Overview

J-Term Pro provides you with some of the most powerful scripting available in any telnet client. The J-Term Pro scripting language is a subset of the Pascal programming language, with special enhancements for performing tasks such as opening and closing connections, using FTP from a script, and capturing text coming from the remote host. This help file is designed to give you enough of an introduction to scripting with J-Term Pro to get you started writing your own scripts. A good text on the Pascal language might be helpful, too.

The following script rules apply:

All J-Term Pro script blocks begin with the word **begin** and end with the word **end**.
All script statements should be followed with a semi-colon (;).
Any variables your script uses should be declared outside the **begin..end;** block.
Comments can be enclosed in curly braces {}, or anything after two slashes (//) on a line is considered a comment.
J-Term Pro is case insensitive. ABC = aBc = abc. Literal strings are case sensitive.

Below is a very simple script to show the format of a typical J-Term Pro script. Try typing it in the script manager and running it now.

```
// Script to get a user's name

var Name: String;   // Variable to hold the name

begin
  Name:=InputBox('Getting user name','Please enter your name','');
  ShowMessage('Hello, '+Name);
end;
```

Data types and variables

J-Term Pro has the ability for your script to use variables for holding values of data. These variables can be any one of six basic types:

- Integer - Integer variables hold whole numbers in the range –32768..32767
- LongInt - LongInt variables hold whole numbers in the range -2147483648..2147483647
- Real - Real variables hold real numbers in the range
- String – String variables hold text strings, like file names, or machine names. Literal strings in J-Term Pro are surrounded by single quotes.
- Char – Char variables hold a single character. Literal characters in J-Term Pro are surrounded by single quotes.
- Boolean – Boolean values hold one of two values, true or false.

As mentioned before, variables are used to hold data values. To declare a variable for use in your script, use the following syntax:

```
var VarName: VarType;
```

The reserved word ‘var’ need only be used once in any variable block declaration. Multiple variables of the same type can be declared together, with a comma separating the names. Variable names are composed of combinations of uppercase and lowercase letters. Here are some sample variable declarations in J-Term Pro:

```
var X, Y, Z: Integer;
    Name, Address: String;
    Income: Real;
    Status: Char;
```

Assignment statements

Variables in J-Term Pro are assigned values with assignment statements. Assignment statements take the following form:

```
VarName:=Expression;
```

Where 'Expression' is a valid value or mathematical expression. Here are some examples, in a sample script:

```
var MyInt: Integer;
    MyStr: String;
    MyBoolean: Boolean;
    MyReal: Real;
    MyChar: Char;
    MyLongInt: LongInt;


begin
  MyInt:=5;
  MyStr:='J-Term Pro scripting is great!';
  MyBoolean:=True;
  MyReal:=4.756 + 2.345;
  MyChar:='A';
  MyLongInt:=MyInt*5;   // Would yield 5*5 or 25 as the value
end;
```

A variable can also be assigned the return value of a J-Term Pro function. For example (using the variables from above):

```
MyStr:=InputBox('Getting user name','Please enter your name','');
```

places the return value of the InputBox function (a string) into the MyStr variable. See the section on functions for a fuller explanation of function return values.

Data conversion functions

IntToStr

```
function IntToStr(IntValue:Integer): String;
```

The IntToStr function is used to convert an integer value to a string value.

```
var MyStr: String;
    MyInt: Integer;

begin
  MyInt:=5;
  MyStr:=IntToStr(MyInt);
  MyStr:=IntToStr(3);
end;
```

StrToInt

```
function StrToInt(StrValue:String): Integer;
```

The StrToInt function is used to convert a string value to an integer value.

```
var MyStr: String;
    MyInt: Integer;

begin
  MyInt:=StrToInt('1234');
  MyStr:='556';
  MyInt:=StrToInt(MyStr);
end;
```

* Note – a string value as the argument to this function that is not a valid integer will yield an error.

