      set the itemDel to quote
      put item 2 of tReply into laHome[laUrl[x]]
      set the itemDel to comma
    end if

    put "connected" into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put "connected" into laUrlLoadStatus[laUrl[x]]
    ulSendCallback laUrl[x],"connected"  ##CALLBACK FEATURE
    switch
    case laAction[laUrl[x]] is "getData"
      ##ADD open file for libUrlDownloadToFile
      if laFile[laUrl[x]] <> empty then
        open file laFile[laUrl[x]] for binary write
        if the result is not empty then
          ulFtpSetError x,the result
          close socket x
          delete local laSocketUser[x]
        end if
      end if

      ulFtpGet x,tNum
      break
    case laAction[laUrl[x]] is "putData"
      ##ADD open file for libUrlFtpUploadFile
      if laFile[laUrl[x]] <> empty then
```

```
      open file laFile[laUrl[x]] for binary read
      if the result is not empty then
        ulFtpSetError x,the result
        close socket x
        delete local laSocketUser[x]
      end if

    end if
    ulFtpSend x,tNum
    break
  case laAction[laUrl[x]] is "deleteData"
    ulFtpDelete x
    break
  default
    put false into laStatus[laUrl[x]]
    put "error Command not handled" into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
  end switch
end if
##block ALL requests here until everything finished
##load requests already got past in ulFtpSocket
repeat while laStatus[laUrl[x]] is empty
  if lvJumpOut then exit to top
  wait for messages
end repeat

## CLEAN UP POINT
if word 1 of laUrlErrorStatus[laUrl[x]] is "error" then
  ulSendCallback laUrl[x], "error"  ##send CALLBACK here if error
end if
-----------------------------------

put laUrlLoadStatus[laUrl[x]] into tLoadStatus
-----------------------------------

#start timer routine for closing ftp connection
if laStopUnit[x] = 0 then
  put "1" into laStopUnit[x]
  send "ulFtpStopWatch " & x to me in 50 milliseconds
end if
##do cleanup here
##first close file if necessary
if laFile[laUrl[x]] <> empty then
  if laUrlByFile[laFile[laUrl[x]]] = laUrl[x] or laAction[laUrl[x]] = "getData" then ##hasn't
been opened by new request
    close file laFile[laUrl[x]] ##close here??
```

```
    delete local laUrlByFile[laFile[laUrl[x]]]
   else
    seek to 0 in file laFile[laUrl[x]] ##reset position for subsequent reads
   end if
  end if
  delete local laFile[laUrl[x]]
  put laLoadReq[laUrl[x]] into tLoadReq ##holder
  put laUrl[x] into tUrlHolder #so we can delete in cleanUp
  put laConnectHost[laUrl[x]] into tConnectHost #holder so we can delete in clean up
  ulCleanUpFtp x

  if tLoadReq and laCancelled[tUrlHolder] then
   delete local laLoadedUrls[tUrlHolder]
   delete local laUrlLoadStatus[tUrlHolder]
   delete local laUrlErrorStatus[tUrlHolder]
   delete local laStatus[tUrlHolder]
  end if
  if not laCancelled[tUrlHolder] then
   ulSendMessage tUrlHolder
  else
   delete local laMessg[tUrlHolder]
  end if

  delete local laCancelled[tUrlHolder]
  #change dc 210702
  if tLoadReq then
   send "ulNextFtpLoadRequest" && quote & tConnectHost & quote to me in 1
milliseconds
  end if

end ulFtpStartPoint


####################FTP GET#################


on ulFtpGet x,z
  local tCmd,tReply,tNeedCWDReset,tTempPath,tStatus

  if lvFtpMode is "active" then
   put "active" into laMode[laUrl[x]]
  else
   put "passive" into laMode[laUrl[x]]
  end if

  put "contacted" into laUrlErrorStatus[laUrl[x]]
```

```
  if laLoadReq[laUrl[x]] then put "contacted" into laUrlLoadStatus[laUrl[x]]
  ulSendCallback laUrl[x],"contacted"  ##CALLBACK FEATURE
  -----TYPE--------------------
  put "TYPE I" into tCmd
  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    close socket x
    delete local laSocketUser[x]
    exit "ulFtpGet"
  end if
  ----------------------
  ##sort out file path
  if laHome[laUrl[x]] is not "/" then ##otherwise laFileName should already be OK
    if laHome[laUrl[x]] is not char 1 to length(laHome[laUrl[x]]) of
laLongFileName[laUrl[x]] then
      put laHome[laUrl[x]] before laLongFileName[laUrl[x]]
    end if
  end if

  ##SIZE get file size or CWD if a directory
  put empty into laLength[laUrl[x]] ##set up
  if last char of laLongFileName[laUrl[x]] is not "/" then ##file not directory
    put "SIZE " & laLongFileName[laUrl[x]]  into tCmd
    put false into tNeedCWDReset
    put ulFtpCommand(tCmd,x) into tReply
    ## 191002 changed following; can't use 550 response from SIZE command to
assume file can't be transferred
    ##  if item 1 of tReply is "ftpErr" or word 1 of tReply is 550 then
    if item 1 of tReply is "ftpErr"  then
      ulFtpSetError x,tReply
      close socket x
      delete local laSocketUser[x]
      exit "ulFtpGet"
    end if
    if word 1 of tReply = 213 then ##good reply
      get word 2 of tReply
      if it is an integer then
        put  it  into laLength[laUrl[x]]
      end if
    end if

  else ##need directory listing so we must CWD before getting listing

    put laLongFileName[laUrl[x]] into tTempPath
    if the length of tTempPath >1 then
```

```
    ##remove final forward slash
    delete last char of tTempPath
  end if
  put "CWD " & tTempPath into tCmd
  put true into tNeedCWDReset

  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    close socket x
    delete local laSocketUser[x]
    exit "ulFtpGet"
  end if
end if
-------------------------------------
if laMode[laUrl[x]] is "active" then
  ulTransferActive x
else
  ulTransferPassive x
end if

if laStatus[laUrl[x]] <> empty then ##failed to set up data connection
  close socket x
  delete local laSocketUser[x]
  exit ulFtpGet
end if
-------------------------------------------------
##prepare for reading data
put empty into laFtpDataDone[laUrl[x]] ##flag for checking transfer is over
put empty into laReadBytes[laUrl[x]]
if laLoadReq[laUrl[x]] then
  put empty into laLoadedurls[laUrl[x]]
else
  put empty into laData[laUrl[x]]
end if
-----RETR or LIST-------------------
if last char of laLongFileName[laUrl[x]] is not "/" then
  put "RETR " & laLongFileName[laUrl[x]] into tCmd
else ##need directory listing
  if lvFtpListCommand = "NLST" then
    put "NLST" into tCmd
  else
    put "LIST" into tCmd
  end if
end if
put ulFtpCommand(tCmd,x) into tReply
```

```
if not ulFtpGoodReply(tReply, tCmd) then
  ulFtpSetError x,tReply
  exit "ulFtpGet"
else
  put "requested" into laUrlErrorStatus[laUrl[x]]
  if laLoadReq[laUrl[x]] then put "requested" into laUrlLoadStatus[laUrl[x]]
  ulSendCallback laUrl[x],"requested"  ##CALLBACK FEATURE
end if
-----------------------------------

##blocking point ACTIVE??
if laMode[laUrl[x]] is "active" then
  repeat while  laFtpDataDone[laUrl[x]] is empty and laStatus[laUrl[x]] is empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat
end if

##blocking point PASSIVE??
if laMode[laUrl[x]] is not "active" then
  read from socket laTransPasvIP[laUrl[x]]  with message "ulGetData"
  repeat while  laFtpDataDone[laUrl[x]] is empty and laStatus[laUrl[x]] is empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat
end if

if laStatus[laUrl[x]] is not empty then ##error occurred
  if  laUrlErrorStatus[laUrl[x]] is empty then
    put "error" into laUrlErrorStatus[laUrl[x]]
  end if
  if laLoadReq[laUrl[x]] then
    put "error" into laUrlLoadStatus[laUrl[x]]
    delete local laLoadedUrls[laUrl[x]] ##clear data
  else
    delete local laData[laUrl[x]] ##clear data
  end if
  close socket x
  delete local laSocketUser[x]
else

  #now check for 226 completion

  put ulTransferCompleteResponse(x) into tReply
```

```
    if char 1 of tReply <> 2 then ## we may have 226 or 200 and treat both as
successful
      replace "ftperr," with empty in tReply
      put "error" && tReply into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] then
        put "error" into laUrlLoadStatus[laUrl[x]]
        delete local laData[laUrl[x]] ##clear data
      end if
      close socket x
      delete local laSocketUser[x]

    else ## download successful

      if tNeedCWDReset then
        ##051202 reset current directory to original
        put "CWD " & laHome[laUrl[x]] into tCmd
        put ulFtpCommand(tCmd,x) into tReply
        if not ulFtpGoodReply(tReply, tCmd) then
          ulFtpSetError x,tReply
          close socket x
          delete local laSocketUser[x]
          exit "ulFtpGet"
        end if
      end if

      put empty into laUrlErrorStatus[laUrl[x]]
      if laFile[laUrl[x]] is empty then
        put "cached" into tStatus
      else
        put "downloaded" into tStatus
      end if
      if laLoadReq[laUrl[x]] then put tStatus into laUrlLoadStatus[laUrl[x]]
      ulSendCallback laUrl[x],"downloaded"  ##CALLBACK FEATURE


    end if
    put true into laStatus[laUrl[x]] ##break wait
  end if
end ulFtpGet
############get data port from ftp server-answer to PASV##############

on ulTransferPassive x
  local y,tCmd,tReply,n1,transPasvIP,a1,a2,tPort,tErr
  set the itemDel to "|"
  put last item of x into y
  set the itemDel to comma
```

```
  put "PASV"  into tCmd
  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    exit "ulTransferPassive"
  end if

  if last char of tReply is "." then delete last char of tReply
  replace ")" with empty in tReply
  set the itemDel to "("
  put item 2 of tReply  into n1
  set itemDel to ","
  put item 1 to 4 of n1 into  transPasvIP
  replace "," with "." in transPasvIP
  put (item -2 of n1)*256  into a1
  put item  -1 of n1 into a2
  put a1+a2  into tPort

  put transPasvIP  & ":" & tPort & "|" & y into laTransPasvIP[laUrl[x]]
  get ulOpenSocket(laTransPasvIP[laUrl[x]])
  if not it then
    put "error Couldn't open passive transfer connection" into tErr
    ulFtpSetError x,tErr
    exit "ulTransferPassive"
  end if
  put x into laControlXDataMap[laTransPasvIP[laUrl[x]]] ##
end ulTransferPassive
----------------------------------------
####################Send port to server for Active transfer and listen for
data###############
on ulTransferActive x
  local thisIP,tErr,i1,i2,tCmd,tReply

  if lvDataPortCount is empty or lvDataPortCount >= 65535  then
    put 6923 into lvDataPortCount
  else
    add 1 to lvDataPortCount
  end if

  put x into laControlXLocalMap[lvDataPortCount]
  put hostAddress(x) into thisIP
  replace "." with "," in thisIP

  put lvDataPortCount into laTransActvIP[x]
  if laAction[laUrl[x]] is "putData"  then
```

```
    accept connections on port  laTransActvIP[x]  with message "ulPortMessageSend"
  else
    accept connections on port  laTransActvIP[x]  with message "ulPortMessageGet"
  end if
  if the result <> empty then
    put "error Couldn't open transfer port" into tErr
    ulFtpSetError x,tErr
    exit "ulTransferActive"
  end if

  put  laTransActvIP[x] div 256 into i1
  put  laTransActvIP[x]  mod 256 into i2

  put "PORT " & thisIP & "," & i1&","& i2  into tCmd
  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    exit "ulTransferActive"
  end if
end ulTransferActive
----------------------------------------
on ulPortMessageGet x,y
  local tControlSock,tReply

  ##active transfer message received
  put laControlXLocalMap[y] into tControlSock
  put tControlSock into laControlXDataMap[x]
  if x is among the lines of the openSockets then
    read from socket x with message "ulGetData"
    if the result <> empty then
      put the result into tReply
      ulFtpSetError tControlSock,tReply
    end if
  end if
end ulPortMessageGet

