LPQMgr.RC

```
/*****************************************************************************
lpqmgr.rc
produced by VisualAge Resource Workshop
*****************************************************************************/

#define DIALOG_3  3
#define IDDELETE  13
#define IDC_JOBLIST  101
#define IDC_AGENT 102
#define IDDELJOBS 15
#define DIALOG_2  2
#define DIALOG_1  1
#define IDC_JOBS  12
#define IDC_PRINTER  11
#define IDC_SERVER   10
#define IDREFRESH 14
#define IDEXIT 1
DIALOG_1 DIALOG 26, 20, 288, 200
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CAPTION "Line Print Queue Manager"
FONT 8, "System"
{
 DEFPUSHBUTTON "Refresh", IDREFRESH, 5, 180, 50, 15
 PUSHBUTTON "Exit", IDEXIT, 60, 180, 50, 15
 LTEXT "Server Alias:", -1, 5, 5, 55, 8
 LTEXT "Printer Queue:", -1, 5, 20, 55, 8
 COMBOBOX IDC_SERVER, 65, 5, 170, 50, CBS_DROPDOWN | CBS_SORT | WS_TABSTOP
 EDITTEXT IDC_PRINTER, 65, 20, 170, 10, WS_BORDER | WS_TABSTOP
 LTEXT "Jobs:", -1, 5, 35, 30, 8
 LISTBOX IDC_JOBS, 5, 50, 275, 125, WS_BORDER | LBS_MULTIPLESEL | LBS_USETABSTOPS | WS_BORDER | WS_VSCROLL
| WS_HSCROLL
 PUSHBUTTON "Delete", IDDELETE, 115, 180, 50, 15
}

DIALOG_2 DIALOG 80, 80, 114, 21
STYLE WS_CHILD | WS_VISIBLE
FONT 8, "System"
{
 LTEXT "Accessing queue. Please wait...", -1, 5, 5, 110, 12
}

DIALOG_3 DIALOG 6, 15, 207, 74
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Delete Print Jobs"
FONT 8, "System"
{
 DEFPUSHBUTTON "Delete", IDDELJOBS, 148, 6, 50, 14
 PUSHBUTTON "Cancel", IDCANCEL, 148, 24, 50, 14
 LTEXT "Jobs:", -1, 5, 8, 30, 10
 EDITTEXT IDC_JOBLIST, 5, 20, 135, 10, WS_BORDER | WS_TABSTOP
 LTEXT "User Agent:", -1, 5, 40, 45, 10
 EDITTEXT IDC_AGENT, 5, 55, 135, 10, WS_BORDER | WS_TABSTOP
}
```

LPQMgr.REX

```
/*****************************************************************************/
/* Name: LPQMgr.Rex                                                          */
/* Type: Object REXX Script using OODialog                                   */
/* Author: Christian Michel                                                  */
/* Resource: Lpqmgr.rc                                                       */
/*                                                                           */
/* Description:                                                              */
/* This file has been created by the Object  REXX Workbench OODIALOG         */
/* template generator.                                                       */
/*                                                                           */
/* Copyright (C) IBM Corporation 1999. All Rights Reserved.                  */
/*                                                                           */
/*****************************************************************************/

/* Load REXX Socket library if not already loaded       */
If RxFuncQuery("SockLoadFuncs") Then
 Do
   Call RxFuncAdd "SockLoadFuncs","RXSOCK","SockLoadFuncs"
   Call SockLoadFuncs
 End
```

```
/* profile and .rc will be read from same path as program  LPQMGR.REX */
Parse Source . . LPQMgrFile
LPQMgrPath = SubStr(LPQMgrFile, 1, LastPos("\", LPQMgrFile))
.environment["LPQMGR_PATH"] = LPQMgrPath /* remember path for all */

Call ReadProfile

LPQMgr = .LPQMgrClass~new
if LPQMgr~InitCode = 0 then do
  rc = LPQMgr~Execute("SHOWTOP")
end

/* Add program code here */

exit   /* leave program */

::requires "OODIALOG.CLS"    /* This file contains the OODIALOG classes */

/* --------------------------- Directives ---------------------------------*/

::class LPQMgrClass subclass UserDialog

::method Init
  use arg InitStem.
  if Arg(1,"o") = 1 then
     InitRet = self~Init:super
  else
     InitRet = self~Init:super(InitStem.)   /* Initialization stem is used */

  LPQMgrPath = .environment["LPQMGR_PATH"]
  if self~Load(LPQMgrPath || "LPQMgr.rc", 1) \= 0 then do
     self~InitCode = 1
     return
  end

  /* Connect dialog control items to class methods  */
  self~ConnectButton("IDREFRESH","IDREFRESH")
  self~ConnectButton("IDEXIT","IDEXIT")
  self~ConnectButton("IDDELETE","IDDELETE")
  self~ConnectControl("IDC_SERVER","ServerSelected")

  /* Initial values that are assigned to the object attributes  */
  self~IDC_PRINTER= ''                              /* Entry Line */
  self~IDC_JOBS= ''                                 /* List Box */

  /* Add your initialization code here */
  return InitRet

::method InitDialog
  ListFont = self~CreateFont("Courier", 10)
  self~SetItemFont("IDC_JOBS", ListFont)
  self~SetListWidth("IDC_JOBS", 500)

  DefaultServer = ""
  DefaultPrinter = ""

  Do Entry Over .LPRServerData~List
    self~AddComboEntry("IDC_SERVER", .LPRServerData~List[Entry]~Server)
    If .LPRServerData~List[Entry]~Default = 1 Then
     Do
       DefaultServer = .LPRServerData~List[Entry]~Server
       DefaultPrinter = .LPRServerData~List[Entry]~Printer
     End
  End

  If DefaultServer \= "" Then
   Do
     self~SetComboLine("IDC_SERVER", DefaultServer)
     self~SetValue("IDC_PRINTER", DefaultPrinter)
     self~IDREFRESH
   End

/* --------------------- message handler ----------------------------------*/

  /* Method IDREFRESH is connected to item "IDREFRESH" */
::method IDREFRESH
  self~GetData
  LPRServer = .LPR~New(self~IDC_SERVER~Word(1), self~IDC_PRINTER)
  Reply = LPRServer~QueryQueue
  self~ToTheTop
  self~ListDrop("IDC_JOBS")
  Do Item Over Reply
    self~AddListEntry("IDC_JOBS", Item)
  End
```

```
  /* Method IDEXIT is connected to item "IDEXIT" */
::method IDEXIT

  /* Method IDDELETE is connected to item "IDDELETE" */
::method IDDELETE
  self~GetData
  SelectedLines = self~GetMultiList("IDC_JOBS")
  JobList = ""
  Do i=1 To SelectedLines~Words
    Details = self~GetListEntry("IDC_JOBS", SelectedLines~Word(i))
    /* see if details contain a job identifier */
    Do j = 1 To Details~Words
      If Details~Word(j)~DataType("N") = 1 Then