FloatToStr

```
function FloatToStr(RealValue:Real): String;
```

The FloatToStr function is used to convert a real value to a string value.

```
var MyReal: Real;

begin
  MyReal:=123.456;
  ShowMessage(FloatToStr(MyReal));
end;
```

StrToFloat

```
function StrToFloat(StrValue:String): Real;
```

The StrToFloat function is used to convert a string value to a real value.

```
var MyString: String;
```

```
begin
  MyString:='123.456';
  MyReal:=StrToFloat(MyString);
  MyReal:=StrToFloat('54.56');
end;
```

* Note - a string value as the argument to this function that is not a valid real will yield an error.

String functions

Length

```
function Length(StrValue:String): Integer;
```

The length function returns the length, in characters, of the given string.

```
var MyInt: Integer;

begin
  MyInt:=Length('J-Term Pro scripting');
end;
```

Pos

```
function Pos(SubStr, StrValue:String): Integer;
```

The Pos function returns the position of the string SubStr in the string StrValue. The position is returned as an integer representing the offset from the beginning of StrValue where SubStr is located.

```
var MyPos: Integer;

begin
  MyPos:=Pos('erm','J-Term Pro');
  // MyPos is 4, because 'erm' starts at position 4 in 'J-Term Pro'
end;
```

Copy

function Copy(StrValue:String; Start,Count:Integer): String;

The Copy function copies Count characters from the Start position in StrValue. The function returns the copied substring as a result.

```
var MyStr: String;

begin
  MyStr:=Copy('J-Term Pro',3,4);
  // MyStr is now 'Term'  - Copy copied the four characters starting
  // from position three
end;
```

Insert

```
procedure Insert(Str1:String; var Str2:String; Index:Integer);
```

The Insert procedure inserts Str1 into Str2 at position Index.

```
var Str: String;

begin
  Str:='J-Term Pro scripting';
  Insert('does ',Str,12);
```

```
  // Str is now 'J-Term Pro does scripting'
end;
```

Delete

```
procedure Delete(var Str:String; Start,Count:Integer);
```

The Delete procedure deletes Count characters from Str starting at position Start.

```
var Str: String;

begin
  Str:='J-Term Pro does scripting';
  Delete(Str,12,5);
  // Str is now 'J-Term Pro scripting'
end;
```

Loop control

The for *loop*

```
for LoopVar:=StartVal to|downto EndVal do statement;
```

The for loop counts from StartVal to EndVal by 1, executing *statement* each time. LoopVar, StartVal, and EndVal are all integers or longints. To count backwards, use downto. If *statement* needs to be a block of statements, surround the block with `begin..end;`

```
var X: Integer;
    Str: String;

begin
  for X:=1 to 5 do
    ShowMessage(IntToStr(X));
  Str:='';
  for X:=5 downto 1 do
  begin
    Str:=Str+IntToStr(X);
    ShowMessage(Str);
  end;
  // Str is now '54321'
end;
```

The while *loop*

```
while condition do statement;
```

The while loop executes *statement* as long as condition is true. If *statement* needs to be a block of statements, surround the block with `begin..end;`

```
var X, Y: Integer;

begin
  X:=1;
  Y:=0;
  while X<100 do
  begin
    Y:=Y+X;
    X:=X+1;
  end;
  ShowMessage(IntToStr(Y));
  // Y is the sum of all the numbers from 1 to 100
end;
```

The repeat *loop*

```
repeat statement until condition;
```

The repeat loop repeats *statement* until *condition* is true. *statement* can be a block of statements without the necessity of a `begin..end`.

```
var X, Y: Integer;

begin
  X:=1;
  Y:=0;
  repeat
    Y:=Y+X;
    X:=X+1;
  until X=101;
  // Y is the sum of the integers from 1 to 100
end;
```

Conditions

The if *statement*

```
if condition then statement1
[ else statement2 ];
```

The if statement is used to test one or more *conditions* and perform *statement1* if that/those conditions are true. If you are testing multiple conditions, each condition must be in a set of parentheses, and the conditions must be separated by reserved words and or or.