##################the ftp download routine##################
on ulGetData x,y
  local mSock,tStatusString

  put laControlXDataMap[x] into mSock
  ulStoreData laUrl[mSock],y
  add length(y) to laReadBytes[laUrl[mSock]]
  put "loading," & laReadBytes[laUrl[mSock]] & "," & laLength[laUrl[mSock]]  into
tStatusString
  put tStatusString into laUrlErrorStatus[laUrl[mSock]]
```

```
  if laLoadReq[laUrl[mSock]] then put tStatusString into laUrlLoadStatus[laUrl[mSock]]
  ulSendCallback laUrl[mSock],tStatusString  ##CALLBACK FEATURE
  if x is among the lines of the openSockets then
    read from socket x with message "ulGetData"
    if the result <> empty then
      put false into laStatus[laUrl[mSock]]
    end if
  end if
end ulGetData

##############FTP PUT ##########################

on ulFtpSend x,z
  local tErr,tNeedCWDReset,tTempPath,tCmd,tReply

  if lvFtpMode is "active" then
    put "active" into laMode[laUrl[x]]
  else
    put "passive" into laMode[laUrl[x]]
  end if

  put "contacted" into laUrlErrorStatus[laUrl[x]]
  if laLoadReq[laUrl[x]] then put "contacted" into laUrlLoadStatus[laUrl[x]]
  ulSendCallback laUrl[x],"contacted"  ##CALLBACK FEATURE
  ---------------------------------------------------------------------
  ##sort out file path
  if laHome[laUrl[x]] is not "/" then ##otherwise laFileName should already be OK
    if laHome[laUrl[x]] is not char 1 to length(laHome[laUrl[x]]) of
laLongFileName[laUrl[x]] then
      put laHome[laUrl[x]] before laLongFileName[laUrl[x]]
    end if
  end if

  ## check for valid filename
  if last char of laLongFileName[laUrl[x]] is  "/" or laLongFileName[laUrl[x]] is empty then
    put "File not specified" into tErr
    ulFtpSetError x,tErr
    exit "ulFtpSend"
  end if

  ### CWD to directory if it exists
  put false into tNeedCWDReset
  set the itemDel to "/"
  put laLongFileName[laUrl[x]] into tTempPath
  put empty into item -1 of tTempPath
  if tTempPath <> laHome[laUrl[x]] then
```

```
    delete char -1 of tTempPath
    put "CWD " & tTempPath into tCmd
    put true into tNeedCWDReset

    put ulFtpCommand(tCmd,x) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      ulMakeDirectory x,tTempPath,1 ##1 = first try
      if the result <> empty then
        ulFtpSetError x,the result
        exit "ulFtpSend"
      end if
    end if
  end if

  if tNeedCWDReset then ##RESET working directory

    put "CWD " & laHome[laUrl[x]] into tCmd
    put ulFtpCommand(tCmd,x) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      ulFtpSetError x,tReply
      close socket x
      delete local laSocketUser[x]
      exit "ulFtpSend"
    end if
  end if

  ----------------------------------------------------------------------
  put "TYPE I" into tCmd
  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    exit "ulFtpSend"
  end if

  if laFile[laUrl[x]] is empty then
    put length(laPostData[laUrl[x]]) into laLength[laUrl[x]]
  else
    put ulFileLength(laFile[laUrl[x]]) into laLength[laUrl[x]]
  end if
  put empty into laWriteBytes[laUrl[x]]
  put empty into laFtpDataDone[laUrl[x]] ##used below to control exit from
ulFtpSend
  if laMode[laUrl[x]] is "active" then
    ulTransferActive x
  else
    ulTransferPassive x
```

```
  end if
  if laStatus[laUrl[x]] is not empty then ##couldn't make data connection
    exit "ulFtpSend"
  end if

  put "STOR "  & laLongFileName[laUrl[x]] into tCmd


  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    ulFtpSetError x,tReply
    exit "ulFtpSend"
  end if
  put "requested" into laUrlErrorStatus[laUrl[x]]
  if laLoadReq[laUrl[x]] then put "requested" into laUrlLoadStatus[laUrl[x]]
  ulSendCallback laUrl[x],"requested"  ##CALLBACK FEATURE

  ##Block here while sending data
  if laMode[laUrl[x]] is not "active" then ulSendDataP x
  repeat while laFtpDataDone[laUrl[x]] is empty and laStatus[laUrl[x]] is empty##waiting
for write to complete
    if lvJumpOut then exit to top
    wait for messages
  end repeat
  if laStatus[laUrl[x]] <> empty then ##error occurred
    if  laUrlErrorStatus[laUrl[x]] is empty then
      put "error" into laUrlErrorStatus[laUrl[x]]
    end if
    put empty into laData[laUrl[x]] ##clear data
    if laLoadReq[laUrl[x]] then
      put "error" into laUrlLoadStatus[laUrl[x]]
      delete local laData[laUrl[x]] ##clear data
    end if
    close socket x
    delete local laSocketUser[x]
  else
    ##look for 226 response

    put ulTransferCompleteResponse(x) into tReply

    if char 1 of tReply <> 2 then ## we treat 226 or 200 as successful -- see
ulTransferCompleteResponse
      replace "ftperr," with empty in tReply
      put "error" && tReply into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] then
        put "error" into laUrlLoadStatus[laUrl[x]]
```

```
      delete local laData[laUrl[x]] ##clear data
    end if
    close socket x
    delete local laSocketUser[x]
  else

    put empty into laUrlErrorStatus[laUrl[x]]
    if laLoadReq[laUrl[x]] then
      put "uploaded" into laUrlLoadStatus[laUrl[x]]
      delete local laData[laUrl[x]] ##clear data
    end if
    ulSendCallback laUrl[x],"uploaded"  ##CALLBACK FEATURE
  end if
  put true into laStatus[laUrl[x]]
 end if

end ulFtpSend
-----------------------------------------
on ulMakeDirectory x, pDir
 local tTempPath,tCmd,tReply

 -- first we CWD to the parent directory
 set the itemDel to "/"
 put pDir into tTempPath
 delete item -1 of tTempPath ##parent directory
 if tTempPath is empty then put "/" into tTempPath  ## root directory
 put empty into lvNeedDir

 put "CWD " & tTempPath into tCmd

 put ulFtpCommand(tCmd,x) into tReply
 if not ulFtpGoodReply(tReply, tCmd) then
  if tTempPath = laHome[laUrl[x]] then
   return "error Unable to create directory path"
  else
   ulMakeDirectory x,tTempPath
   if the result <> empty then
    return the result
   end if
  end if
 end if

 put "MKD " & pDir into tCmd
 put ulFtpCommand(tCmd,x) into tReply
 if not ulFtpGoodReply(tReply, tCmd) then
  return "error Unable to create directory path"
```

```
  else  ##now CWD to the created directory
    put "CWD " & pDir into tCmd
    put ulFtpCommand(tCmd,x) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      return tReply
    else
      return empty
    end if
  end if

end ulMakeDirectory

############ACTIVE UPLOAD############
on ulPortMessageSend x,y
  ##active transfer message received
  local tControlSock,nData,tStatusString

  put laControlXLocalMap[y] into tControlSock
  put tControlSock into laControlXDataMap[x]
  put ulNextData(laUrl[tControlSock]) into nData

  if nData <> empty then
    add length(nData) to laWriteBytes[laUrl[tControlSock]]
    put "uploading, " & laWriteBytes[laUrl[tControlSock]] & "," &
laLength[laUrl[tControlSock]] into tStatusString
    put tStatusString into laUrlErrorStatus[laUrl[tControlSock]]

    write nData to  socket x  with  message "ulWriteMoreA"
    if the result <> empty then
      put "error" && the result into laUrlErrorStatus[laUrl[tControlSock]]
      if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[tControlSock]]
      put false into laStatus[laUrl[tControlSock]]
    end if
  else
    ##put false into laStatus[laUrl[tControlSock]]
    ##added on 1.1.5b4
    put true into laFtpDataDone[laUrl[tControlSock]]

    put empty into laUrlErrorStatus[laUrl[tControlSock]]
    close socket x
    close socket y  ##local port
    delete local laControlXDataMap[x]

  end if

end ulPortMessageSend
```

```
----------------------------------------

on ulWriteMoreA x
  local tId, nData,tStatusString

  put laControlXDataMap[x] into tId

  put ulNextData(laUrl[tId]) into nData
  if nData <> empty then
    add length(nData) to laWriteBytes[laUrl[tId]]
    put "uploading," & laWriteBytes[laUrl[tId]] & "," & laLength[laUrl[tId]] into tStatusString
    put tStatusString into laUrlErrorStatus[laUrl[tId]]
    if laLoadReq[laUrl[tId]] then put tStatusString into laUrlLoadStatus[laUrl[tId]]
    ulSendCallback laUrl[tId],tStatusString  ##CALLBACK FEATURE
    write nData to  socket x  with message "ulWriteMoreA"
    if the result <> empty then
      put false into laStatus[laUrl[tId]]
    end if
  else
    put true into laFtpDataDone[laUrl[tId]]

    put empty into laUrlErrorStatus[laUrl[tId]]
    close socket x
    close socket laTransActvIP[tID]  ##local port
    delete local laControlXDataMap[x]
  end if

end ulWriteMoreA

##########PASSIVE UPLOAD############

on ulSendDataP x
  local nData,tStatusString

  put ulNextData(laUrl[x]) into nData
  if nData <> empty then
    add length(nData) to laWriteBytes[laUrl[x]]
    put "uploading," &  laWriteBytes[laUrl[x]] & "," & laLength[laUrl[x]] into tStatusString
    put tStatusString into laUrlErrorStatus[laUrl[x]]
    write nData to  socket laTransPasvIP[laUrl[x]]  with  message "ulWriteMoreP"
    if the result <> empty then
      put false into laStatus[laUrl[x]]
    end if
  else
    ##put false into laStatus[laUrl[x]]
    ## changed 1.1.5b4
```

```
    close socket laTransPasvIP[laUrl[x]]  ##close data socket here
    delete local laControlXDataMap[laTransPasvIP[laUrl[x]] ]
    put empty into laUrlErrorStatus[laUrl[x]]
    put true into laFtpDataDone[laUrl[x]] #set flag before closing socket

  end if
end ulSendDataP
-----------------------------------------------------------------
on ulWriteMoreP x
  local mSock,nData,tStatusString

  put laControlXDataMap[x] into mSock
  put ulNextData(laUrl[mSock]) into nData
  if nData <> empty then
    add length(nData) to laWriteBytes[laUrl[mSock]]
    put "uploading, " & laWriteBytes[laUrl[mSock]] & "," & laLength[laUrl[mSock]]  into
tStatusString
    put tStatusString into laUrlErrorStatus[laUrl[mSock]]
    if laLoadReq[laUrl[mSock]] then put tStatusString into laUrlLoadStatus[laUrl[mSock]]
    ulSendCallback laUrl[mSock],tStatusString  ##CALLBACK FEATURE
    if laStatus[laUrl[mSock]] is empty then
      write nData to  socket x  with message "ulWriteMoreP"
    end if
    if the result <> empty then
      put false into laStatus[laUrl[mSock]]
    end if
  else
    close socket x ##close data socket here
    delete local laControlXDataMap[x]
    put empty into laUrlErrorStatus[laUrl[mSock]]
    put true into laFtpDataDone[laUrl[mSock]] #set flag before closing socket
  end if
end ulWriteMoreP

####################FTP DELETE###############

on ulFtpDelete x
  local tCmd,tReply,mType

  ###########make sure we use the full path
  ##sort out file path
  if laHome[laUrl[x]] is not "/" then ##otherwise laFileName should already be OK
    if laHome[laUrl[x]] is not char 1 to length(laHome[laUrl[x]]) of
laLongFileName[laUrl[x]] then
      put laHome[laUrl[x]] before laLongFileName[laUrl[x]]
    end if
```

```
  end if

  if last char of laLongFileName[laUrl[x]] is "/" then
    #delete directory
    put "RMD "  &  laLongFileName[laUrl[x]] into tCmd
    put "directory" into mType
  else
    #delete file
    put "DELE "  &  laLongFileName[laUrl[x]] into tCmd
    put "file" into mType
  end if

  put ulFtpCommand(tCmd,x) into tReply
  if not ulFtpGoodReply(tReply, tCmd) then
    replace "ftpErr," with empty in tReply
    put "error" && tReply into laUrlErrorStatus[laUrl[x]]
  else
    put empty into laUrlErrorStatus[laUrl[x]]
  end if
  put true into laStatus[laUrl[x]]
end ulFtpDelete