       Do
         JobList = JobList Details~Word(j)
         Leave /* this line is done */
       End
    End
  End

  /* get rid of leading and trailing spaces */
  JobList = JobList~Strip("B")

  /* get user agent information from LPRServerData if available */
  Server = self~IDC_SERVER
  if .LPRServerData~List~HasIndex(Server) then do
     Entry = .LPRServerData~List[Server]
     UserAgent = Entry~Agent
  end

  If Length(JobList) > 0 Then
   Do
    JobDelDlg = .LPQDelJobDlg~new(Server~Word(1), self~IDC_PRINTER, JobList, UserAgent)
    if JobDelDlg~InitCode = 0 then do
      rc = JobDelDlg~Execute("SHOWTOP")
      if rc = 1 then /* if job was deleted refresh the display */
        self~IDREFRESH
    end
    self~ToTheTop
   End

  /* Method ServerSelected is connected to item "IDC_SERVER" */
::method ServerSelected
  use arg MsgParm1, MsgParm2

  if MsgParm1 % x2d("10000") = 9 then do /* CBN_SELENDOK */
     /* a new server has been selected from the combo box */
     NewServer = self~GetValue("IDC_SERVER")

     if .LPRServerData~List~HasIndex(NewServer) then do
        Entry = .LPRServerData~List[NewServer]
        self~SetValue("IDC_PRINTER", Entry~Printer)
     end

     self~IDREFRESH
  end

/* ------------------------ Status Message Window --------------------------*/

::class LPQMgrWait subclass UserDialog
::method Init
  InitRet = self~Init:super

  LPQMgrPath = .environment["LPQMGR_PATH"]
  if self~Load(LPQMgrPath || "LPQMgr.rc", "DIALOG_2") \= 0 then do
     self~InitCode = 1
     return
  end

::method Close Unguarded
  self~finished = 1

/* ------------------------ Job Deletion Window ----------------------------*/

::class LPQDelJobDlg subclass UserDialog

::method Init
  Expose LPServer LPPrinter UserAgent
  Use Arg LPServer, LPPrinter, JobList, UserAgent
  InitRet = self~Init:super

  LPQMgrPath = .environment["LPQMGR_PATH"]
  if self~Load(LPQMgrPath || "LPQMgr.rc", "DIALOG_3") \= 0 then do
```

```
      self~InitCode = 1
      return
    end

    /* Connect dialog control items to class methods  */
    self~ConnectButton("IDDELJOBS","IDDELJOBS")
    self~ConnectButton(2,"Cancel")

    /* Initial values that are assigned to the object attributes  */
    self~IDC_JOBLIST= JobList                          /* Entry Line */
    self~IDC_AGENT= UserAgent                          /* Entry Line */

    /* Add your initialization code here */
    return InitRet

/* --------------------- message handler ----------------------------------*/

    /* Method IDDELJOBS is connected to item "IDDELJOBS" */
::method IDDELJOBS
    Expose LPServer LPPrinter
    self~GetData
    LPRServer = .LPR~New(LPServer, LPPrinter)
    Reply = LPRServer~CancelJobs(self~IDC_AGENT,  self~IDC_JOBLIST)
    self~Finished = self~OK:super
    return self~Finished


    /* Method Cancel is connected to item 2  */
::method Cancel
    resCancel = self~Cancel:super  /* make sure self~InitCode is set to 2 */
    self~Finished = resCancel      /* 1 means close dialog, 0 means keep open  */
    return resCancel

/* --------------------- LPR command class --------------------------------*/

::CLASS LPR
::METHOD Init
    Expose Server Queue
    Use Arg Server, Queue

::METHOD SendCommand
    Expose Server Queue

    /* Assemble command string */
    CmdStr = Arg(1)~D2C || Queue
    If Arg(2,"E") = 1 Then
      CmdStr = CmdStr Arg(2)

    WaitMsg = .LPQMgrWait~New
    if WaitMsg~InitCode = 0 then
      WaitMsg~ExecuteAsync(100,"SHOWTOP")

    Socket = .LPDSocket~New(Server)
    Response = Socket~SendCommand(CmdStr)
    Socket~Close

    WaitMsg~Close
    WaitMsg~EndAsyncExecution

    /* In case the reply contains CR we replace them with */
    /* spaces (e.g. for OS/2 daemons).                    */
    Response = Response~Translate(" ", "0D"x)
    Return Response

::METHOD QueryQueue
    Response = Self~SendCommand(3)

    /* Now split the lines into single entries in the  */
    /* result List. */
    Reply = .List~New
    Do While Length(Response) > 0
      Parse Var Response Line "0A"x Response
      Reply~Insert(Line)
    End

    Return Reply

::METHOD CancelJobs
    /* Cancel takes the agent and a job list as parameters  */
    Use Arg UserAgent, JobList

    Parms = UserAgent JobList
    Response = Self~SendCommand(5, Parms)
```

```
  Return Response

/* --------------------- LPD socket handling class -------------------------*/

::CLASS LPDSocket
/*******************************************************/
/*                                                     */
/* Method:    Init                                     */
/* Purpose:   Create a socket and connect it to server.*/
/* Arguments: Server - server name                     */
/* Returns:   0 if successful, -1 otherwise            */
/*                                                     */
/*******************************************************/
::METHOD Init
  Expose Socket
  Use Arg Server

  /* resolve server name alias to dotted IP address     */
  rc = SockGetHostByName(Server, "Host.!")
  If rc = 0 Then
   Do
     Say "Unable to resolve server:" Server
     Return -1
   End

  /* create a TCP socket                                */
  Socket = SockSocket("AF_INET", "SOCK_STREAM", "0")
  If Socket < 0 Then
   Do
     Say "Unable to create socket"
     Return -1
   End

  /* bind socket to local ports 721-731 (req'd by LPD!) */
  Local.!family = "AF_INET"
  Local.!addr = SockGetHostId()
  Local.!port = 721
  Do While SockBind(Socket, "Local.!") = -1
    Local.!port = Local.!port + 1
    If Local.!port > 731 Then
     Do
       Say "Unable to bind to local port between 721 and 731"
       Return -1
     End
  End

  /* connect the new socket to the specified server     */
  Host.!family = "AF_INET"
  Host.!port = 515 /* default LPD port */
  rc = SockConnect(Socket, "Host.!")