If you use the optional else, then *statement2* is performed if *condition* is false. Additionally, if you use an else, *statement1* must *not* have a semi-colon at the end.

Valid comparison operators for conditions are:

- = - equal to
- < - less than
- > - greater than
- <> - not equal to
- <= - less than or equal to
- >= - greater than or equal to
- not – for Boolean types only. (not true) = false, and (not false) = true

```
var Age, Loop: Integer;
    AgeStr: String;
    IsValid: Boolean;
    AgeChar: Char;


begin
  IsValid:=False;
  while (not IsValid) do // Make sure they enter a valid number
  begin
    AgeStr:=InputBox('Getting age','Please enter your age','');
     IsValid:=True;
    for Loop:=1 to Length(AgeStr) do
    begin
      AgeChar:=Copy(AgeStr,Loop,1);
      // Check to make sure each char is a valid number
      if (AgeChar<'0') or (AgeChar>'9') then
        IsValid:=False;
    end;
    if (not IsValid) then
      ShowMessage('Sorry, '+AgeStr+
                  ' is not a valid number. Please try again.');
  end;
  Age:=StrToInt(AgeStr);
  if (Age<21) then ShowMessage('Still a kid.')
  else if (Age<40) then ShowMessage('Wet behind the ears')
  else if (Age<65) then ShowMessage('Middle aged')
  else ShowMessage('Would you like a 10% discount?');
end;
```

Input and output

Input

InputBox

```
function InputBox(Caption,Prompt,Default:String): String;
```

The InputBox function displays a dialog that allows the user to enter a string. Caption is used as the caption for the dialog, Prompt is used as a string prompt, and Default is a default value for the string entered. The function returns the string entered as it's value.

```
var Name: String;

begin
  Name:=InputBox('Getting name','Please enter your name','');
  if (Name<>'') then ShowMessage('Hello, '+Name);
end;
```

InputQuery

function InputQuery(Caption, Prompt: String; var Value:String): Boolean;

The InputQuery function displays a dialog that allows the user to enter a string. Caption is used as the caption for the dialog. Prompt is used as a string prompt, and Value is a variable to hold the value of the string entered. You can set Value to some default string before you call InputQuery. The function returns True if the user clicks the Ok button, and False otherwise.

```
var Name: String;

begin
  Name:='Bubba';
  if (InputQuery('Getting name','Please enter your name',Name)) then
    ShowMessage('Hello, '+Name);
end;
```

GetUserPass

```
function GetUserPass(var User, Pass:String; Caption:String): Boolean;
```

The GetUserPass function opens a dialog prompting the user to enter a username and password. The password is masked by asterisks, so it can't be read from the screen. Caption is used as the caption for the dialog. The function returns True if the user clicks 'Ok' and False if the user clicks 'Cancel'.

```
var UserName, Password: String;

begin
  GetUserPass(UserName,Password,'Enter account info');
end;
```

GetScreenText

```
function GetScreenText(PageNo, StartRow, StartCol, StopRow, StopCol:
                       Integer): String;
```

The GetScreenText returns a string containing the screen text inside the box defined by the start/stop rows and columns. The rows and columns are zero-based, meaning row numbers go from 0 to 24 or 49 and columns are from 0 to 79 or 131, depending on your connection configuration.

```
var X: String;

begin
  // Grab the whole screen
  X:=GetScreenText(ActivePage,0,0,10,79);
  // Display the 3rd line
  ShowMessage(Copy(X,(80*2)+1,80));
end;
```

Output

ShowMessage

```
procedure ShowMessage(Msg:String);
```

The ShowMessage procedure displays a dialog box with Msg inside it. The user can click the Ok button to dismiss the dialog.

```
var Name: String;

begin
  Name:=InputBox('Getting name','Please enter your name','');
  if (Name<>'') then ShowMessage('Hello, '+Name);
end;
```