##########################################################
on socketClosed x
  local tStatus,tControlSocket

  if laSocketClosedByScript[x] then
    ##workaround on OS X where closing a socket produes a socketClosed
message
    delete local laSocketClosedByScript[x]
    exit socketClosed
  end if
  ulLogIt "CLOSED" && x & cr##LOG
  delete local laSocketUser[x] ##reference for allocating sockets for FTP logons
  ##need to check whether ftp data port or not
  if x is among the lines of keys(lvSocketToken) then ##trying to open a socket
    put "socket closed" into lvSocketToken[x]

  else  if x is lvFtpCommandSocket then ##handling libUrlFtpCommand
    if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for a
server reply
      put "socket closed" into laFTPCommandStatus[x] ##unblock waits
    end if
  else if x is among the lines of keys(laUrl) then ##http or ftp control socket
    ## check for two situations here
    ## first is a premature close on a socket when we know the data length
```

```
    ## second is for cases when we don't know the data length
    ## a normal close when we  know the data length isn't handled here
    if laLength[laUrl[x]] > laReadBytes[laUrl[x]] then ##fixed dc 250103
      put "Socket closed before end of file" into laUrlErrorStatus[laUrl[x]]
      if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
      put "false" into laStatus[laUrl[x]] ##unblock waits
      if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for
a server reply
        put "socket closed" into laFTPCommandStatus[x] ##unblock waits
      end if
    else if laLength[laUrl[x]] is empty and char 1 to 4 of laUrl[x] is "http" then ##when we
don't have a length

      ----------------------
      ##assume download completed
      if char 1 of laStatusCode[laUrl[x]] = 2 then
        if laFile[laUrl[x]] is empty then
          put "cached" into tStatus
        else
          put "downloaded" into tStatus
        end if
        put empty into laUrlErrorStatus[laUrl[x]]
        if  laLoadReq[laUrl[x]] is "true" then put tStatus into laUrlLoadStatus[laUrl[x]]
        ulSendCallback laUrl[x],"downloaded"  ##CALLBACK FEATURE
      else
        put "error" && laStatusMessage[laUrl[x]] into laUrlErrorStatus[laUrl[x]]
        if  laLoadReq[laUrl[x]] is "true" then put "error" into laUrlLoadStatus[laUrl[x]]
      end if

      ----------------------

      put true into  laStatus[laUrl[x]] ##unblock waits
      put "true" into  laHttpDataDone[laUrl[x]] ##unblock waits
    end if

  else if x is among the lines of keys(laControlXDataMap)then##must be ftp remote
data socket
    put laControlXDataMap[x] into tControlSocket
    put true into laFtpDataDone[laUrl[tControlSocket]]
    if laTransActvIP[tControlSocket] is among the lines of the openSockets then
      close socket laTransActvIP[tControlSocket] ##local data port during active
transfers
    end if
    delete local laControlXDataMap[x]
  else
    pass socketClosed
```

```
  end if
end socketClosed
----------------------------------------
##SocketTimeout defaults to 10000 milliseconds.
##To change that one can set the "socketTimeoutInterval" to a different value.

on socketTimeout x
  local tControlSocket

  ulLogit "socket timeout" && x & cr ##LOG
  ##need to check whether data port or not
  if x is among the lines of keys(lvSocketToken) then ##trying to open a socket
    put "socket timeout" into lvSocketToken[x]
    delete local laSocketUser[x]
  else if x is lvFtpCommandSocket then ##handling libUrlFtpCommand
    if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for
a server reply
      put "socket timeout" into laFTPCommandStatus[x] ##unblock waits
    end if
  else if x is among the lines of keys(laUrl) then ##http or ftp control socket
    if laLoadReq[laUrl[x]] then put "timeout" into laUrlLoadStatus[laUrl[x]]
    put "socket timeout" && x into laUrlErrorStatus[laUrl[x]]
    put "false" into laStatus[laUrl[x]] ##unblock waits
    if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for
a server reply
      put "socket timeout" into laFTPCommandStatus[x] ##unblock waits
    end if
    close socket x
    delete local laSocketUser[x]
  else if x is among the lines of keys(laControlXDataMap)then##must be ftp remote
data socket
    put laControlXDataMap[x] into tControlSocket
    put false into laStatus[laUrl[tControlSocket]] ##unblock waits
    put "socket timeout" into laFTPCommandStatus[tControlSocket]
    if laLoadReq[laUrl[x]] then put "timeout" into laUrlLoadStatus[laUrl[tControlSocket]]
    put "socket timeout" && x into laUrlErrorStatus[laUrl[tControlSocket]]
  else if x is a number then##local port for active ftp transfer
    put laControlXLocalMap[x] into tControlSocket
    if tControlSocket <> empty then
      put false into laStatus[laUrl[tControlSocket]] ##unblock waits
      put "socket timeout" into laFTPCommandStatus[tControlSocket]
      if laLoadReq[laUrl[tControlSocket]] then put "timeout" into
laUrlLoadStatus[laUrl[tControlSocket]]
      put "socket timeout" && x into laUrlErrorStatus[laUrl[tControlSocket]]
    end if
  else
```

```
    pass socketTimeout
  end if

end socketTimeout
----------------------------------------
on socketError x, pErr
  local tControlSocket

  ulLogit "socket error" && x & cr & pErr & cr ##LOG
  ##need to check whether data port or not
  if pErr is empty then put "unknown error" into pErr
  if x is among the lines of keys(lvSocketToken) then ##trying to open a socket
    put pErr into lvSocketToken[x]

  else  if x is lvFtpCommandSocket then ##handling libUrlFtpCommand
    if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for
a server reply
      put "socket error" into laFTPCommandStatus[x] ##unblock waits
    end if
  else if x is among the lines of keys(laUrl) then ##http or ftp control socket
    if laLoadReq[laUrl[x]] then put "error" into laUrlLoadStatus[laUrl[x]]
    put "error" && pErr into laUrlErrorStatus[laUrl[x]]
    put "false" into laStatus[laUrl[x]] ##unblock waits
    if x is among the lines of keys(laFTPCommandStatus) then ##may be waiting for
a server reply
      put pErr into laFTPCommandStatus[x] ##unblock waits
    end if

    ## Handle cleanup for load url
    if laLoadReq[laUrl[x]] then
      ulHttpLateCleanUp x
    end if

  else if x is among the lines of keys(laControlXDataMap)then##must be ftp passive
data socket
    put laControlXDataMap[x] into tControlSocket
    put false into laStatus[laUrl[tControlSocket]] ##unblock waits
    put pErr into laFTPCommandStatus[tControlSocket]
    if laLoadReq[laUrl[tControlSocket]] then put "error" into
laUrlLoadStatus[laUrl[tControlSocket]]
    put "error" && pErr into laUrlErrorStatus[laUrl[tControlSocket]]

  else if x is a number then##local port for active ftp transfer
    put laControlXLocalMap[x] into tControlSocket
    if tControlSocket <> empty then
      put false into laStatus[laUrl[tControlSocket]] ##unblock waits
```

```
        put pErr into laFTPCommandStatus[tControlSocket]
        if laLoadReq[laUrl[tControlSocket]] then put "error" into
laUrlLoadStatus[laUrl[tControlSocket]]
        put "error" && pErr into laUrlErrorStatus[laUrl[tControlSocket]]
      end if
    else
      pass socketError
    end if
end socketError
-----------------------------------------
on libUrlResetAll
  local i

  -- CW-2016-06-11: [[ External driver support ]] Call driver specific reset command
if external driver is in use.
  if lvExtDriver is not empty then
    ulDeleteLocals
    ulExtResetDriver
  else
    if there is a stack "libUrl" then put empty into fld "log1" of stack "libURL"
    repeat for each line i in the openSockets
      close socket i
    end repeat

    ulDeleteLocals
    put true into lvJumpOut
    send "ulDeleteLocals" to me in 5 milliseconds
  end if
end libUrlResetAll

# OK-2008-09-19 : Added for revOnline to allow post commands to be cancelled,
# not sure if this works yet...

command libUrlCancel
   -- CW-2016-06-11: [[ External driver support ]] Call driver specific reset
command if external driver is in use.
   if lvExtDriver is not empty then
      ulExtResetDriver
   else
      put true into lvJumpOut
   end if
end libUrlCancel

-----------------------------------------
on ulDeleteLocals
  local e
```

```
  repeat for each item e in line 3 of the localNames
    -- CW-2016-06-11: [[ External driver support ]] Don't clear locals that determine
what driver is in use.
    if e is not "lvExtDriver" then
      get "delete" && "local" && e
      do it
    end if
  end repeat
end ulDeleteLocals
-----------------------------------------------
on resetAll
  ## included for compatibility with previous versions
  ##ibUrlResetAll should be used instead
  libUrlResetAll
end resetAll
-----------------------------------------------
on ulFtpStopWatch x
  local tCmd

  if x is among the lines of the OpenSockets then
    if lvFtpStopTime is empty or lvFtpStopTime is not a number then
      put 15 into lvFtpStopTime
    end if
    switch
    case laStopSec[x] >= lvFtpStopTime
      delete local laStopSec[x]
      delete local laStopUnit[x]
      put  "QUIT" into tCmd
      get ulFtpCommand(tCmd,x)
      delete local laFtpCommandStatus[x]
      # write "QUIT" & CRLF to socket x ##tidy finish
      close socket x
      delete local laSocketUser[x]
      break
    case  laStopUnit[x]=1
      add laStopUnit[x] to laStopSec[x]
      send "ulFtpStopWatch " & x to me in 1 sec
      break
    case laStopUnit[x] = 0
      break
    end switch
  else
    delete local laStopSec[x]
    delete local laStopUnit[x]
  end if
```

```
end ulFtpStopWatch
-----------------------------------------
on libUrlFtpUpload pData,pUrl,pMessage
  local newUrl

  put false into lvJumpOut
  put ulStripUrl(pUrl) into newUrl
  if lvCount is empty then
    put "6923" into lvCount
  else
    #add 1 to lvCount
  end if
  switch
  case newUrl is among the keys of laLoadingUrls
    ##don't allow loads if the same url is waiting to load
    return "error  URL is currently loading" with urlResult empty
    break
  default
    put pData into laPostData[newUrl]
    if pMessage <> empty then
      put ulGetCaller(), pMessage into laMessg[newUrl]
      --put the long id of the target &","& pMessage into laMessg[newUrl]
    end if
    put true into laLoadReq[newUrl]
    put 1 into laLoadingUrls[newUrl] #for tracking
    put "putData" into laAction[newUrl]
    put empty into laUrlErrorStatus[newUrl]
    put empty into laUrlLoadStatus[newUrl]
    put empty into laData[newUrl]
    ulGetFormat newUrl,lvCount
    if laUrlLoadStatus[newUrl] is "error" and not laCancelled[newUrl] then
      ulSendMessage newUrl ##send message now only if error occurred
      return "error"
    else if laCancelled[newUrl] then
      ##user cancelled after starting but before blocking point
      delete local laLoadedUrls[newUrl]
      delete local laUrlLoadStatus[newUrl]
      delete local laUrlErrorStatus[newUrl]
      delete local laStatus[newUrl]
      delete local laCancelled[newUrl]

    else
      return empty
    end if
    break
  end switch
```

```
end libUrlFtpUpload
-----------------------------------------
on libUrlFtpUploadFile pFile,pUrl,pMessage
  local newUrl

  put false into lvJumpOut
  put ulStripUrl(pUrl) into newUrl

  #removed from here in 1.1.6b1
  #  open file pFile for binary read
  #  if the result is not empty then
  #    return the result
  #  end if
  #  put newUrl into laUrlByFile[pFile]

  if lvCount is empty then
    put "6923" into lvCount
  else
    #add 1 to lvCount
  end if
  switch
  case newUrl is among the keys of laLoadingUrls
    ##don't allow loads if the same url is waiting to load
    return "error  URL is currently loading" with urlResult empty
    break
    #case newUrl is not among the lines of  the keys of laLoadedUrls OR
laUrlLoadStatus[newUrl] is not "cached"
  default

    if pMessage <> empty then
      put ulGetCaller(), pMessage into laMessg[newUrl]
      --put the long id of the target &","& pMessage into laMessg[newUrl]
    end if
    put true into laLoadReq[newUrl]
    put pFile into laFile[newUrl]
    put newUrl into laUrlByFile[pFile] ## added here for 1.1.6b1
    put 1 into laLoadingUrls[newUrl] #for tracking
    put "putData" into laAction[newUrl]
    put empty into laUrlErrorStatus[newUrl]
    put empty into laUrlLoadStatus[newUrl]
    #put empty into laData[newUrl]
    ulGetFormat newUrl,lvCount
    if laUrlLoadStatus[newUrl] is "error" and not laCancelled[newUrl] then
      ulSendMessage newUrl ##send message now only if error occurred
      return "error"
    else if laCancelled[newUrl] then
```