  If rc < 0 Then
   Do
     Say "Unable to connect to server:" Server
     Self~Close
     Return -1
   End

  Return 0

/*******************************************************/
/*                                                     */
/* Method:    SendCommand                              */
/* Purpose:   Send a command via the connected socket  */
/*            and return the full response to caller.  */
/* Arguments: Command - command string                 */
/* Returns:   Response from server or empty string if  */
/*            failed.                                  */
/*                                                     */
/*******************************************************/
::METHOD SendCommand
  Expose Socket
  Use Arg Command

  /* append an LF character to end the command string */
  Command = Command || "0A"x
  BytesSent = SockSend(Socket, Command)
  Response = ""
  Do Forever
    BytesRcvd = SockRecv(Socket, "RcvData", 2048)
    If BytesRcvd <= 0 Then
      Leave
    Response = Response || RcvData
  End
```

```
  Return Response

/*****************************************************/
/*                                                   */
/* Method:    Close                                  */
/* Purpose:   Close the specified socket.            */
/* Arguments: None                                   */
/* Returns:   nothing                                */
/*                                                   */
/*****************************************************/
::METHOD Close
  Expose Socket

  Call SockShutDown Socket, 2
  Call SockClose Socket
  Return

/* --------------------- LPR Server data class -----------------------------*/

::CLASS LPRServerData
::METHOD Server ATTRIBUTE
::METHOD Printer ATTRIBUTE
::METHOD Agent ATTRIBUTE
::METHOD Default ATTRIBUTE
::METHOD List CLASS ATTRIBUTE

::METHOD Init CLASS
  /* initialize the list that will hold all LPRServerData objects */
  self~List = .Directory~New

::METHOD New CLASS
  /* intercept creation of a new LPRServerData object */
  /* see if this is an existing entry we can update  */
  If self~List~HasIndex(Arg(1)) Then
   Do
     /* Update existing entry */
     Entry = self~List[Arg(1)]
     Entry~Server = Arg(1)
     Entry~Printer = Arg(2)
     Entry~Agent = Arg(3)
     Entry~Default = Arg(4)
   End
  Else
   Do
     /* Create new entry */
     Forward Class(Super) Continue

     /* add newly created object to list */
     Entry = Result
     self~List[Entry~Server] = Entry
   End

  Return Entry

::METHOD Init
  Use Arg Server, Printer, Agent, Default
  self~Server = Server
  self~Printer = Printer
  self~Agent = Agent
  self~Default = Default

/* --------------------- Profile evaluation Routine  ------------------------*/

::ROUTINE ReadProfile
  Profile = .Stream~New(.environment["LPQMGR_PATH"] || "LPQMgr.Prf")

  /* find out default agent name (USERNAME@local ip address) */
  User = Value("USERNAME",, "ENVIRONMENT")
  If Length(User) = 0 Then
    User = Value("USER",, "ENVIRONMENT")

  /* depending on what host format is used you either need to  uncomment */
  /* the retrieval of the local IP address or the environment variable   */
  /* COMPUTERNAME                                                        */
  LocalHost = SockGetHostId()
  /* LocalHost = Value("COMPUTERNAME",, "ENVIRONMENT") */
  DefaultAgent = User || "@" || LocalHost
  DefaultPrinter = "afccu2"

  /* initialize to empty strings */
  Server = ""
  Printer = DefaultPrinter
  Agent = DefaultAgent
  Default = 0
```

```
Do While Profile~Lines > 0
   Line = Profile~LineIn
   Keyword = Line~Word(1)~Translate
   Data = Line~SubStr(Line~WordIndex(2))

   Select
    When Keyword = "SERVER" | Keyword =  "DEFAULTSERVER" Then
     Do
       /* store old entry if it is valid */
       If Server \= "" & Printer \= "" & Agent \= "" Then
          .LPRServerData~New(Server, Printer, Agent, Default)

       /* set new server and clear other fields */
       Server = Data
       Printer = DefaultPrinter
       Agent = DefaultAgent
       If Keyword = "SERVER" Then
         Default = 0
       Else
         Default = 1
     End

    When Keyword = "PRINTER" Then
      Printer = Data

    When Keyword = "AGENT" Then
      Agent = Data
    End
 End

 /* store last entry if it is valid */
 If Server \= "" & Printer \= "" & Agent \= "" Then
   .LPRServerData~New(Server, Printer, Agent, Default)

 Profile~Close
```

LPQMgr.PRF

```
DEFAULTSERVER pr3130f.ae.boeblingen.ibm.com ($3130E06)
SERVER pr3130c.ae.boeblingen.ibm.com ($3130E03)
SERVER pr3130b.ae.boeblingen.ibm.com ($3130E02)
```

SMTPMail.CMD

```
/*******************************************************/
/*                                                     */
/* SMTPMAIL.CMD - IBM REXX Sample Program              */
/*                                                     */
/* Send a mail to a list of recipients through a  SMTP */
/* mail server. The recipients and the sender will be  */
/* extracted from the mail file from the appropriate   */
/* lines ("From:", "To:", "Cc:"). Blind copying is not */
/* supported. The mail will be sent as read from the   */
/* input file, no lines will be added or removed.      */
/*                                                     */
/* The address format in the From/To/Cc lines allows   */
/* two formats:                                        */
/*   "From: Real Name <email@domain.com>"              */
/*   "From: email@domain.com"                          */
/*                                                     */
/* If <> is found then the enclosed string will be used */
/* as the email address, otherwise the first word after */
/* the token From/To/Cc will be used as  email address. */
/*                                                     */
/* Parameters:                                         */
/*   MailFile: Fully specified file name containing the */
/*             mail document.                          */
/*   SMTPServer: Server alias and optional port address */
/*               (if not at default  SMTP port)        */
/*                                                     */
/* Author:                                             */
/*   Christian Michel (cmichel@de.ibm.com)             */
/*                                                     */
/* History:                                            */
/*  1998/10/26 - 1.00 Initial release to public        */
/*                                                     */
/* --------------------------------------------------- */
```

```
/* (C) Copyright IBM Corp. 1998 - All Rights Reserved.   */
/*                                                       */
/* DISCLAIMER OF WARRANTIES.  The following [enclosed]   */
/* code is sample code created by IBM Corporation. This  */
/* sample code is not part of any standard or IBM        */
/* product and is provided to you solely for the         */
/* purpose of assisting you in the development of your   */
/* applications.  The code is provided "AS IS", without  */
/* warranty of any kind.  IBM shall not be liable for    */
/* any damages arising out of your use of the sample     */
/* code,  even if they have been advised of the          */
/* possibility of such damages.                          */
/*                                                       */
/*******************************************************/
Parse Arg MailFile SMTPServer

If MailFile = "" | SMTPServer = "" Then
 Do
   Say "Missing arguments. See program header for options"
   Return -2
 End

/* Load REXX Socket library if not already loaded       */
If RxFuncQuery("SockLoadFuncs") Then
 Do
   Call RxFuncAdd "SockLoadFuncs","RXSOCK","SockLoadFuncs"
   Call SockLoadFuncs
 End

/* Load REXX Utilities if not already loaded            */
If RxFuncQuery("SysLoadFuncs") Then
 Do
   Call RxFuncAdd "SysLoadFuncs","REXXUTIL","SysLoadFuncs"
   Call SysLoadFuncs
 End

/* First of all scan through the mail file to find out   */
/* the sender and all recipients.                        */
Sender = ""
Recip.0 = 0
Mail.0 = 0

Do While Lines(MailFile) > 0
  Idx = Mail.0 + 1
  Input = Linein(MailFile)
  Mail.Idx = Input
  Mail.0 = Idx

  If Translate(Word(Input, 1)) = "FROM:" Then
   Do
     If (Pos("<", Input) \= 0) &,
        (Pos(">", Input) \= 0) Then
      /* email address is enclosed in "<>" */
      Parse Var Input . "<" Sender ">" .