WriteStatus

```
procedure WriteStatus(Msg:String);
```

The WriteStatus procedure writes Msg to the J-Term Pro status bar.

```
var Name: String;

begin
  Name:=InputBox('Getting name','Please enter your name','');
  if (Name<>'') then WriteStatus('Hello, '+Name);
end;
```

ClearStatus

```
procedure ClearStatus;
```

The ClearStatus procedure clears the J-Term Pro status bar of any messages.

```
begin
  ClearStatus;
end;
```

StartFileLog

```
procedure StartFileLog(PageNo:Integer; FileName:String);
```

The StartFileLog procedure turns on file logging for the specified open connection (PageNo). The data sent to and received from the session is stored in the file specified by the FileName parameter. The logging can be stopped by closing the connection, clicking ‘Log to file’ on the Tools menu, or by calling the StopFileLog procedure.

```
var PN: Integer;

begin
  PN:=OpenConnection('JAI Linux');
  StartFileLog(PN,'C:\TMP\JAI.LOG');
end;
```

StopFileLog

```
procedure StopFileLog(PageNo:Integer);
```

The StopFileLog procedure turns off data logging to a file for the specifiect connection.

```
begin
  StopFileLog(ActivePage);
end;
```

File handling

FileCreate

```
function FileCreate(FileName:String): Integer;
```

The FileCreate function creates an empty file with the name FileName on your hard drive. The return value of FileCreate is an integer handle to the newly created file. This handle can be used in the FileRead, FileWrite, and FileClose routines. FileCreate will return a –1 if the specified file could not be created. The FileName can include an absolute path.

```
var FileDes: Integer;

begin
  FileDes:=FileCreate('C:\TMP\TEST.TXT');
  if (FileDes>=0) then
  begin
    FileWrite(FileDes,'J-Term Pro',10);
    FileClose(FileDes);
  end;
end;
```

FileOpen

```
function FileOpen(FileName:String; Mode:Integer): Integer;
```

The FileOpen function opens the file named FileName on your hard drive. The return value of FileOpen is an integer handle to the newly opened file. This handle can be used in the FileRead, FileWrite, and FileClose routines. The Mode of the file tells how the file will be used, and can be fmOpenRead, fmOpenWrite, or fmOpenReadWrite. FileOpen will return a –1 if the specified file could not be opened (for example, if it doesn't exist). The FileName can include an absolute path.

```
var FileDes: Integer;
    Data: String;

begin
  FileDes:=FileOpen('C:\TMP\TEST.TXT',fmOpenRead);
  if (FileDes>=0) then
  begin
    FileRead(FileDes,Data,255);
    FileClose(FileDes);
    ShowMessage(Data);
  end;
end;
```

FileClose

```
procedure FileClose(FileDes:Integer);
```

The FileClose procedure will close the open file specified by FileDes. See the help for FileOpen or FileCreate for an example of using this procedure.

FileRead

```
function FileRead(FileDes:Integer; var Data:String;
                  BytesToRead:Integer): Integer;
```

The FileRead function attempts to read BytesToRead bytes into Data from the open file specified by FileDes. The file must have previously been opened with mode fmOpenRead or fmOpenReadWrite. The function returns the number of bytes actually read.

```
var FileDes: Integer;
    Data: String;


begin
  FileDes:=FileOpen('C:\TMP\TEST.TXT',fmOpenRead);
  if (FileDes>=0) then
  begin
    FileRead(FileDes,Data,255);
    FileClose(FileDes);
    ShowMessage(Data);
  end;
end;
```

FileWrite

```
function FileWrite(FileDes:Integer; Data:String;
                   BytesToWrite:Integer): Integer;
```

The FileWrite function attempts to write BytesToWrite bytes from Data into the open file specified by FileDes. . The file must have previously been opened with mode fmOpenWrite or fmOpenReadWrite. The function returns the number of bytes actually written.

```
var FileDes: Integer;


begin
  FileDes:=FileCreate('C:\TMP\TEST.TXT');
  if (FileDes>=0) then
  begin
    FileWrite(FileDes,'J-Term Pro',10);
    FileClose(FileDes);
  end;
end;
```