```
      ##user cancelled after starting but before blocking point
      delete local laLoadedUrls[newUrl]
      delete local laUrlLoadStatus[newUrl]
      delete local laUrlErrorStatus[newUrl]
      delete local laStatus[newUrl]
      delete local laCancelled[newUrl]

    else
      return empty
    end if
    break
  end switch
end libUrlFtpUploadFile

-----------------------------------------------
on libUrlDownloadToFile pUrl,pFile,pMessage
  local newUrl

  put false into lvJumpOut
  put ulStripUrl(pUrl) into newUrl

  ##uncommented for 1.1.6b1 ## now opened elsewhere
  #  open file pFile for binary write
  #  if the result is not empty then
  #    return the result
  #  end if
  if lvCount is empty then
    put "6923" into lvCount
  else
    #add 1 to lvCount
  end if
  switch
  case newUrl is among the keys of laLoadingUrls
    ##don't allow loads if the same url is waiting to load
    return "error  URL is currently loading" with urlResult empty
    break
    #case newUrl is not among the lines of  the keys of laLoadedUrls OR
laUrlLoadStatus[newUrl] is not "cached"
  default
    if pMessage <> empty then
      put ulGetCaller(), pMessage into laMessg[newUrl]
     --put the long id of the target &","& pMessage into laMessg[newUrl]
    end if
    put true into laLoadReq[newUrl]
    put pFile into laFile[newUrl]
    put 1 into laLoadingUrls[newUrl] #for tracking
```

```
    put "getData" into laAction[newUrl]
    put empty into laUrlErrorStatus[newUrl]
    put empty into laUrlLoadStatus[newUrl]

    ulGetFormat newUrl,lvCount

    if laUrlLoadStatus[newUrl] is "error" and not laCancelled[newUrl] then
      ulSendMessage newUrl ##send message now only if error occurred
      return "error"
    else if laCancelled[newUrl] then
      ##user cancelled after starting but before blocking point
      delete local laLoadedUrls[newUrl]
      delete local laUrlLoadStatus[newUrl]
      delete local laUrlErrorStatus[newUrl]
      delete local laStatus[newUrl]
      delete local laCancelled[newUrl]

    else
      return empty
    end if
    break
  end switch
end libUrlDownloadToFile

-----------------------------------------------
function libUrlErrorData pUrl
  return laUrlErrorStatus[pUrl]
end libUrlErrorData
-----------------------------------------------
on libUrlSetFtpMode pMode
  ##default to passive
  -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
  if lvExtDriver is not empty then
    ulExtSetFtpMode pMode
  else
    if pMode is "active" or pMode is "a" then
      put "active" into lvFtpMode
    else
      put "passive" into lvFtpMode
    end if
  end if
end libUrlSetFtpMode

----------------------------
on libUrlSetFtpListCommand pCommand
```

```
  -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
  if lvExtDriver is not empty then
    ulExtSetFtpListCommand pCommand
  else
    if pCommand is "NLST" then
      put "NLST" into lvFtpListCommand
    else
      put "LIST" into lvFtpListCommand
    end if
  end if
end libUrlSetFtpListCommand
----------------------------
function libUrlVersion
  return the cVersion of me
end libUrlVersion
----------------------------
on libUrlSetLogField pField
  local tField

  if word 1 of pField is "field" then
    put pField into tField
  else
    if pField is a number then
      put "field" && pField into tField
    else
      if word 1 of pField is "id" then
        put "field" && pField into tField
      else
        put "field" && quote & pField & quote into tField
      end if
    end if
  end if
  if exists(tField) then
    put the long ID of tField into tField##standardise
    put tField into lvLogField
  else
    put empty into lvLogField
  end if

  -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
  if lvExtDriver is not empty then
    ulExtSetLogField lvLogField
  end if
end libUrlSetLogField
```

```
----------------------------
on libUrlSetStatusCallback pMessage,pObject
  ##pObject must be a long ID
  ##Allow empty value in which case we use send to self.
  if pMessage is not empty and the paramCount is 1 then
    put pMessage & comma into lvStatusCallback
  else if pMessage is not empty and the paramCount is 2 then
    if not exists(pObject) then
      return "invalid callback target object" for error
    end if
    put pMessage & comma & pObject into lvStatusCallback
  else
    put empty into lvStatusCallback
  end if
end libUrlSetStatusCallback
----------------------------
function libUrlLastHttpHeaders
  return the lastHttpHeaders of me
end libUrlLastHttpHeaders
----------------------------
on libUrlSetCustomHttpHeaders pHeaders
  set the customHTTPHeaders of me to pHeaders
end libUrlSetCustomHttpHeaders
----------------------------
function libUrlLastRhHeaders
  return the lastRhHeaders of me
end libUrlLastRhHeaders
----------------------------
on libUrlSetFtpStopTime pSecs
  if pSecs is empty or pSecs < 1 or pSecs is not a number then
    put 15 into lvFtpStopTime
  else
    put pSecs into lvFtpStopTime
  end if
end libUrlSetFtpStopTime
----------------------------
on libUrlSetExpect100 pLimit
  if pLimit is empty or pLimit is not a integer then
    put empty into laMaxPostWithoutExpect
  else
    put pLimit into laMaxPostWithoutExpect
  end if
end libUrlSetExpect100
----------------------------
function libUrlFormData
  local tNumParams,tNumParts,tPart,tFormString,i
```

```
  put the paramcount into tNumParams
  put tNumParams div 2 into tNumParts
  if tNumParts < 1 then return empty

  put 0 into tPart
  put empty into tFormString
  repeat with i = 1 to tNumParams

    if i mod 2 = 1 then #stage 1 of part
      add 1 to tPart
      if tPart > tNumParts then exit repeat
      put urlEncode(param(i)) after tFormString
    else
      put "=" & urlEncode(param(i)) & "&" after tFormString
    end if
  end repeat
  delete char -1 of tFormString
  return tFormString
end libUrlFormData

--------------------------
function libUrlMultipartFormData @pFormData, pParam
  local tNumParams,tNumParts,tBoundary,tKeys,tKey,tValue,tPart
  local tFile,tNumFiles,tFilepath,tFilename,tSubBoundary

  put the paramcount into tNumParams
  if tNumParams < 1 then return "error Insufficient parameters"
  put (tNumParams - 1) div 2 into tNumParts

  put empty into pFormData ##ensure it is empty

  ##create initial boundary
  put "__Part__" into tBoundary
  put ulFormBoundary() after tBoundary

  put "Content-type: multipart/form-data; boundary=" & quote & tBoundary & quote &
return into pFormData

  if tNumParams < 2 then
    ## do nothing for now, this will create empty shell for use with
libUrlMultipartFormAddPart
  else if tNumParams = 2 then ##treat as array
    put keys(pParam) into tKeys
    sort tKeys numeric
    repeat for each line tKey in tKeys
```

```
      put "--" & tBoundary & CRLF after pFormData
      put "Content-Disposition: form-data; name=" & quote & item 1 of pParam[tKey] &
quote after pFormData
      put item 2 to -1 of pParam[tKey] into tValue
      if char 1 to 6 of tValue = "<file>" then
        put tValue into tFile
        put empty into tValue
        put char 7 to -1 of tFile into tFilepath
        set the itemDel to "/"
        put item -1 of tFilepath into tFilename
        set the itemDel to comma
        put ";" && "filename=" & quote & tFilename & quote after pFormData
        put CRLF & "Content-Type: application/octet-stream" after pFormData
        put url ("binfile:" & tFilepath) into tValue
        if the result <> empty then
          return "error" && the result
        end if
      end if
      put CRLF & CRLF after pFormData
      put tValue after pFormData
      put CRLF after pFormData

    end repeat

  else if tNumParts > 0 then ##treat as key/value pairs
    put 0 into tPart
    repeat with i = 2 to tNumParams

      if i mod 2 = 0 then ##stage 1 of part
        add 1 to tPart
        if tPart > tNumParts then exit repeat
        put "--" & tBoundary & CRLF after pFormData
        put "Content-Disposition: form-data; name=" & quote & param(i) & quote after
pFormData

      else
        put param(i) into tValue

        if char 1 to 6 of tValue = "<file>" then
          put tValue into tFile
          put empty into tValue
          put char 7 to -1 of tFile into tFilepath
          set the itemDel to "/"
          put item -1 of tFilepath into tFilename
          set the itemDel to comma
          put ";" && "filename=" & quote & tFilename & quote after pFormData
```

```
        put CRLF & "Content-Type: application/octet-stream" after pFormData
        put url ("binfile:" & tFilepath) into tValue
        if the result <> empty then
          return "error" && the result
        end if
      end if
      put CRLF & CRLF after pFormData
      put tValue after pFormData
      put CRLF after pFormData
    end if
  end repeat

 end if

 put  "--" & tBoundary & "--" after pFormData ##end boundary
 return empty

end libUrlMultipartFormData
-------------------------
function libUrlMultipartFormAddPart
@pFormData,pName,pValue,pMimeType,pEncoding
 local tLastLine,tBoundary,tPartData,tFile,tNumFiles,tFilepath,tFilename
 local tSubBoundary,tSubEncodings,tSubMimeTypes

 put the number of lines of pFormData into tLastLine
 put line -1 of pFormData into tBoundary
 delete char -2 to -1 of tBoundary ##remove final 2 hyphens
 put tBoundary & CRLF into tPartData
 put "Content-Disposition: form-data; name=" & quote & pName & quote after tPartData
 if char 1 to 6 of pValue is "<file>" then

  put   pValue into tFile
  put empty into pValue
  put char 7 to -1 of tFile into tFilepath

  set the itemDel to "/"
  put item -1 of tFilepath into tFilename
  set the itemDel to comma
  put ";" && "filename=" & quote & tFilename & quote after tPartData

  put url ("binfile:" & tFilepath) into pValue
  if the result <> empty then return "error" && the result

  if pMimeType is empty then put "application/octet-stream" into pMimeType
  if pEncoding is empty then
    put "Binary" into pEncoding
```

```
    end if

  end if
  if pMimeType <> empty then
    put CRLF & "Content-Type:" && pMimeType after tPartData
  end if
  if pEncoding <> empty then
    put CRLF & "Content-transfer-encoding:" && pEncoding after tPartData
  end if
  put CRLF & CRLF after tPartData
  put pValue after tPartData
  put CRLF after tPartData
  put tBoundary & "--" after tPartData
  put tPartData into line -1 of pFormData
  return empty
end libUrlMultipartFormAddPart
---------------------------
##new 1.015b2
on libUrlSetAuthCallback pMethod, pMessage
  local tCount,tLine

  put 0 into tCount
  repeat for each line tLine in lvAuthCallbacks
    add 1 to tCount
    if pMethod is item 1 of tLine then
      delete line tCount of lvAuthCallbacks
      exit repeat
    end if
  end repeat

  if pMessage <> empty then
    put pMethod & "," & pMessage & "," & the long id of the target & return after
lvAuthCallbacks
  end if
end libUrlSetAuthCallback
-----------------------
on libUrl_authcb_Resend pUrl

  switch  laAction[pUrl]
  case "getData"
    put true into lvAuthBlockBypass
    get url pUrl

    break

  case "postData"
```

```
    put true into lvAuthBlockBypass
    post laPostData[pUrl] to url pUrl
    break
  default
    return empty
    break
  end switch

  put false into lvAuthBlockBypass
  return 1
end libUrl_authcb_Resend
----------------------
on libUrl_authcb_SetAuthToken pUrl, pHeaderString, pToken, pUseAgain
  local tRealm, tRegEx,tKey,tKeys,tNeedToAdd,tThisKey,tOldUrl

  put "realm=(" & quote & ".+?" & quote & ")" into tRegEx
  get matchText(pHeaderString, tRegEx,tRealm)
  if pUseAgain is not true then put false into pUseAgain ##default to false

  if "proxy" is in word 1 of pHeaderString then
    set the itemDel to "|"
    put item 1 of laConnectHost[pUrl] after tKey
    set the itemDel to comma
    put "," after tKey
    put tRealm after tKey
    put pUseAgain & "," & pToken into laProxyAuthTokens[tKey]
  else ##assume it is "www"
    set the itemDel to "/"
    put item 1 to -2 of pUrl after tKey
    set the itemDel to comma
    put "," after tKey
    put tRealm after tKey
    put keys(laServerAuthTokens) into tKeys
    filter tKeys with "*," & tRealm
    put true into tNeedToAdd
    repeat for each line tThisKey in tKeys
      put item 1 of tThisKey into tOldUrl
      if  tOldUrl is in pUrl then ##current one shorter so leave
        ## put false into tNeedToAdd
        put tThisKey into tKey
        exit repeat
      else if pUrl is in tOldUrl then ##replace old with new
        delete local laServerAuthTokens[tThisKey]
        exit repeat
      end if
```