     Else
      Sender = Word(Input, 2)
   End

  If (Translate(Word(Input, 1)) = "TO:") |,
     (Translate(Word(Input, 1)) = "CC:") Then
   Do
     If (Pos("<", Input) \= 0) &,
        (Pos(">", Input) \= 0) Then
      /* email address is enclosed in "<>" */
      Parse Var Input . "<" Recipient ">" .
     Else
      Recipient = Word(Input, 2)

     i = Recip.0 + 1
     Recip.i = Recipient
     Recip.0 = i
   End
End

Call Stream MailFile, "C", "CLOSE"

/* check for at least one recipient and sender  */
If Sender = "" Then
 Do
   Say "No sender found in mail file"
   Return -1
 End

If Recip.0 = 0 Then
```

```
 Do
   Say "No recipients found in mail file"
   Return -1
 End

/* now find out the local host name (req'd by SMTP) */
LocHost.!addr = SockGetHostId()
If SockGetHostByAddr(LocHost.!addr, "LocHost.!") = 0 Then
 Do
   Say "Could not resolve local host name"
   Return -1
 End

/* Now connect to the server and send the mail item */
Socket = Connect(SMTPServer)
If Socket = -1 Then
  Return -1

/* initialize the session with the SMTP Server */
If SendCommand(Socket, "HELO" LocHost.!Name) = -1 Then
 Do
   Call Close Socket
   Return -1
 End

/* set the sender of the mail item */
If SendCommand(Socket, "MAIL FROM:<" || Sender || ">") = -1 Then
 Do
   Call Close Socket
   Return -1
 End

/* set all recipients of the mail item */
Do Idx = 1 To Recip.0
  If SendCommand(Socket, "RCPT TO:<" || Recip.Idx || ">") = -1 Then
   Do
     Call Close Socket
     Return -1
   End
End

/* initiate mail text transfer */
If SendCommand(Socket, "DATA", 354) = -1 Then
 Do
   Call Close Socket
   Return -1
 End

/* transfer the mail data, special handling for leading dot */
MailData = ""
Do Idx = 1 To Mail.0
  Line = Mail.Idx || "0D0A"x
  If Left(Line, 1) = "." Then
    Line = "." || Line

  MailData = MailData || Line
End

/* set end of mail marker and send the mail data */
MailData = MailData || "."
If SendCommand(Socket, MailData) = -1 Then
 Do
   Call Close Socket
   Return -1
 End

/* Terminate mail server session */
Call SendCommand Socket, "QUIT", 221
Call Close Socket
Exit 0

/*****************************************************/
/*                                                   */
/* Function:  Connect                                */
/* Purpose:   Create a socket and connect it to server.  */
/* Arguments: Server - server name, may contain port  no.*/
/* Returns:   Socket number if successful, -1 otherwise  */
/*                                                   */
/*****************************************************/
Connect: Procedure
  Parse Arg Server

  /* if the servername has a port address specified     */
  /* then use this one, otherwise use the default  smtp  */
```

```
  /* port 25                                                  */
  Parse Var Server Server ":" Port
  If Port = "" Then
    Port = 25

  /* resolve server name alias to dotted IP address     */
  rc = SockGetHostByName(Server, "Host.!")
  If rc = 0 Then
   Do
     Say "Unable to resolve server:" Server
     Return -1
   End

  /* create a TCP socket                                      */
  Socket = SockSocket("AF_INET", "SOCK_STREAM", "0")
  If Socket < 0 Then
   Do
     Say "Unable to create socket"
     Return -1
   End

  /* connect the new socket to the specified server     */
  Host.!family = "AF_INET"
  Host.!port = Port
  rc = SockConnect(Socket, "Host.!")