FileExists

function FileExists(FileName:String): Boolean;

The FileExists function returns True if the file specified by FileName exists on the hard drive; otherwise, it returns False. An absolute path can precede the absolute file name.

```
begin
  if (FileExists('C:\CONFIG.SYS')) then
    ShowMessage('You have a CONFIG.SYS file')
  else ShowMessage('You do NOT have a CONFIG.SYS file');
end;
```

Writing procedures and functions

J-Term Pro scripting provides them means for you to write your own procedures and functions for your scripts, because sometimes scripts get too long and cumbersome to have everything in one big `begin..end` block. The following rules apply when writing procedures and functions:

- Procedure and function names can be any combination of upper and lower case letters
- All procedures and functions must be defined *above* the main `begin..end` block of the script.
- Variables declared inside a procedure or function are local to that procedure or function only. They are not known outside the procedure or function.
- Procedures and functions can use global variables from the script, as long as the procedures and functions come after the global variable declarations.
- Any time you want to modify the value of a parameter to a procedure or function, you must use the reserved word 'var' in front of the variable declaration in the parameter list.

Procedures

```
procedure ProcName[(parameter list)];
[ var section ]
begin
end;
```

As an example, let's revisit the script from the help on the `if` statement, this time rewritten to use a procedure.

```
var Age: Integer;
    AgeStr: String;
    IsValid: Boolean;

procedure IsValidInt(Str:String; var IsValid:Boolean);
var Loop: Integer;
    AgeChar: Char;
begin
  IsValid:=True;
  if (Str='') then Result:=False
  else
    for Loop:=1 to Length(AgeStr) do
    begin
      AgeChar:=Copy(AgeStr,Loop,1);
      // Check to make sure each char is a valid number
      if (AgeChar<'0') or (AgeChar>'9') then
        IsValid:=False;
    end;
end;


begin
  repeat
    AgeStr:=InputBox('Getting age','Please enter your age','');
    IsValidInt(AgeStr,IsValid);
    if (not IsValid) then
      ShowMessage('Sorry, '+AgeStr+
                  ' is not a valid number. Please try again.');
  until (IsValid) or (AgeStr='');
  if (AgeStr='') then AgeStr:=0;
  Age:=StrToInt(AgeStr);
  if (Age<21) then ShowMessage('Still a kid.')
  else if (Age<40) then ShowMessage('Wet behind the ears')
```

```
  else if (Age<65) then ShowMessage('Middle aged')
  else ShowMessage('Would you like a 10% discount?');
end;
```

Functions

```
function FuncName[(parameter list)]: ResultType;
[var section]
begin
end;
```

Functions are like procedures, except they can have a return value. The type of the return value is spceified by the ResultType. To set the return value for a function, use Result. Let's revisit the script from the help on the `if` statement, this time rewritten with functions.

```
var Age: Integer;
    AgeStr: String;
    IsValid: Boolean;

function IsValidInt(Str:String): Boolean;
var Loop: Integer;
    AgeChar: Char;
begin
  Result:=True;
  if (Str='') then Result:=False
  else
    for Loop:=1 to Length(AgeStr) do
    begin
      AgeChar:=Copy(AgeStr,Loop,1);
      // Check to make sure each char is a valid number
      if (AgeChar<'0') or (AgeChar>'9') then
        Result:=False;
    end;
end;

begin
  repeat
    AgeStr:=InputBox('Getting age','Please enter your age','');
    IsValid:=IsValidInt(AgeStr);
    if (not IsValid) then
      ShowMessage('Sorry, '+AgeStr+
                  ' is not a valid number. Please try again.');
  until (IsValid) or (AgeStr='');
  if (AgeStr='') then AgeStr:=0;
  Age:=StrToInt(AgeStr);
  if (Age<21) then ShowMessage('Still a kid.')
  else if (Age<40) then ShowMessage('Wet behind the ears')
  else if (Age<65) then ShowMessage('Middle aged')
  else ShowMessage('Would you like a 10% discount?');
end;
```

Controlling J-Term Pro

J-Term Pro scripting provides many procedures and functions you can use to control the behavior of the program.