```
    end repeat
    if tNeedToAdd then
      put pUseAgain & "," & pToken into laServerAuthTokens[tKey]
    end if
  end if

end libUrl_authcb_SetAuthToken
----------------------
function libUrlBasicAuthToken pName, pPass
  get "Basic" && base64Encode(pName & ":" & pPass)
  replace return with empty in it ##in case of long passwords
  return it
end libUrlBasicAuthToken
----------------------
on libUrlSetSSLVerification pWhich
  -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
  if lvExtDriver is not empty then
    ulExtSetSSLVerification pWhich
  else
    if pWhich is false then
      put false into lvSSLVerification
    else
      put true into lvSSLVerification
    end if
  end if
end libUrlSetSSLVerification
----------------------
on libUrlFollowHttpRedirects pWhich
  -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
  if lvExtDriver is not empty then
    ulExtFollowHttpRedirects pWhich
  else
    if pWhich is false then
      put false into lvFollowHttpRedirects
    else
      put true into lvFollowHttpRedirects
    end if
  end if
end libUrlFollowHttpRedirects
----------------------
on ulSendAuthMessage pMethod, pUrl, pHeaderString
  local tCount,tLine,tMessage,tObject

  put 0 into tCount
```

```
  repeat for each line tLine in lvAuthCallbacks
    add 1 to tCount
    if item 1 of tLine is pMethod then
      put item 2 of tLine into tMessage
      put item 3 of tLine into tObject
      exit repeat
    end if
  end repeat
  if tMessage <> empty and exists(tObject) then
    send tMessage &&  "pUrl, pHeaderString" to tObject
    return the result ## must ensure this contains data
  else
    return empty ##possible problems
  end if

end ulSendAuthMessage
-------------------------
on ulSendMessage pUrl
  local xmessg,omessg

  ##send any requested message on completion
  if laMessg[pUrl] is not empty then
    if item 2 of laMessg[pUrl] is not quote & "" & quote  then

      put item 1 to -2 of laMessg[pUrl] into xmessg  ##modified dc 220905
      put item -1 of laMessg[pUrl] into omessg  ##modified dc 220905
      replace quote with empty in omessg
      if there is a xmessg then
        send omessg && quote & pUrl & quote & "," & laUrlLoadStatus[pUrl]  to xmessg in
0 milliseconds
      end if
    end if
  end if
  delete local laMessg[pUrl]
end ulSendMessage
---------------------------
on ulSendCallback pUrl, pStatus
  local tMessage,tObject,tSendStr

  if lvStatusCallback is empty then exit ulSendCallback
  put item 1 of lvStatusCallback into tMessage

  put item 2 to -1 of lvStatusCallback into tObject   ##modified dc 220905
  put tMessage && "pURL, pStatus" into tSendStr

  if exists(tObject) then
```

```
    ## need quotes for the url formatting and the possibility of multiple items in
second argument
    send tSendStr to tObject in 0 milliseconds
   else
    # If no target then just send to self so it propagates through message path.
    send tSendStr to me in 0 milliseconds
  end if
end ulSendCallback
-----------------------------
function ulFileLength pFile
  local tSavedDir,tDir,tFileName,tFileData

  if there is a file pFile then
    put the directory into tSavedDir
    put pFile into tDir
    set the itemDel to "/"
    put item -1 of pFile into tFileName
    delete item -1 of tDir
    set the directory to tDir
    put the detailed files into tFileData
    set the directory to tSavedDir
    set the itemDel to comma
    split tFileData by cr and ","
    return item 1 of tFileData[urlEncode(tFileName)]
  else
    return "no file"
  end if
end ulFileLength
-----------------------------
on ulLogIt pMessage
  local tExp

  if exists(lvLogField) then
    put "put pMessage after fld" && word 2 to -1 of lvLogField into tExp
    do tExp
  end if
end ulLogIt
-----------------------------
on ulStartTickle

  ## safeguard against possible hangs in "wait for messages" loops
  if lvTickle is empty then
    put true into lvTickle
    send "ulTickleMe" to me in 1 seconds
  end if
end ulStartTickle
```

```
--------------------------
on ulTickleMe
  ## safeguard against possible hangs in "wait for messages" loops
  if the openSockets <> empty then
    send "ulTickleMe" to me in 1 seconds
  else
    put empty into lvTickle
  end if
end ulTickleMe
--------------------------
function isIPNumber pHost
  replace "." with empty in pHost
  replace ":" with empty in pHost
  replace "l" with empty in pHost
  return pHost is a number
end isIPNumber
---------------------------
on ulCleanUpHttp x #x is socket
  local tlLoadReq,tConnectHost,tempUrl

  put laLoadReq[laUrl[x]] into tlLoadReq ##holder
  put laConnectHost[laUrl[x]] into tConnectHost
  ulCleanUpHttpLocals laUrl[x] ## remove url referenced locals
  delete local laLoadReq[laUrl[x]] ##OK here??
  delete local laSocketClosedByScript[x]

  -- MM-2014-02-27: [[ HTTPS Proxy ]]
  delete local laSocketSecured[x]

  ##delete socket referenced locals

  put laUrl[x] into tempUrl

  delete local laUrl[x]

  --prepare for next request
  if tlLoadReq then
    delete local laLoadingUrls[tempUrl]
    ## delete line 1 of laLoadQ[tConnectHost] ##commented out for 1.0.8r4 -- now
done in ulNextHttpLoadRequest
    if the number of lines of laLoadQ[tConnectHost] = 0 then
      delete local laLoadQ[tConnectHost] ##important
      delete local laConnectID[tConnectHost]
    end if
  end if
end ulCleanUpHttp
```

```
-----------------------
on ulCleanUpHttpLocals pUrl
  ##crude clean up
  delete local laLength[pUrl]
  delete local laConnectHost[pUrl]
  delete local laAuth[pUrl]
  delete local laUser[pUrl]
  delete local laPasswd[pUrl]
  delete local laHost[pUrl]
  delete local laLongFileName[pUrl]
  delete local laLineNum[pUrl]
  delete local laTmpData[pUrl]
  delete local laTemp[pUrl]
  delete local laAction[pUrl]
  delete local laConn[pUrl]
  delete local laRhHeader[pUrl]
  delete local laNeedChunk[pUrl]
  delete local laStatusCode[pUrl]
  delete local laNewLoc[pUrl]
  delete local laStatusMessage[pUrl]
  delete local laCode[pUrl]
  delete local laChunk[pUrl]
  delete local laHaveHeader[pUrl]
  delete local laHttpDataDone[pUrl]
  delete local laPostData[pUrl]
  delete local laPostLength[pUrl]
  delete local laPostBytes[pUrl]
  delete local laReadBytes[pUrl]
  delete local laCurrentHttpHeaders[pUrl]
  delete local laCurrentSSLVerify[pUrl]

  -- MM-2014-02-27: [[ HTTPS Proxy]]
  delete local laUrlProxy[pURL]
end ulCleanUpHttpLocals
-----------------------------
on ulCleanUpFtp x
  local tlLoadReq,tempUrl,tConnectHost

  put laLoadReq[laUrl[x]] into tlLoadReq ##holder
  put laConnectHost[laUrl[x]] into tConnectHost #holder
  ulCleanUpFtpLocals laUrl[x] ## remove url referenced locals
  delete local laLoadReq[laUrl[x]] ##OK here??

  ##close any data ports ##should be closed already, but if error occurred
  if laTransPasvIP[laUrl[x]] is among the lines of the openSockets then
    close socket laTransPasvIP[laUrl[x]]
```

```
  end if
  if laTransActvIP[x] is among the lines of the openSockets then ##local port
    close socket laTransActvIP[x]
  end if

  delete local  laControlXLocalMap[laTransActvIP[x]]
  delete local  laControlXDataMap[laTransPasvIP[laUrl[x]]]
  delete local laTransPasvIP[laUrl[x]]
  delete local  laTransActvIP[x]
  delete local laSocketClosedByScript[x]

  ##delete socket referenced locals

  put laUrl[x] into tempUrl
  delete local laUrl[x]
  delete local laFtpCommandStatus[x]

  --prepare for next request
  if tlLoadReq then
    delete local laLoadingUrls[tempUrl]
    -- delete line 1 of laLoadQ[tConnectHost] ##commented out for 1.0.8r4 -- now
done in ulNextFtpLoadRequest
    if the number of lines of laLoadQ[tConnectHost] = 0 then
      delete local laLoadQ[tConnectHost] ##important
      delete local laConnectID[tConnectHost]

    end if
  end if
end ulCleanUpFtp
-----------------------------
on ulCleanUpFtpLocals pUrl
  ##clean up locals
  delete local laConnectHost[pUrl]
  delete local laLength[pUrl]
  delete local laAuth[pUrl]
  delete local laUser[pUrl]
  delete local laPasswd[pUrl]
  delete local laHost[pUrl]
  delete local laLongFileName[pUrl]
  delete local laAction[pUrl]
  delete local laHome[pUrl]
  delete local laFtpDataDone[pUrl]
  delete local laMode[pUrl]
  delete local laPostData[pUrl]
  delete local laReadBytes[pUrl]
  delete local laWriteBytes[pUrl]
```

```
  delete local laPostLength[pUrl]
   delete local laPostBytes[pUrl]
end ulCleanUpFtpLocals
----------------------------
on ulCancelRequest pUrl
  local tError

  put true into laCancelled[pUrl]
  put "error cancelled" into tError

  if lvExtDriver is not empty then
    put tError into laUrlErrorStatus[pUrl]
    ulExtCancelRequest pURL
  else
    ulStopRequest pUrl,tError
  end if
end ulCancelRequest
----------------------------
on ulStopRequest pUrl, pErrMessage
  local tSocketKeys,tKey,tItsSocket

  put keys(laUrl) into tSocketKeys##test
  repeat for each line tKey in tSocketKeys
    if laUrl[tKey] = pUrl then
      put tKey into tItsSocket
      exit repeat
    end if
  end repeat
  if tItsSocket is among the lines of the openSockets then
    put false into laStatus[pUrl] ##should cause everything to wind up cleanly
    put pErrMessage into laUrlErrorStatus[pUrl]
    put empty into laData[pUrl]
  end if
  if laLoadReq[pUrl] then
    put "error" into laUrlLoadStatus[pUrl]
    delete local laData[pUrl]
  end if

end ulStopRequest

----------------------------
on ulStoreData pUrl,@pData
  local tErr

  if laFile[pUrl] <> empty then
    write pData to file laFile[pUrl]
```

```
    if the result is not empty then
      put "error" && the result into tErr
      ulStopRequest pUrl,tErr
    end if
  else if laLoadReq[pUrl] <> empty then
    put pData after laLoadedUrls[pUrl]
  else
    put pData after laData[pUrl]
  end if
end ulStoredata
-----------------------------
function ulNextData pUrl
  local tData,tErr

  if laFile[pUrl] is empty then
    put char 1 to 4096 of laPostdata[pUrl] into tData
    delete char 1 to 4096 of laPostData[pUrl]
  else
    read from file laFile[pUrl] for 4096
    if the result <> empty and the result <> "eof" then
      put "error" && the result into tErr
      ulStopRequest pUrl,tErr
    else
      put it into tData
    end if
  end if
  return tData
end ulNextData
-----------------------------
function ulStripUrl pUrl
  ## clean out any whitespace before and after url
  local tString

  put space & tab & cr into tString
  repeat while char 1 of pUrl is in tString
    delete char 1 of pUrl
  end repeat
  repeat while char -1 of pUrl is in tString
    delete char -1 of pUrl
  end repeat
  return pUrl
end ulStripUrl
-------------------------------
function ul_TraceLocals ##DEBUG ROUTINE, used in development
  local e,tRetVal,tDoString,tKeys
```