  If rc < 0 Then
   Do
     Say "Unable to connect to server:" Server
     Call Close Socket
     Return -1
   End

  /* read welcome string from SMTP server */
  Response = ""
  Do Forever
    BytesRcvd = SockRecv(Socket, "RcvData", 4096)
    If BytesRcvd <= 0 Then
      Leave
    Response = Response || RcvData
    If BytesRcvd < 4096 Then
      Leave
  End

  If GetRC(Response) \= 220 Then
   Do
     Say "SMTP Server not ready:" Response
     Call Close Socket
     Return -1
   End

  Return Socket

/*******************************************************/
/*                                                     */
/* Function:  SendCommand                              */
/* Purpose:   Send a command via the specified socket  */
/*            and return the full response to caller.  */
/* Arguments: Socket - active socket number            */
/*            Command - command string                 */
/*            ExpRc - expected return code             */
/* Returns:   Response from server or empty string if  */
/*            failed.                                  */
/*                                                     */
/*******************************************************/
SendCommand: Procedure
  Parse Arg Socket, Command, ExpRC

  /* append a pair of CRLF to end the command string */
  Command = Command || "0D0A"x
  BytesSent = SockSend(Socket, Command)
  Response = ""
  Do Forever
    BytesRcvd = SockRecv(Socket, "RcvData", 4096)
    If BytesRcvd <= 0 Then
      Leave
    Response = Response || RcvData
    If BytesRcvd < 4096 Then
      Leave
  End

  If ExpRc = "" Then
    ExpRc = 250

  If GetRC(Response) \= ExpRc Then
```

```
   Do
     Say "Error executing command:" Command "-" Response
     Return -1
   End
  Return 0

/*******************************************************/
/*                                                     */
/* Procedure: Close                                    */
/* Purpose:   Close the specified socket.              */
/* Arguments: Socket - active socket number            */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
Close: Procedure
  Parse Arg Socket

  Call SockShutDown Socket, 2
  Call SockClose Socket
  Return

/*******************************************************/
/*                                                     */
/* Procedure: GetRC                                    */
/* Purpose:   Isolate the return code of the answer.   */
/* Arguments: Response - full response string          */
/* Returns:   numeric return code from server          */
/*                                                     */
/*******************************************************/
GetRC: Procedure
  Parse Arg ServerRC .
  Return ServerRC
```

CREATEDB.REX

```
/* Create database and table */
"DB2 -O CREATE DATABASE REXXSAMP ON C"
"DB2 -O CONNECT TO REXXSAMP"
"DB2 -O CREATE TABLE REXX.CUSTOMER(CUSTNUM SMALLINT NOT NULL, CUSTNAME CHAR(40) NOT NULL, CUSTADDR
CHAR(40), CUSTCITY CHAR(40), CUSTSTATE CHAR(20), CUSTZIP INTEGER, CUSTPHONE CHAR(30))"
"DB2 -O CONNECT RESET"
```

LOADTBL.REX

```
/* Load a DB2 table from an ASCII file with column names  */
Table = "CUSTOMER"
Database = "REXXSAMP"
Schema = "REXX"
Datafile = "address.lst"

/* register DB2 functions */
Call RxFuncAdd "SqlExec", "DB2AR", "SqlExec"
Call RxFuncAdd "SqlDbs", "DB2AR", "SqlDbs"

/* connect to database */
Call MySqlExec "CONNECT TO" Database

/* read table columns from data file */
Col.0 = 0
Line = LineIn(DataFile)
Do While Pos("|", Line) > 0
  Parse Var Line ColName "|" Line
  Idx = Col.0 + 1
  Col.Idx = Strip(ColName, "B")
  Col.0 = Idx
End

/* assemble the columns for the insert statement */
Columns = "("
Do Idx = 1 To Col.0
  If Idx < Col.0 Then
    Columns = Columns Col.Idx ","
  Else
    Columns = Columns Col.Idx ")"
End

/* now read the table data */
Do While Lines(DataFile) > 0
```

```
  Line = LineIn(DataFile)
  Val.0 = 0
  Do While Pos("|", Line) > 0
    Parse Var Line DataVal "|" Line
    Idx = Val.0 + 1
    Val.Idx = Strip(DataVal, "B")
    Val.0 = Idx
  End

  /* assemble the insert statement */
  Values = "VALUES ("
  Do Idx = 1 To Val.0
    If Datatype(Val.Idx) = "NUM" Then
      Values = Values Val.Idx
    Else
      Values = Values "'" || Val.Idx || "'"

    If Idx < Val.0 Then
      Values = Values ","
    Else
      Values = Values ")"
  End

  /* insert data into table */
  Call MySqlExec2 "INSERT INTO" Schema || "." || Table Columns Values
End

/* commit changes and disconnect */
Call MySqlExec "COMMIT"
Call MySqlExec "CONNECT RESET"
Exit

MySqlExec:
  Parse Arg Args
  Call SqlExec Args
  If sqlca.sqlcode \= 0 Then
   Do
     Say "SQL Statement failed:"
     Say "  Statement:" Args
     Say "  SQLCode:" sqlca.sqlcode
     Call SQLDBS 'GET MESSAGE INTO :errmsg LINEWIDTH 80'
     Say "  Message:"
     Say errmsg
   End
  Return sqlca.sqlcode

MySqlExec2:
  Parse Arg stmt
  Call SqlExec "EXECUTE IMMEDIATE :stmt"
  If sqlca.sqlcode \= 0 Then
   Do
     Say "SQL Statement failed (EXECUTE IMMEDIATE):"
     Say "  Statement:" stmt
     Say "  SQLCode:" sqlca.sqlcode
     Call SQLDBS 'GET MESSAGE INTO :errmsg LINEWIDTH 80'
     Say "  Message:"
     Say errmsg
   End
  Return sqlca.sqlcode
```

ADDRESS.LST

```
CUSTNUM | CUSTNAME         | CUSTADDR          | CUSTCITY    | CUSTSTATE | CUSTZIP | CUSTPHONE      |
1       | Donald Duck      | 1 Disney Drive    | Orlando     | FL        | 12345   | (515)-123-4567 |