Connections

OpenConnection

```
function OpenConnection(ConnectionName: String): Integer;
```

Opens a new connection to remote host. The ConnectionName parameter must be a valid connection name as defined in the Connection Manager. The return value of this function is the page number that J-Term Pro gives the new connection. Page numbers are zero-based, meaning the first open connection gets page 0, the second gets page 1, etc.

CloseConnection

```
procedure CloseConnection(ConnectionName: String);
```

Closes the first open connection in the connection pages that has a name that matches the name in the ConnectionName parameter. See the help for `OpenConnection` for a cross reference.

ActiveConnection

```
function ActiveConnection: String;
```

Returns the name of the currently active connection in J-Term Pro.

ClosePage

```
procedure ClosePage(PageNo: :Integer);
```

Closes page number PageNo in J-Term Pro.

ActivePage

```
function ActivePage: Integer;
```

Returns the page number of the currently active page in J-Term Pro.

SetActivePage

```
procedure  SetActivePage(PageNo: Integer);
```

Sets the current active page to PageNo

SendLine

```
procedure SendLine(PageNo: Integer; Text: String);
```

Sends the sequence of characters in Text to the connection specified by PageNo, followed by a carriage-return, as though you had typed the line.

SendText

```
procedure SendText(PageNo: Integer; Text: String);
```

Sends the sequence of characters in Text to the connection specified by PageNo, as though you had typed them.

ConnectionName

```
function ConnectionName: String;
```

Returns the name of the last connection opened, whether it is the active connection or not.

CloseAllConnections

```
procedure CloseAllConnections;
```

Should be self-evident.

HostName

```
function HostName(ConnectionName:String): String;
```

This function returns the DNS host name or IP address of the given connection name.

CloseProgram

```
procedure CloseProgram;
```

Exits J-Term Pro, closing all open connections.

Using FTP in scripts

FTPConnect

```
function FTPConnect(HostName, UserName, Password: String): Boolean;
```

This function opens an FTP connection to the given host, using the account specified in UserName/Password. It returns a true on successful connection, or a false if it was unable to connect.

FTPDisconnect

```
function FTPDisconnect: Boolean;
```

This function disconnects the current FTP connection opened with FTPConnect. It returns a true on a successful disconnect, or a false if the disconnect failed.

FTPBinary

```
procedure FTPBinary;
```

This procedure sets the FTP transfer mode to binary mode.

FTPAscii

```
procedure FTPAscii;
```

This procedure sets the FTP transfer mode to ASCII mode.

FTPSend

```
function FTPSend(LocalFile, RemoteFile:String): Boolean;
```

This function sends the file specified by LocalFile to the remote host. The file is sent to the location specified by RemoteFile. The function returns a true if the file was successfully sent, or a false it if was not.

FTPReceive

```
function FTPReceive(LocalFile, RemoteFile): Boolean;
```

This function retrieves the file specified by RemoteFile and places it on the local machine in the location specified in LocalFile. The function returns a true if the file was successfully received, or a false if it was not.

Database Storage functions

J-Term Pro provides some limited database-type storage routines for saving variables and data across scripts. Though the data is not stored in an INI file, the same principle is used. Imagine the data being stored like this:

[SectionName]
Key1=Value1
Key2=Value2

[AnotherSection]
Key1=Value1
Key2=Value2

SetValue

```
procedure SetValue(Section,Key,Value: String);
```

This procedure is used to store data values in a local database. Value is stored associated with Key under Section.

GetValue

```
function GetValue(Section,Key: String): String;
```

This function retrieves the specified value from the local database.

DeleteValue

```
procedure DeleteValue(Section,Key: String): String;
```

This procedure removes an entry from the local database.