```
  repeat for each item e in line 3 of the localNames
    put e & cr after tRetVal
    put "put the keys of" && e && "into tKeys" into tDoString
    do tDoString
    if tKeys is empty then
      put "put" && e && "& cr after tRetVal" into tDoString
    else
      put "put ul_PrintKeys(" & e & ") & cr after tRetVal" into tDoString
    end if
    do tDoString
  end repeat
  return tRetVal
end ul_TraceLocals
--------------------------------
function ul_PrintKeys @pArray, pDimension
  local tKeys, tKey, tText

  if pDimension is empty then put 0 into pDimension

  put the keys of pArray into tKeys
  sort tKeys numeric

  repeat for each line tKey in tKeys
    if pArray[tKey] is an array then
      put _printCharXTimes(space, pDimension * 5) & tKey & cr after tText
      put ul_PrintKeys(pArray[tKey], pDimension + 1) after tText
    else
      put _printCharXTimes(space, pDimension * 5) &  tKey & ":" && line 1 of
pArray[tKey] & cr after tText
    end if
  end repeat

  return tText
end ul_PrintKeys
--------------------------------
private function _printCharXTimes pChar, pTimes
  local tStr

  repeat with i = 1 to pTimes
    put pChar after tStr
  end repeat
  return tStr
end _printCharXTimes
--------------------------------
function ulFtpCommand pCommandString, pSocket
  ##executes ftp commands
```

```
  ##returns the response from the server (or ftpErr if error occurs)
  if pSocket is not among the lines of the openSockets then
    return "ftpErr, socket not open"
  end if
  if pCommandString is empty then
    return "ftpErr, no command to send"
  end if
  put empty into laFTPCommandStatus[pSocket]
  if "PASS" is word 1 of pCommandString then
    ulLogit "PASS <password>" & return #LOG
  else
    ulLogit pCommandString & return #LOG
  end if
  write pCommandString & CRLF to socket pSocket
  if the result <> empty then return "ftpErr," & the result
  read from socket pSocket  for 1 line with message "ulGetFtpReply"
  if the result <> empty then return "ftpErr," & the result
  repeat while laFTPCommandStatus[pSocket] is empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat
  return laFTPCommandStatus[pSocket]
end ulFTPCommand
-----------------------------------
function ulFtpWaitResponse pSocket
  ##used for collecting server responses
  ##that are not in response to a direct command
  ##for example when opening a connection to the server, and when transfers
complete
  put empty into laFTPCommandStatus[pSocket]
  read from socket pSocket  for 1 line with message "ulGetFtpReply"
  repeat while laFTPCommandStatus[pSocket] is empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat
  return laFTPCommandStatus[pSocket]
end ulFtpWaitResponse
-----------------------------------
on ulGetFtpReply pSocket,pReply
  ##reads data from the command port
  ##generally called by ulFTPCommand, but also by ulFtpWaitResponse
  local tReply,tReplyNum

  ulLogIt pReply##LOG
  put line -1 of pReply into tReply ##should only be one line
  get char 1 to 3 of tReply
```

```
  if it is an integer and it >= 100 then
    put it into tReplyNum
    if char 4 of tReply <> "-" then
      put tReply into laFTPCommandStatus[pSocket]

    else
      read from socket pSocket for 1 line with message "ulGetFtpReply"
      if the result <> empty then
        put "ftpErr," & the result into laFTPCommandStatus[pSocket]

      end if
    end if
  else
    read from socket pSocket for 1 line with message "ulGetFtpReply"
    if the result <> empty then
      put "ftpErr," & the result into laFTPCommandStatus[pSocket]

    end if
  end if
end ulGetFtpReply
------------------------------
function ulTransferCompleteResponse pSocket
  local tTimerStart, t226Timeout, tCmd

  ## handle 226 responses here
  ## we try to get round a problem for some users
  ## where a 226 response is not received and everything times out
  ## problem may be due to bad routers
  ## we wait for half the socketTimeoutInterval
  ## and if no 226 has been received we "prod" the server with a NOOP command

  put empty into laFTPCommandStatus[pSocket]
  read from socket pSocket  for 1 line with message "ulGetFtpReply"
  put the milliseconds into tTimerStart
  put the socketTimeoutInterval div 2 into t226Timeout

  local tTimerID
  put the result into tTimerID
  repeat while laFTPCommandStatus[pSocket]  is empty AND (the milliseconds -
tTimerStart < t226Timeout)
    if lvJumpOut then exit to top
    wait for messages
  end repeat

  if laFTPCommandStatus[pSocket] <> empty then ##normal situation
    return laFTPCommandStatus[pSocket]
```

```
   else
      put "NOOP"  into tCmd
      return ulFtpCommand(tCmd,pSocket)
   end if

end ulTransferCompleteResponse
---------------------------------


--------------------------------
function ulFtpGoodReply pReply, pCommand
  ##compares a reply code against a predetermined list
  ##of "good" reply codes for a particular command
  local tGoodCodes

  if item 1 of pReply is "ftpErr" then return false
  put the cFtpGoodCodes[word 1 of pCommand] of me into tGoodCodes
  if word 1 of pReply is among the items of tGoodCodes then
    return true
  else
    return false
  end if
end ulFtpGoodReply
-----------------------------------
function libUrlFtpCommand pCommand, pHost, pUser, pPass
   local tHost,tPort,tRegEx,tTempHost,tConnectHost,tDummyUrl,tSocket
   local tLogonReply,tFtpReply

   -- CW-2016-06-11: [[ External driver support ]] Call the external driver
implementation if it is enabled.
   if lvExtDriver is not empty then
      return ulExtFtpCommand(pCommand, pHost, pUser, pPass)
   end if

   put false into lvJumpOut
   if lvCount is empty then
      put "6923" into lvCount
   end if

   ##separate host and port
   put "([^:]*)(.*)" into tRegEx
   if not matchText(pHost,tRegEx,tHost,tPort) then return "error Invalid host address"

   put tHost into tTempHost
   if tPort is empty then put ":21" into tPort

   ##get IP address
```

```
replace "." with empty in tTempHost
replace ":" with empty in tTempHost
if tTempHost  is not a number then
  get hostnameToAddress(tHost)
  if the result is empty then

    put line 1 of it & tPort into tConnectHost
  else
    return "error" && the result
  end if
else
  put tHost & tPort into tConnectHost
end if

##set anonymous user if needed
if pUser is empty then
  put "anonymous" into pUser
  put "guest" into pPass
end if
##make dummy url to use other parts of libUrl
put "ftp:" & pCommand into tDummyUrl
## make laUser and laPasswd entries
put pUser into laUser[tDummyUrl]
put pPass into laPasswd[tDummyUrl]
##check no other ftp activity on this account

local tAccountBusy
put false into tAccountBusy

local tCHKeys
put keys(laConnectHost) into tCHKeys
repeat for each line tCHKey in tCHKeys
  if  laConnectHost[tCHKey] = tConnectHost & "|" & pUser then
    put true into tAccountBusy
    exit repeat
  end if
end repeat
if tAccountBusy then
  return "Error Previous request not completed."

end if
##make laConnectHost entry so we can use ulWhichSocket
##laConnectHost has format host:port|user
put tConnectHost & "|" & pUser into laConnectHost[tDummyUrl]
##do we have an open socket for this user/host combination?
put ulWhichSocket(tDummyUrl) into tSocket
```

```
  put tSocket into lvFtpCommandSocket
  ##don't need any more
  delete local laUser[tDummyUrl]
  delete local laPasswd[tDummyUrl]
  delete local laConnectHost[tDummyUrl]

  put ulFtpLogon(tSocket, pUser,pPass) into tLogonReply
  if tLogonReply is empty then
    put ulFtpCommand(pCommand, tSocket) into tFtpReply
    if laStopUnit[tSocket] = 0 then
      put "1" into laStopUnit[tSocket]
      send "ulFtpStopWatch " & tSocket to me in 50 milliseconds
    end if
    delete local lvFtpCommandSocket
    return tFtpReply
  else
    delete local lvFtpCommandSocket
    return tLogonReply
  end if
end libUrlFtpCommand
---------------------------

function ulFtpLogon pSocket, pUser, pPass
  local tReply,tCmd

  put "0" into laStopUnit[pSocket]
  put "0" into laStopSec[pSocket]
  ulStartTickle ##safeguard routine
  if pSocket is not among the lines of the openSockets then
    get ulOpenSocket(pSocket)
    if not it then return it ##error opening socket
    ----------------------get server response (220)
    put ulFtpWaitResponse(pSocket) into tReply
    if not ulFtpGoodReply(tReply, "connect") then
      return tReply
    end if
    ---------------
    put "USER " & pUser into tCmd
    put ulFtpCommand(tCmd,pSocket) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
      return tReply
    end if
    ----------------------
    put  "PASS " & pPass into tCmd
    put ulFtpCommand(tCmd,pSocket) into tReply
    if not ulFtpGoodReply(tReply, tCmd) then
```

```
      return tReply
    end if
  end if
  return empty
end ulFtpLogon
-----------------------------
##########################################
######socket opening routines#################
##########################################
-------------------------------------------------------------
function ulOpenSocket x
  local tSocketToken

  put empty into lvSocketToken[x]
  put the milliseconds into lvSocketOpenStart[x]
  -- MM-2014-02-27: [[ HTTPS Proxy ]] Only create a secure socket if conneting to
a HTTPS URL directly.
  --  If going though a proxy, first communicate with the proxy unencrypted before
securing the socket later.
  if laUrlProxy[laUrl[x]] is empty and ulIsSecure(laUrl[x]) then
    put true into laSocketSecured[x]
    if laCurrentSSLVerify[laUrl[x]] is false then
      open secure socket to x with message "ulGotSocket" without verification
    else
      open secure socket to x with message "ulGotSocket" with verification
    end if
  else
    put false into laSocketSecured[x]
    open socket to x with message "ulGotSocket"
  end if
  if the result is not empty then
    return the result
  end if
  send "ulSocketTimeout" && x to me in 500 milliseconds
  put the result into lvSocketOpenMessageID[x]

  repeat until lvSocketToken[x] is not empty
    if lvJumpOut then exit to top
    wait for messages
  end repeat
  cancel lvSocketOpenMessageID[x]
  delete local lvSocketOpenStart[x]
  delete local lvSocketOpenMessageID[x]
  put lvSocketToken[x] into tSocketToken ##swap out so we can delete persistent
local
  delete local lvSocketToken[x]
```

```
   if not tSocketToken then
     if x is among the lines of the openSockets then
       close socket x
     end if
   end if
   return tSocketToken
end ulOpenSocket
----------------------------------------------
function ulIsSecure pUrl
  return char 1 to 5 of pUrl is "https"
end ulIsSecure
----------------------------------------------
on ulGotSocket x
  put true into lvSocketToken[x]
end ulGotSocket
----------------------------------------------
on ulSocketTimeout x
  if the milliseconds - lvSocketOpenStart[x] > the socketTimeoutInterval then
    put "timeout" into lvSocketToken[x]
  else
    send "ulSocketTimeout" && x to me in 500 milliseconds
    put the result into lvSocketOpenMessageID[x]
  end if
end ulSocketTimeout
----------------------------------------
function ulFormBoundary
  local tBoundary

  repeat 25
    put any char of "1234567890abcdefghijklmnopqrstuvwxyz" after tBoundary
  end repeat
  return tBoundary
end ulFormBoundary
----------------------------------------
function ulSubForm @pData, ,pFiles, pSubBoundary, pMimeTypes, pEncodings
  ##creates a set of "sub parts" when including multiple files in one part of a
multipart/form-data form
  local tType,tEnc,tCount,tFile,tFilename

  put empty into pData
  if pMimeTypes is empty then
    put "application/octet-stream" into tType
  end if
  if pEncodings is empty then
    put "Binary" into tEnc
  end if
```