2       | Neil Armstrong   | 754546 Moon Blvd  | Space City  | TX        | 98734   | (453)-434-9836 |
3       | Henry Ford       | 65 Blackcar Lane  | Motor Town  | MI        | 45723   | (234)-234-1235 |
```

LISTTBL.REX

```
/* List contents of a DB2 table */
Table = "CUSTOMER"
Database = "REXXSAMP"
Schema = "REXX"

/* register DB2 functions */
Call RxFuncAdd "SqlExec", "DB2AR", "SqlExec"
Call RxFuncAdd "SqlDbs", "DB2AR", "SqlDbs"
```

```
/* connect to database */
Call MySqlExec "CONNECT TO" Database

/* get a list of columns in this table */
SelectStmt = "SELECT C.NAME FROM SYSIBM.SYSCOLUMNS C",
             "WHERE C.TBNAME='" || Table || "'"
Call MySqlExec "PREPARE s1 FROM :SelectStmt"
Call MySqlExec "DECLARE c1 CURSOR FOR s1"
Call MySqlExec "OPEN c1"

Columns.0 = 0
Do Until sqlca.sqlcode \= 0
  Call MySqlExec "FETCH c1 INTO :Column"
  If sqlca.sqlcode = 0 Then
   Do
     /* Store column name */
     Idx = Columns.0 + 1
     Columns.Idx = Column
     Columns.0 = Idx
   End
End
Call MySqlExec "CLOSE c1"

/* now query all columns from this table */
SelectStmt = "SELECT"
Vals = ""
Do Idx = 1 To Columns.0
  SelectStmt = SelectStmt "C." || Columns.Idx
  Vals = Vals ":Value" || Right(Idx, 3, "0")
  If Idx < Columns.0 Then
   Do
     SelectStmt = SelectStmt || ","
     Vals = Vals || ","
   End
End

SelectStmt = SelectStmt "FROM" Schema || "." || Table "C ORDER BY 1"

Call MySqlExec "PREPARE s1 FROM :SelectStmt"
Call MySqlExec "DECLARE c1 CURSOR FOR s1"
Call MySqlExec "OPEN c1"

Do Until sqlca.sqlcode \= 0
  Call MySqlExec "FETCH c1 INTO" Vals
  If sqlca.sqlcode = 0 Then
   Do Idx = 1 To Columns.0
     Say Left(Columns.Idx || ":", 15) Value('Value' || Right(Idx, 3, "0"))
   End
  Say
End

Call MySqlExec "CLOSE c1"

/* disconnect */
Call MySqlExec "CONNECT RESET"
Exit

MySqlExec:
  Parse Arg Args
  Call SqlExec Args
  If sqlca.sqlcode \= 0 & sqlca.sqlcode \= 100 Then
   Do
     Say "SQL Statement failed:"
     Say "  Statement:" Args
     Say "  SQLCode:" sqlca.sqlcode
     Call SQLDBS 'GET MESSAGE INTO :errmsg LINEWIDTH 80'
     Say "  Message:"
     Say errmsg
   End
  Return sqlca.sqlcode
```

HLLAPI.CLS

```
/* Load HLLAPI interface library if not already loaded */
If RxFuncQuery("HLLAPI") Then
  Call RxFuncAdd "HLLAPI","SAAHLAPI","HLLAPISRV"

::CLASS HLLAPISession PUBLIC
::METHOD INIT
  Expose Session
```

```
  Use Arg ID
  Session = .HLLAPISessionStatus~New(ID)

::METHOD Connect
  Expose Session
  Return HLLAPI("Connect", Session~ID)

::METHOD Disconnect
  Expose Session
  Return HLLAPI("Disconnect")

::METHOD QuerySessionStatus
  Expose Session

  Data = HLLAPI("Query_Session_Status", Session~ID)
  If Data = "" Then
    Return .nil

  Session~LongName = Data~SubStr(2, 8)~Strip("B")
  Session~Type = Data~SubStr(10, 1)
  Select
    When Session~Type = "D" Then
      Session~TypeStr = "3270 Host"
    When Session~Type = "E" Then
      Session~TypeStr = "3270 Printer"
    When Session~Type = "F" Then
      Session~TypeStr = "5250 Host"
    When Session~Type = "G" Then
      Session~TypeStr = "5250 Printer"
    When Session~Type = "H" Then
      Session~TypeStr = "ASCII"
    Otherwise
      Session~TypeStr = "Unknown type:" Session~Type
  End

  Session~Characteristics = Data~SubStr(11, 1)~C2X~X2B
  Session~Rows = Data~SubStr(12, 2)~Reverse~C2D
  Session~Columns = Data~SubStr(14, 2)~Reverse~C2D
  Session~Codepage = Data~SubStr(16, 2)~Reverse~C2D
  Return Session

::METHOD CopyPS
  Expose Session

  Data = HLLAPI("Copy_PS")

  If Data = "" Then
    Return .nil

  If Length(Data) \= Session~Rows * Session~Columns Then
    self~QuerySessionStatus /* get correct dimensions */

  /* now create an array with every line in a single element  */
  Screen = .Array~New
  Do Row = 1 To Session~Rows
    Screen[Row] = Data~SubStr((Row - 1) * Session~Columns + 1, Session~Columns)
  End

  Return Screen

::METHOD Wait
  Return HLLAPI("Wait")

::METHOD CopyOIA
  Return HLLAPI("Copy_OIA")

::METHOD SendKey
  Use Arg Keys
  Return HLLAPI("Sendkey", Keys)

::METHOD ClearScreen
  Return self~SendKey("@C")

::METHOD ReceiveFile
  Expose Session

  If Arg() \= 2 & Arg() \= 3 Then
    Return 2 /* a parameter error occurred */

  PCFilename = Arg(1)
  HostFilename = Arg(2)

  /* add a session ID if it's not there */
  If HostFilename~Pos(":") = 0 Then
```

```
    HostFilename = Session~ID || ":" || HostFilename

  /* assemble command string */
  Command = PCFilename HostFilename
  If Arg() = 3 Then
    Command = Command || " (" || Arg(3)

  Return HLLAPI("Receive_file", Command)

::METHOD SendFile
  Expose Session

  If Arg() \= 2 & Arg() \= 3 Then
    Return 2 /* a parameter error occurred */

  PCFilename = Arg(1)
  HostFilename = Arg(2)

  /* add a session ID if it's not there */
  If HostFilename~Pos(":") = 0 Then
    HostFilename = Session~ID || ":" || HostFilename

  /* assemble command string */
  Command = PCFilename HostFilename
  If Arg() = 3 Then
    Command = Command || " (" || Arg(3)

  Return HLLAPI("Send_file", Command)

::METHOD QueryCursorPos
  Expose Session
  Position = HLLAPI("Query_cursor_pos")
  If Position = 0 Then
    Return .nil

  self~QuerySessionStatus
  Return .HLLAPICursorPosition~New(Position, Session~Columns)

::METHOD SetCursorPos
  Expose Session

  If Arg() = 1 & Arg(1)~Class = .HLLAPICursorPosition Then
     Return HLLAPI("Set_cursor_pos", Arg(1)~Position)

  If Arg() = 2 Then
   Do
     /* Parameters are row/column */
     self~QuerySessionStatus
     Return HLLAPI("Set_cursor_pos", (Arg(1) - 1) * Session~Columns + Arg(2))