Running scripts, events, timers, and programs

RunScript

```
procedure RunScript(ScriptName: String);
```

This procedure executes the script specified in the ScriptName parameter. The parameter must have the *exact* script name, including the extension, if there is one. It may be a good idea to specify the full path to the script, too.

ExecuteFile

```
procedure ExecuteFile(FileName, Params, StartDir: String;
                      Style: Integer);
```

This procedure runs the file specified by FileName, using the command line parameters specified in Params. The file is started in the directory specified my StartDir, and the initial program window is opened with the style set in the Style parameter. Valid values for Style are: SW_SHOWNORMAL, SW_SHOWMAXIMIZED, and SW_SHOWMINIMIZED.

ExecuteFileWithWait

```
function ExecuteFileWithWait(FileName,Params:String;
                             Style:Integer): Boolean;
```

This function runs the file specified by FileName, using the command line parameters specified in Params. The file is started in the directory specified my StartDir, and the initial program window is opened with the style set in the Style parameter. Valid values for Style are: SW_SHOWNORMAL, SW_SHOWMAXIMIZED, and SW_SHOWMINIMIZED. The function does not return until the program file executed is closed. The function returns True if all went well, or False if the specified program could not be run.

GlobalEvent

```
procedure GlobalEvent(EventNo: Integer);
```

This procedure causes a global event to fire, thus executing any code associated with that event. Valid values for EventNo are: OnBeforeConnect, OnConnect, OnStart, OnStop, OnWatch, and OnDisconnect. If you override an event for a connection, and want the global event to fire after the connection event, use this procedure, or the global event will never get called.

SetTimer

```
procedure SetTimer(NumSecs: Integer; ScriptToRun:String);
```

This procedure sets a system timer to fire in NumSecs seconds. When the timer fires, the script in the file name specified by ScriptToRun is executed;

ClearTimer

```
procedure ClearTimer;
```

This procedure is used to clear (or disable) the sytem timer started with a call to `SetTimer`.

OnNetworkError

```
procedure OnNetworkError(PageNo: Integer; ScriptToRun:String);
```

This procedure sets the script to run should the connection in PageNo receive a network connection error. PageNo is an integer representing one of the open pages (connections) in J-Term Pro. PageNo is a zero-based number. Should a network error occur in the connection, ScriptToRun will be executed.

Watches

J-Term Pro scripting provides many ways to use a script to interact with a telnet session. One such way is to use *watches*. J-Term Pro has a watch list at the global level, and each connection also has a watch list. A watch list is a list of text J-Term Pro is watching for to come across a telnet connection. When text matching text in a watch list is seen, one of the following things can happen:

- A global OnWatch event can be fired
- A connection-level OnWatch event can be fired
- A script can be run

The following functions and procedures are used to deal with watches in J-Term Pro.

WatchText

```
function WatchText: String;
```

Returns the text of the last watch seen.

AddWatch

```
procedure AddWatch(WatchText: String; NumOccurrences: Integer);
```

This procedure adds a 'watch' for the text specified in the WatchText parameter to the active connection's watch list. When the specified text comes across the connection, the OnWatch event for that connection is fired, if the event is enabled. If the event is not enabled, the global OnWatch event is fired. The event will continue to fire until the WatchText has been seen NumOccurrences times. For example, set NumOccurrences to 1 if you only want to catch the text once.

AddGlobalWatch

```
procedure AddGlobalWatch(WatchText: String; NumOccurences: Integer);
```

Similar to the AddWatch procedure, but sets the watch at a global level. Any new connections that are created will also have the watch set.

AddWatchScript

```
procedure AddWatchScript(WatchText: String; NumOccurrences: Integer;
                         ScriptName: String);
```

Similar to the AddWatch procedure, but the script specified in ScriptName is run when the watch text is seen, instead of the OnWatch event being fired.

AddGlobalWatchScript

```
procedure AddGlobalWatchScript(WatchText: String;
                               NumOccurrences: Integer;
                               ScriptName: String);
```

Similar to the AddWatchScript procedure, but sets the watch at a global level. Any new connections that are created will also have the watch set.