```
  put 0 into tCount
  repeat for each item tFile in pFiles
    add 1 to tCount
    if item tCount of pMimeTypes <> empty then
      put item tCount of pMimeTypes into tType
    end if
    if item tCount of pEncodings <> empty then
      put item tCount of pEncodings into tEnc
    end if
    put "--" & pSubBoundary after pData
    set the itemDel to "/"
    put item -1 of tFile into tFilename
    set the itemDel to comma
    put CRLF & "Content-Disposition: attachment; filename=" & quote & tFilename &
quote after pData
    put CRLF & "Content-Type:" &&  tType after pData
    put CRLF & "Content-Transfer-Encoding:" &&  tEnc after pData
    put CRLF & CRLF after pData
    put url ("binfile:" & tFile) after pData
    if the result <> empty then return "error" && the result
    put CRLF after pData
  end repeat
  put "--" & pSubBoundary & "--" after pData
  return empty
end ulSubForm
-----------------------------------------

-- MW-2013-07-01: [[ Bug 10985 ]] Returns the caller of the caller.
private function ulGetCaller
  get item 1 to -3 of line -3 of the executionContexts
  if there is not an it then
    delete item -1 of it
  end if
  return it
end ulGetCaller

-- MM-2014-02-27: [[ PAC Support ]]
private function _extractHost pURL
  ## http://bob:jones@www.screensteps.com

  local tHost
  put pURL into tHost
  set the itemDelimiter to slash

  local tCharNo
  put offset("://", tHost) into tCharNo
```

```
   if tCharNo is 0 then
      put item 1 of pURL into tHost
   else
      put item 3 of pURL into tHost
   end if

   ## strip username/password
   put offset("@", tHost) into tCharNo
   if tCharNo > 0 then delete char 1 to tCharNo of tHost

   ## clip port
   put offset(":", tHost) into tCharNo
   if tCharNo > 0 then delete char tCharNo to -1 of tHost
   return tHost
end _extractHost

----------------------------------------------

private command libURLInitializeProxy
   if the HTTPProxy is not empty or lvProxyInitialized then
      return empty
   end if
   put true into lvProxyInitialized

   local tError
   put empty into tError

   if tError is empty then
      _proxyConfig_SetHTTPProxy
      put the result into tError
   end if

   if tError is empty then
      _proxyConfig_ConfigureBypassList
      put the result into tError
   end if

   return tError
end libURLInitializeProxy

-- Attempt to autodetect any proxy servers the system has configured.
-- First of all try WPAD and PAC.
-- If that fails, extract any system proxy settings and if found, set proxy manually.
--
-- Returns any errors or empty on success.
--
```

```
private command _proxyConfig_SetHTTPProxy
  local tResult
  _proxyConfig_SetHTTPProxyUsingWPADAndPAC
  put the result into tResult

  if tResult is not empty then
    _proxyConfig_SetHTTPProxyManually
    put the result into tResult
  end if

  return tResult
end _proxyConfig_SetHTTPProxy

-- Add any URLs that the OS states bypass proxy settings to the bypass list.
--
-- Returns any errors or empty on success.
--
private command _proxyConfig_ConfigureBypassList
  local tBypassList
  switch the platform

    case "MacOS"
      -- On Mac, we parse the system prefs using scutil.
      --
      local tExceptionList
      put shell("scutil --proxy") into tExceptionList
      if the result is empty then

        local tStartLine, tEndLine
        set the itemDelimiter to ":"
        put lineOffset("ExceptionsList", tExceptionList) into tStartLine
        if tStartLine > 0 then
          put lineOffset("}", tExceptionList, tStartLine) into tEndLine
          if tEndLine > 0 then
            add tStartLine to tEndLine
            put line tStartLine + 1 to tEndLine -1 of tExceptionList into tExceptionList
            repeat for each line tURL in tExceptionList
              put word 1 to -1 of tURL & cr after tBypassList
            end repeat
            delete the last char of tBypassList
            libURLSetProxyBypassList tBypassList
          end if
        end if
      end if
      break
```

```
    case "Win32"
      -- On Windows, we check the registry.
      --
      local tBoolean
      get binaryDecode("I*",
queryRegistry("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyEnable"), tBoolean)
      if tBoolean is 1 then
        put
queryRegistry("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyOverride") into tBypassList
        replace ";" with cr in tBypassList
        libURLSetProxyBypassList tBypassList
      else if tBoolean is empty then
        get binaryDecode("I*",
queryRegistry("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyEnable"), tBoolean)
        if tBoolean is 1 then
          put
queryRegistry("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyOverride") into tBypassList
          replace ";" with cr in tBypassList
          libURLSetProxyBypassList tBypassList
        end if
      end if
      break

    case "Linux"
      break

  end switch

  return empty
end _proxyConfig_ConfigureBypassList

-- Extracts the systems PAC (Proxy Auto-Configuration) file using WPAD.
-- If found, libURL is then configured to use the PAC file for determnining the
proxy server to use for a URL.
--
-- Returns empty if successful.
--
private command _proxyConfig_SetHTTPProxyUsingWPADAndPAC
  local tError

  if tError is empty then
    if not _proxyConfig_SystemUsesWPAD() then
```

```
      put "System does not support WPAD" into tError
    end if
  end if

  local tPacURL
  if tError is empty then
    put _proxyConfig_LocatePACFileUsingWPAD() into tPacURL
    if tPacURL is empty then
      put "No PAC file configured on system" into tError
    end if
  end if

  if tError is empty then
    ulLogIt "pac file that wpad found: " & tPacURL
    _proxyConfig_ProcessPACURL tPacURL
    put the result into tError
  end if

  if tError is empty then
    ulLogIt "using wpad PAC file"
    put true into lvUsePACFileForProxy
  else
    ulLogIt "wpad pac error:" && tError
  end if

  return tError
end _proxyConfig_SetHTTPProxyUsingWPADAndPAC

-- Check to see if the OS supports WPAD (Web Proxy Auto-Discovery Protocol).
-- Only supported on Windows currently.
--
-- MM-2014-08-13: Updated to add OS X support as per bug 13172.
--
-- Returns true or false.
--
private function _proxyConfig_SystemUsesWPAD
  local tUseWPADDetection
  put "false" into tUseWPADDetection

  switch the platform

    case "MacOS"
      local tSystemProxyConfig, tLineNo, tError
      put shell("scutil --proxy") into tSystemProxyConfig
      put the result into tError
```

```
        if tError is empty then
          put lineOffset("ProxyAutoConfigEnable", tSystemProxyConfig) into tLineNo
          if tLineNo > 0 then
            put lineOffset("ProxyAutoConfigURLString", tSystemProxyConfig) into tLineNo
            if tLineNo > 0 then
              local tAutoConfigURL
              set the itemDelimiter to ":"
              put item 2 to -1 of line tLineNo of tSystemProxyConfig into tAutoConfigURL
              put word 1 to -1 of tAutoConfigURL into tAutoConfigURL
              put tAutoConfigURL is "http://wpad/wpad.dat" into tUseWPADDetection
              set the itemDel to comma
            end if
          end if
        end if
        break

      case "Win32"
        local tType, tBinary
        put
queryRegistry("HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersi
on\Internet Settings\Connections\DefaultConnectionSettings", tType) into tBinary
        if tBinary is empty then
          put
queryRegistry("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersi
on\Internet Settings\Connections\DefaultConnectionSettings", tType) into tBinary
        end if

        ulLogit "registry type for DefaultConnectionSettings is " & tType

        local tResult, tValue
        if tType is "binary" then
          put binaryDecode("c*", char 9 of tBinary, tValue) into tResult
          put (tValue bitand 8) is 8 into tUseWPADDetection
          ulLogit "theValue bitand 8:" && (tValue bitand 8)
          ulLogit "useWPADDetection:" && tUseWPADDetection
        end if
        break

      case "Linux"
        put false into tUseWPADDetection
        break

    end switch

    return tUseWPADDetection
end _proxyConfig_SystemUsesWPAD
```

```
-- Determine the location of the systems PAC file using WPAD.
--
-- Returns URL of PAC file or empty if none found.
--
private function _proxyConfig_LocatePACFileUsingWPAD
  local tSocketTimeOut, tHostName
  set the wholeMatches to true
  put the socketTimeoutInterval into tSocketTimeOut
  set the socketTimeoutInterval to 3000

  -- Fist of all determine the host name of the current system.
  --
  set the itemDelimiter to "."
  put the hostName into tHostName
  switch the platform

    case "Win32"
      -- Apend the systems domain to the host name, first of all trying next.exe
then the registry.
      --
      local tDomain
      put _proxyConfig_ExtractDomainFromNetExe() into tDomain
      if tDomain is empty or the number of items of tDomain < 2 then
        put
queryRegistry("HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\P
arameters\Domain") into tDomain
      end if

      if the number of items of tDomain > 1 then
        put "." & tDomain after tHostName
      end if
      break

    case "MacOS"
      break

    case "Linux"
      break

  end switch

  ulLogIt "WPAD initial host: " & tHostName

  -- Trim items of the host name until we find a valid IP address for host.
  --
```

```
local i
local tPacURL, tPerformedPreliminaryNetworkTest
put false into tPerformedPreliminaryNetworkTest
repeat forever
  add 1 to i

  -- Do the trimming, stripping out any IP address.
  --
  get item 1 to 4 of tHostName
  replace "." with empty in it
  if it is an integer then
    delete item 1 to 4 of tHostName
  else
    delete item 1 of tHostName
  end if

  ulLogIt "WPAD host (" & i & "): " & tHostName

  if the number of items of tHostName is 0 then
    exit repeat
  end if

  -- Only check network connectivity once. If none, then exit.
  --
  if not tPerformedPreliminaryNetworkTest then
    get hostNameToAddress(tHostName)
    if the result is not empty then
      exit repeat
    end if
    put true into tPerformedPreliminaryNetworkTest
  end if

  -- Filter out known suffixes.
  --
  if item 1 of tHostName is among the lines of the cTopLevelDomains of me then
    exit repeat
  end if

  -- Check to see if this is a valid host name. If so, then build PAC URL from it.
  --
  local tWPADHostName, tIPAddress
  put "wpad." & tHostName into tWPADHostName
  ulLogIt "WPAD host name (" & i & "): " & tWPADHostName
  put hostNameToAddress(tWPADHostName) into tIPAddress
  if tIPAddress is not empty then
    put line 1 of tIPAddress into tIPAddress
```

```
      put format("http://%s:80/wpad.dat", tIPAddress) into tPacURL
      ulLogIt "WPAD url (" & i & "): " & tPacURL
      exit repeat
    end if
  end repeat

  set the socketTimeoutInterval to tSocketTimeOut
  return tPacURL
end _proxyConfig_LocatePACFileUsingWPAD

-- Process the given PAC URL.
--
-- Returns any errors or empty on success.
--
private command _proxyConfig_ProcessPACURL pURL
  local tError
  put empty into tError

  -- If the file is on the local system, the tidy up the URL.
  --
  if pURL begins with "file:" then
    if pURL begins with "file://" then
      put "file:" into char 1 to 7 of pURL
    end if
    replace "%20" with space in pURL
    get offset("?", pURL)
    if it > 0 then
      delete char it to -1 of pURL
    end if
    ulLogIt "modified 'file:' proxy url:" && pURL
  end if

  local tHTTPProxy
  put the HTTPProxy into tHTTPProxy
  set the HTTPProxy to empty

  -- Extract the contents of the PAC file.
  --
  local tPacFile
  put URL pURL into tPacFile
  put the result into tError
  ulLogIt "fetch proxy url result:" && tError

  if tError is empty then
    replace numToChar(13) & numToChar(10) with numToChar(10) in tPacFile
    replace numToChar(13) with numToChar(10) in tPacFile
```

```
    -- Microsoft ins file. Grab the line for AutoConfigJSURL= to determine the
actual location of the PAC file.
    --
    if char -4 to -1 of pURL is ".ins" then
      local tLineNo
      put lineOffset("AutoConfigJSURL=", tPacFile) into tLineNo
      if tLineNo > 0 then
        local tURL
        put line tLineNo of tPacFile into tURL
        set the itemDelimiter to "="
        put item 2 to -1 of tURL into tURL
        set the itemDelimiter to ","

        put URL tURL into tPacFile
        put the result into tError
        if the result is empty then
          replace numToChar(13) & numToChar(10) with numToChar(10) in tPacFile
          replace numToChar(13) with numToChar(10) in tPacFile
        else
          put format("error retrieving PAC file in INS file %s (%s)", pURL, tError) into
tError
        end if
      else
        put format("unable to extract AutoConfigJSURL from ins file %s", pURL) into
tError
      end if
    end if
  end if