   End

  Return 1 /* error */

::METHOD QuerySystem
  Data = HLLAPI("Query_system")
  If Data = "" Then
    Return .nil

  Return .HLLAPISystemInfo~New(Data)

::METHOD SearchPS
  Expose Session

  If Arg() \= 1 & Arg() \= 2 Then
    Return .nil /* nothing found due to parameter error */

  If Arg() = 2 Then
    StartPos = Arg(2)
  Else
    StartPos = 1

  Position = HLLAPI("Search_PS", Arg(1), StartPos)
  If Position = 0 Then
    Return .nil

  self~QuerySessionStatus
  Return = .HLLAPICursorPosition~New(Position, Session~Columns)

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

::METHOD SendCommand
  Use Arg Command

  /* check whether "ENTER" key is part of the specified command */
```

```
  If Command~Right(2) \= "@E" Then
    Command = Command || "@E"

  /* clear the screen, then send command */
  self~ClearScreen
  self~SendKey(Command)

  /* now read the output of the command */
  If HLLAPI("Wait") \= 0 Then
    Return .False

  Return .True

::METHOD ReadReply
  Use Arg Command

  /* check whether "ENTER" key is part of the specified command */
  If Command~Right(2) = "@E" Then
    Command = Command~Left(Command~Length - 2)

  SeekCommand = .True
  Reply = .Array~New
  ReplyRow = 0

  Do Screens = 1
    Screen = self~CopyPS
    If Screen = .nil Then
      Return .nil /* this is an error condition */

    /* command output will not be in the last two rows */
    Do Row = 1 To Screen~Last - 2
      /* we skip all lines until the command string */
      If SeekCommand = .True & Screen[Row]~Pos(Command) = 0 Then
        Iterate

      /* we found the command - after this lines come the replies */
      If SeekCommand = .True & Screen[Row]~Pos(Command) = 1 Then
       Do
         SeekCommand = .False
         Iterate
       End

      /* if the current line is "Ready;" then we're done */
      If Screen[Row]~Pos("Ready;") = 1 Then
        Leave Screens

      ReplyRow = ReplyRow + 1
      Reply[ReplyRow] = Screen[Row]
    End

    /* whole screen read, see if there is more data */
    If Screen[Screen~Last]~Pos("MORE...") \= 0 Then
     Do
       /* We have more data waiting */
       self~ClearScreen
       self~Wait
     End
    Else
      /* we're done with the reply */
      Leave Screens
  End

  If Reply~Items = 0 Then
    Reply = .nil /* something went wrong */

  Return Reply

/* ----------------------------------------------------------------------------*/

::CLASS HLLAPISessionStatus PUBLIC
::METHOD INIT
  Expose ID LongName Type TypeStr Characteristics Rows Columns Codepage
  Use Arg ID
  LongName = "N/A"
  Type = "N/A"
  TypeStr = "N/A"
  Characteristics = "N/A"
  Rows = "0"
  Columns = "0"
  Codepage = "N/A"

::METHOD ID ATTRIBUTE
::METHOD LongName ATTRIBUTE
::METHOD Type ATTRIBUTE
```

```
::METHOD TypeStr ATTRIBUTE
::METHOD Characteristics ATTRIBUTE
::METHOD Rows ATTRIBUTE
::METHOD Columns ATTRIBUTE
::METHOD Codepage ATTRIBUTE

/* ---------------------------------------------------------------------------*/

::CLASS HLLAPISystemInfo PUBLIC
::METHOD INIT
  Expose EHLLAPIVersion HardwareBase ProgramType EmulatorVersion NLSType,
         MonitorType

  Use Arg Data

  EHLLAPIVersion = Data~SubStr(1, 1) || "." || Data~SubStr(2, 2)~Strip("B")
  EmulatorVersion = Data~SubStr(17, 1) || "." || Data~SubStr(18, 1)

  HardwareBase = Data~SubStr(13, 1)
  NLSType = Data~SubStr(30, 2)~Reverse~C2D

  ProgramType = Data~SubStr(14, 1)
  If ProgramType = "P" Then
    ProgramType = "IBM Personal Communications"


  MonitorType = Data~SubStr(32, 1)
  Select
    When MonitorType = "A" Then
      MonitorType = "PS/2 Monochrome"
    When MonitorType = "B" Then
      MonitorType = "PS/2 Monochrome Model 30"
    When MonitorType = "C" Then
      MonitorType = "CGA"
    When MonitorType = "E" Then
      MonitorType = "EGA"
    When MonitorType = "G" Then
      MonitorType = "MCGA"
    When MonitorType = "H" Then
      MonitorType = "XGA"
    When MonitorType = "M" Then
      MonitorType = "Monochrome"
    When MonitorType = "V" Then
      MonitorType = "VGA"
    When MonitorType = "U" Then
      MonitorType = "Unknown"
  End

::METHOD String
  Expose EHLLAPIVersion HardwareBase ProgramType EmulatorVersion NLSType,
         MonitorType

  Return "HLLAPI System Info - Version:" || EHLLAPIVersion ,
         "Emulator Version:" || EmulatorVersion ,
         "Program Type:" || ProgramType ,
         "NLS Type:" || NLSType "Monitor:" || MonitorType ,
         "Hardware Base:" || HardwareBase

::METHOD EHLLAPIVersion ATTRIBUTE
::METHOD HardwareBase ATTRIBUTE
::METHOD ProgramType ATTRIBUTE
::METHOD EmulatorVersion ATTRIBUTE
::METHOD NLSType ATTRIBUTE
::METHOD MonitorType ATTRIBUTE

/* ---------------------------------------------------------------------------*/

::CLASS HLLAPICursorPosition PUBLIC
::METHOD INIT
  Expose Row Column Position
  Use Arg Position, Columns
  Row = Position % Columns + 1
  Column = Position // Columns

::METHOD String
  Expose Row Column
  Return "Row:" || Row "Col:" || Column

::METHOD Row ATTRIBUTE
::METHOD Column ATTRIBUTE
::METHOD Position ATTRIBUTE
```

MoveFls.REX

```
/* Move files from host to PC */
Parse Arg HostFileMask PCPath HostSession

/* set default parameters */
If HostFilemask = "" Then
  HostFilemask = "* NOTEBOOK A"
If PCPath = "" Then
  PCPath = Directory()
If HostSession = "" Then
  HostSession = "A"

/* translate filemask (change dot to space) */
HostFilemask = HostFilemask~Translate(" ", ".")