ClearWatches

```
procedure ClearWatches(PageNo: Integer);
```

Empties the watch list for the specified page. Use the RemoveWatch procedure to remove a specific watch.

ClearGlobalWatches

```
procedure ClearGlobalWatches;
```

Empties the global watch list. Note: any connection that has had a watch set via a global watch will still have the watch set. Use the RemoveWatch procedure to clear out a specific watch.

RemoveWatch

```
procedure RemoveWatch(PageNo: Integer; WatchText: String);
```

Removes the watch specified by WatchText from the watch list of the connection specified by PageNo.

RemoveGlobalWatch

```
procedure RemoveGlobalWatch(WatchText: String);
```

Removes the global watch specified by WatchText. Any open connections that have had this watch set will retain the watch.

Sample Scripts

Auto-logins

Making J-Term automatically login for you when you open connections is not a difficult matter, especially if you write the scripts as event handlers. Make sure you enable the handler, or it will not run. Enable handlers by checking the 'Enabled' box at the bottom of the event editor. Here is one way to create auto-logins:

```
// Global OnStart event script
// ---------------------------------------
// This script tells J-Term Pro to look for
// the login and password prompts one time
// for each connection that is opened.

begin
  AddGlobalWatch('ogin:',1);
  AddGlobalWatch('assword:',1);
end;
```

```
// Global OnBeforeConnect event script

var UserName, Password: String;

begin
  // Load the stored user name and password
  UserName:=GetValue(ActiveConnection,'UserName');
  Password:=GetValue(ActiveConnection,'Password');
  // See if we need to prompt the user
  if (UserName='') or (Password='') then
  begin
    GetUserPass(UserName,Password,
                'Enter info for '+ActiveConnection);
    // Store the user name and password by connection
    SetValue(ActiveConnection,'UserName',UserName);
    SetValue(ActiveConnection,'Password',Password);
  end;
end;
```

```
// Global OnWatch event script

var UserName, Password: String;

procedure SendUserName(UN:String);
// Send user name to active page
begin
  SendLine(ActivePage,UN);
end;

procedure SendPassword(PW:String);
// Send password to active page
begin
```

```
  SendLine(ActivePage,PW);
end;

begin
  if (WatchText='ogin:') then
    SendUserName(GetValue(ActiveConnection,'UserName'))
  else if (WatchText='assword:') then
    SendPassword(GetValue(ActiveConnection,'Password'));
end;
```

Local file editing

The following script can be added to the J-Term Pro add-ins to provide local file editing, for those who don't like vi .

```
var FileName: String;
    UserName, Password: String;
    Host: String;

function RetrieveFile(FileName,UserName,Password:String): Boolean;
begin
  if (FTPConnect(Host,UserName,Password)) then
  begin
    FTPAscii;
    if (FTPReceive('C:\TMP.TXT',FileName)) then
    begin
      FTPDisconnect;
      Result:=True;
    end
    else Result:=False;
  end
  else Result:=False;
end;

procedure EditFile;
begin
  ClearStatus;
  ExecuteFileWithWait('NOTEPAD.EXE','C:\TMP.TXT',SW_SHOWNORMAL);
end;

function ReturnFile(FileName,UserName,Password:String): Boolean;
begin
  if (FTPConnect(Host,UserName,Password)) then
  begin
    FTPAscii;
    if (FTPSend('C:\TMP.TXT',FileName)) then
    begin
      FTPDisconnect;
      Result:=True;
    end
    else Result:=False;
  end
```

```
  else Result:=False;
end;

begin
  FileName:=InputBox('Enter full path to file','Enter remote file name to
edit','');
  if (FileName<>'') then
  begin
    Host:=HostName(ActiveConnection);
    GetUserPass(UserName,Password,'FTP account info');
    if (RetrieveFile(FileName,UserName,Password)) then
    begin
      EditFile;
      ReturnFile(FileName,UserName,Password);
    end;
  end;
end;
```