  -- Decode, clean up and initialise the PAC file
  --
  if tError is empty then
    put uniDecode(uniEncode(tPacFile), "UTF8") into tPacFile
    put word 1 to -1 of tPacFile into tPacFile
    if the last char of tPacFile is "=" then
      delete the last char of tPacFile
    end if

    _proxyConfig_InitializePacFile tPacFile
    put the result into tError
  end if

  set the HTTPProxy to tHTTPProxy
  return tError
end _proxyConfig_ProcessPACURL
```

```
-- Initializes httpproxyForURL with the PAC support javascript custom property
and the PAC file data provided.
-- After calling this handler you can call httpproxyforurl with just two parametrs.
--
-- Returns any errors or empty on success.
--
private command _proxyConfig_InitializePacFile pPacFile, pURL, pHost
  ulLogIt "initializing pac file:" & cr & "pURL:" && pURL & cr & "pHost:" && pHost & cr &
pPacFile
  if pURL is empty then
    put "http://127.0.0.1" into pURL
    put "127.0.0.1" into pHost
  end if

  local tProxy
  put httpproxyforurl(pURL, pHost, the cPACSupport of me & cr & pPacFile) into tProxy

  local tError
  if tProxy is empty then
    put "unable to parse PAC script" into tError
  end if
  ulLogIt "pac file initialization error:" && tError

  return tError
end _proxyConfig_InitializePacFile

-- Windows helper function for determining the domain of the computer this
function is running on.
--
private function _proxyConfig_ExtractDomainFromNetExe
  local tResult, tError
  try
    -- MM-2014-06-05: [[ Bug 12470 ]] Make sure the hide console windows is set
to false to prevent popup when accessing net.exe.
    local tOldHideConsoleWindows
    put the hideConsoleWindows into tOldHideConsoleWindows
    set the hideConsoleWindows to true

    put shell("net.exe config workstation") into tResult
    if the result is not 0 then
      put tResult into tError
    end if
    set the hideConsoleWindows to tOldHideConsoleWindows
  catch tError
  end try
```

```
  local tDomain
  if tError is empty then
    local tKey, tLineNo
    put "Workstation Domain DNS Name" into tKey
    put lineOffset(tKey, tResult) into tLineNo
    if tLineNo > 0 then
      local tLine
      put line tLineNo of tResult into tLine
      delete char 1 to length(tKey) of tLine
      put word 1 to -1 of tLine into tDomain

      if tDomain is "(null)" then
        put empty into tDomain
      end if
    end if
  end if

  return tDomain
end _proxyConfig_ExtractDomainFromNetExe

-- Attempt to extract any system proxy setting manually, rather than using WPAD.
--
-- Returns empty on success or any errors otherwise.
--
private command _proxyConfig_SetHTTPProxyManually
  local tError
  put empty into tError

  local tAutoConfigURL, tProxy, tProxyFound
  put false into tProxyFound

  switch the platform

    case "MacOS"

      -- Extract the system proxy settings using scutil
      --
      local tSystemProxyConfig, tLineNo
      put shell("scutil --proxy") into tSystemProxyConfig
      put the result into tError

      if tError is empty then
        -- First of all check to see it we have a PAC file set. If so, attempt to use.
        --
        set the itemDelimiter to ":"
```

```
        put lineOffset("ProxyAutoConfigEnable", tSystemProxyConfig) into tLineNo
        if tLineNo > 0 then
          put lineOffset("ProxyAutoConfigURLString", tSystemProxyConfig) into tLineNo
          if tLineNo > 0 then
            put item 2 to -1 of line tLineNo of tSystemProxyConfig into tAutoConfigURL
            put word 1 to -1 of tAutoConfigURL into tAutoConfigURL

            -- "http://wpad/wpad.dat" is for auto discovery. That is done elsewhere.
            --
            if tAutoConfigURL is not empty and tAutoConfigURL is not "http://wpad/
wpad.dat" then
              local tPacURLError
              _proxyConfig_ProcessPACURL tAutoConfigURL
              put the result into tPacURLError
              put (tPacURLError is empty) into lvUsePACFileForProxy
              put (tPacURLError is empty) into tProxyFound
            end if
          end if
        end if

        -- If no PAC file has been set, then check to see if a proxy server has been
set directly.
        --
        if not tProxyFound then
          put lineOffset("HTTPEnable", tSystemProxyConfig) into tLineNo
          if tLineNo > 0 then
            local tIsEnabled
            put item 2 to -1 of line tLineNo of tSystemProxyConfig into tIsEnabled
            put word 1 to -1 of tIsEnabled into tIsEnabled
            if tIsEnabled is 1 then
              put lineOffset("HTTPProxy", tSystemProxyConfig) into tLineNo
              if tLineNo > 0 then
                put true into tProxyFound
                put item 2 to -1 of line tLineNo of tSystemProxyConfig into tProxy
                put word 1 to -1 of tProxy into tProxy
                put lineOffset("HTTPPort", tSystemProxyConfig) into tLineNo
                local tPort
                if tLineNo > 0 then
                  put item 2 to -1 of line tLineNo of tSystemProxyConfig into tPort
                  put word 1 to -1 of tPort into tPort
                  if tPort is not empty then
                    put ":" & tPort after tProxy
                  end if
                end if
              end if
            end if
```

```
            end if
          end if

        end if
        break

      case "Win32"

        -- Fist of all check the resgistry to see if a PAC URL has been set.
        --
        put
queryRegistry("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\AutoConfigURL") into tAutoConfigURL
        if tAutoConfigURL is empty then
          put
queryRegistry("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\AutoConfigURL") into tAutoConfigURL
        end if
        ulLogIt "auto config url:" && tAutoConfigURL
        if tAutoConfigURL is not empty then
          _proxyConfig_ProcessPACURL tAutoConfigURL
          put the result into tPacURLError
          ulLogIt "manual lookup pac error:" && tPacURLError
          put (tPacURLError is empty) into lvUsePACFileForProxy
          put (tPacURLError is empty) into tProxyFound
        end if

        -- If PAC file set up failed then check to see if a proxy server has been set
directly in the resistry.
        --
        if not tProxyFound then
          local tBoolean
          get binaryDecode("I*",
queryRegistry("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyEnable"), tBoolean)
          if tBoolean is 1 then
            put true into tProxyFound
            put
queryRegistry("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyServer") into tProxy
          else if tBoolean is empty then
            get binaryDecode("I*",
queryRegistry("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyEnable"), tBoolean)
            if tBoolean is 1 then
              put true into tProxyFound
```

```
            put
queryRegistry("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\I
nternet Settings\ProxyServer") into tProxy
          end if
        end if

        if tProxy contains "=" then
          split tProxy by ";" AND "="
          if tProxy["http"] is not empty then
            put tProxy["http"] into tProxy
          end if
        end if

        ulLogIt "proxy in registry:" && tProxy
      end if
      break

    case "Linux"
      break

  end switch

  if not tProxyFound then
    if tError is empty then
      put tPacURLError into tError
    end if
  else if tProxy is not empty then
    put tProxy into lvHTTPProxy
  end if

  return tError
end _proxyConfig_SetHTTPProxyManually

-----------------------------------------------

private command libURLSetProxyBypassList pList
  replace ";" with LF in pList
  put pList into lvProxyBypassList
end libURLSetProxyBypassList

function libURLURLBypassesProxy pURL
  if lvProxyBypassList is empty then
    return false
  end if

  local theCharNo,theHost,theLine,theList
```

```
   local theRegEx,urlIsBypassed
   put _extractHost(pURL) into theHost
   put lvProxyBypassList into theList
   put false into urlIsBypassed
   repeat for each line theLine in theList
      ## Look for wilcard matches
      if theLine is "<local>" then ## bypass proxy for local addresses
         put theHost is "localhost" or theHost is "127.0.0.1" into urlIsBypassed
      else if theLine contains "*" then
         put theLine into theRegEx
         replace "." with "\." in theRegEx ## escape any "."
         replace "*" with ".+?" in theRegEx
         put matchText(theHost, theRegEx) into urlIsBypassed
      else
         put theLine is theHost into urlIsBypassed
      end if
      if urlIsBypassed then
         exit repeat
      end if
   end repeat
   return urlIsBypassed
end libURLURLBypassesProxy

----------------------------------------------

-- Returns the proxy server to use for the given URL.
--
private function _proxyForURL pURL
   ulLogIt "_proxyForURL"

   libURLInitializeProxy

   -- If this URL is on the bypass list then ignore.
   --
   if libURLURLBypassesProxy(pURL) then
      return empty
   end if

   -- If the developer has set the HTTPProxy then use this value.
   --
   if the HTTPProxy is not empty then
      ulLogIt "_proxyForURL use _proxyFromHTTPProxy() with HTTPProxy"
      return _proxyfromHTTPProxy(pURL, the HTTPProxy)
   end if
```

```
  -- If the system has been configured with PAC file, the extract proxy using this
method.
  --
  if lvUsePACFileForProxy then
    ulLogIt "_proxyForURL use _proxyFromPacOfURL()"
    return _proxyFromPacOfURL(pURL)
  end if

  -- If the system has a proxy server configured directly, then use.
  --
  if lvHTTPProxy is not empty then
    ulLogIt "_proxyForURL use _proxyFromHTTPProxy()"
    return _proxyfromHTTPProxy(pURL, lvHTTPProxy)
  end if

  ulLogIt "_proxyForURL end with none found"
  return empty
end _proxyForURL

private function _proxyfromPACofURL pURL
  local tProxyHost
  get httpProxyForURL(pURL, _extractHost(pURL))
  set the itemDelimiter to ";"
  get item 1 of it
  put word 2 of it into tProxyHost

  if char 1 to 4 of tProxyHost is "ftp:" then
    get item 2 of it
    put word 2 of it into tProxyHost
    if char 1 to 4 of tProxyHost is "ftp:" then
      get empty
    end if
  end if

  switch word 1 of it
    case empty
    case "DIRECT"
      return empty
    case "PROXY"
      return "http://" & tProxyHost
    default
      return (word 1 of it) & "://" & tProxyHost
  end switch
end _proxyFromPACofURL

private function _proxyfromHTTPProxy pURL, pProxy
```

```
  if pProxy is empty then
    return empty
  end if

  if not (pProxy contains "://") then
    put "http://" before pProxy
  end if

  local tCharNo
  put offset("://", pProxy) into tCharNo
  put offset(":", pProxy, tCharNo + 3) into tCharNo
  if tCharNo is 0 then
    put ":80" after pProxy
  end if
  return pProxy
end _proxyFromHTTPProxy

----------------------------------------------------
-- CW-2016-06-11: [[ External driver support ]] Add support for using an external library for network functions.

-- Mapping commands for externals to set appropriate local variables as needed by libUrl
command ulExtSetLoadStatus pUrl, pStatus
  put pStatus into laUrlLoadStatus[pUrl]
end ulExtSetLoadStatus

command ulExtSetErrorStatus pUrl, pStatus
  put pStatus into laUrlErrorStatus[pUrl]
end ulExtSetErrorStatus

command ulExtSetAsLoaded pUrl pData
  put pData into laLoadedUrls[pUrl]
end ulExtSetAsLoaded

command ulExtRemoveLoadingVars pUrl
  delete local laLoadingUrls[pUrl]
  delete local laLoadReq[pUrl]
end ulExtRemoveLoadingVars

command ulExtSetData pUrl pData
  put pData into laData[pUrl]
end ulExtSetData

function ulExtIsLoadReq pUrl
  return laLoadReq[pUrl]
```

```
end ulExtIsLoadReq

function ulExtIsFileTransfer pUrl
  return laFile[pUrl]
end ulExtIsFileTransfer

command ulExtSetLastHTTPHeaders pHeaders
  set the lastHTTPHeaders of me to pHeaders
end ulExtSetLastHTTPHeaders

command ulExtSetLastRHHeaders pHeaders
  set the lastRhHeaders of me to pHeaders
end ulExtSetLastRHHeaders

function ulExtGetMaxPostWithoutExpect
  return laMaxPostWithoutExpect
end ulExtGetMaxPostWithoutExpect
```