/* translate slashes in PCPath to backslashes */
PCPath = PCPath~Translate("\", "/")

/* PCPath should contain a backslash at the end */
If PCPath~Right(1) \= "\" Then
  PCPath = PCPath || "\"

/* start a new HLLAPI session */
Session = .HLLAPISession~New(HostSession)
Session~Connect

/* get a list of all files matching filemask */
Command = "listfile" HostFilemask "(isodate noheader"
Session~SendCommand(Command)
CommandReply = Session~ReadReply(Command)

Do FileInfo Over CommandReply
  HostFile = FileInfo~Left(20)
  PCFile = PCPath || HostFile~Word(1)~Strip("B") || "." ||,
           HostFile~Word(2)~Strip("B")

  Options = "ASCII CRLF"
  If Stream("PCFile", "C", "QUERY EXIST") \= "" Then
    Options = Options "APPEND"

  Say "Receiving file:" HostFile "to:" PCFile
  If Session~ReceiveFile(PCFile, HostFile, Options) = 3 Then
   Do
     FileDate = FileInfo~SubStr(57, 10)
     FileTime = FileInfo~SubStr(68, 8)~Translate("0", " ")
     Say "Setting PC file date to:" FileDate FileTime
     Call SysSetFileDateTime PCFile, FileDate, FileTime
     Say "Erasing host file:" HostFile
    /* Session~SendCommand("ERASE " || HostFile) */
   End
End

Session~Disconnect
Exit

::REQUIRES HLLAPI.CLS
```

Registry.REX

```
/* show the .WindowsRegistry class */
myRegistry = .WindowsRegistry~New             /* create a new registry object */
If myRegistry~InitCode \= 0 Then Exit         /* leave if init failed */

/* Open the HKEY_LOCAL_MACHINE\System key */
If myRegistry~Open(, "SYSTEM", "QUERY WRITE") \= 0 Then Do
  SysKey = myRegistry~Current_Key
  If myRegistry~List(, Keys.) = 0 Then Do     /* Get a list of System's subkeys */
    /* open first subkey's Control key */
    CntrlKey = myRegistry~Open(, Keys.1 || "\Control")
    If CntrlKey \= 0 Then Do
      /* Get and display information about the Control key.  */
      q. = myRegistry~Query
      Say "Control key is of class:" q.Class
      Say "It was last modified on:" q.Date "at" q.Time
      Say "It has" q.Subkeys "subkeys, and" q.Values "values."

      Say
```

```
      /* list all subkeys */
      Say "Subkeys of this key:"
      If myRegistry~List(, Keys.) = 0 Then
        Do i Over Keys.
         Say Keys.i
        End

      Say

      Drop Name Data Type   /* these 3 symbols must be uninitialized */

      /* list all values of the first subkey */
      Say "Values of this key:"
      If myRegistry~ListValues(, Vals.) = 0 Then
        Do i = 1 To q.Values
          Say Vals.i.Name "=" Vals.i.Data "(" || Vals.i.Type || ")"
        End
    End
    myRegistry~Close(CntrlKey)    /* close Control key */
  End

  myRegistry~Close(SysKey)   /* close System key */
  Say
End

/* create a new subkey under HKEY_CURRENT_USER and modify it */
If myRegistry~Create(myRegistry~Current_User, "REXX Demo") \= 0 Then Do

  /* set the default value */
  myRegistry~SetValue(, "", "Presented at SHARE 92")

  /* add other values */
  myRegistry~SetValue(, "Location", "San Francisco")
  myRegistry~SetValue(, "Year", "1999", "NUMBER")   /* type NUMBER for numbers */

  /* retrieve the previously added values */
  Say "The following entries have been added to the registry:"
  Value. = myRegistry~GetValue(, "")
  Say "Default value:" Value.Data
  Value. = myRegistry~GetValue(, "Location")
  Say "Location:" Value.Data
  Value. = myRegistry~GetValue(, "Year")
  Say "Year:" Value.Data "of type" Value.Type

  myRegistry~DeleteValue(, "Year")   /* delete a value */

  /* and now delete the whole key */
  myRegistry~Delete(myRegistry~Current_User, "Rexx Demo")
End

/* remove external functions */
myRegistry~Deinstall

::REQUIRES WINSYSTM.CLS /* load definition for .WindowsRegistry class */
```

WinMgr3.REX

```
/* Show features of the .WindowObject class */
wMgr = .WindowsManager~New
"start notepad"
Call SysSleep 1
np = wMgr~Find("Untitled - Notepad")
edit = np~FirstChild /* get window object for edit window */

Childs. = np~EnumerateChildren   /* Get a list of all its child windows */
say "The Notepad editor has the following child windows:"
Do i = 1 To Childs.0
  Say "Class:   " Childs.i.!Class
  Say "ID:      " Childs.i.!ID
  Say "Title:   " Childs.i.!Title
  Say "Coord:   " Childs.i.!Coordinates
  Say "Handle:  " Childs.i.!Handle
  Say "State:   " Childs.i.!State
  Say "Children:" Childs.i.!Children
End

Say "The Notepad window currently is located at:"  np~Coordinates
Say "It will soon be sized and moved..."
Call SysSleep 3
np~Resize(400, 300)
np~MoveTo(50, 50)
```

```
np~ToForeground

edit~SendText("Have you seen the window being moved?")
edit~SendKey("RETURN")
Call SysSleep 3

edit~SendText("Now the window will be hidden and restored...")
edit~SendKey("RETURN")
Call SysSleep 3
np~Hide
Call SysSleep 2
np~Restore
edit~SendText("It was no longer visible in the task list!")
edit~SendKey("RETURN")

edit~SendText("Now the window will be minimized, then maximized and restored...")
edit~SendKey("RETURN")
Call SysSleep 3
np~Minimize
Call SysSleep 2
np~Maximize
Call SysSleep 2
np~Restore

edit~SendText("Watch the title bar...")
edit~SendKey("RETURN")
Call SysSleep 3
np~Title = "Notepad - Started from and controlled by  REXX"
Call SysSleep 2

np~SendText("That's it. Notepad will be closed now without saving the changes...")
Call SysSleep 2

np~SendSysCommand("CLOSE")

/* Now wait a second for the dialog that asks whether to save the changes  */
Call SysSleep 1
fg = wMgr~ForegroundWindow
If fg \= .Nil Then Do
  /* If it is the dialog then decline to save  */
  If fg~Title = "Notepad" Then
    fg~PushButton("&No")
End

wMgr~Deinstall  /* Deinstall the WindowsManager and clean up */

::REQUIRES WINSYSTM.CLS
```