WinRamp™ Lite

Version 1.2

-Your Personal Access to the Data Super Highway℠-

Winramp Lite *is a full-featured communications software product distributed under the Shareware "Try Before You Buy" software marketing concept.*

WR-SCRIPT USER MANUAL

Vironix N.A., Inc.   *   P.O.Box 1570   *   Haverhill   *   MA   *   01831-998   *   USA

Tel: 1-508-373-2402   **Sales:**   1-800-VIRONIX   **Fax:**   1-508-374-7125   **BBS:** 1-508-373-3336   **EMAIL:** info@vironix.com

LICENCE AGREEMENT

Please carefully read the following terms and conditions. Continued use of *WinRamp Lite* constitutes your acceptance of these terms and conditions and your agreement to abide by them.

WinRamp Lite is a "shareware program" and is provided at no charge to the user for an evaluation period of 30 days. If you find this program useful and find that you are using *WinRamp Lite* after the 30 day evaluation period, please register the product (see registration details below). *WinRamp Lite* may not be modified in any way, and should be distributed with all supplied files in its orginal archive format: WRAMP111.ZIP or RMP111.ZIP (on CIS)

The 30-Day Free Evaluation Licence is a legal agreement between you, the end user, and Vironix
N.A., Inc. By using *WinRamp Lite*, you are agreeing to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement please discontinue using *WinRamp Lite*.

Business & Government Site Licence
After the 30 day Evaluation Period, any corporation, institution, government agency or business wishing to continue use of *WinRamp Lite* in the course of its internal business is required to purchase a *WinRamp Lite* registration or site licence. Any individual wishing to use *WinRamp Lite* within a corporation, institution, government agency or business must purchase a *WinRamp Lite* registration or site licence. The site licence is provided for those who want to *WinRamp Lit*e on multiple computers. Please see the REGISTER.WRI or contact Vironix N.A., Inc. for terms.

Upgrade Policy
Registered users of *WinRamp Lite* will automatically be sent a diskette (no printed manual) of the next MAJOR release (for example 1.x to 2.0 etc.) of *WinRamp Lite* at NO CHARGE. Registration serial numbers will expire after the SECOND major release (for example 1.x to 3.x etc.). Thereafter a discounted upgrade fee will apply to further major upgrade releases.

Users wishing to receive MINOR upgrade diskette sets (for example 1.1 to 1.2 etc.) will be charged only for the diskette and mailing. Registered users can apply their registration serial numbers to MINOR upgrade releases downloaded from the WinRamp Lite Support BBS or any online system where the shareware distribution archive can be found.

VIRONIX'S SHAREWARE DISTRIBUTION POLICY

The essence of shareware software is to provide users with an opportunity to evaluate quality software before buying it, thereby keeping prices low while still providing developers with an incentive to continue development. Shareware is a distribution method, not a type of software. Copyright laws apply to registered Shareware just the same as they do for commercial software.

WinRamp Lite as Shareware
WinRamp Lite is distributed under the "Try Before You Buy" Shareware marketing concept. *WinRamp Lite* is a FULLY FUNCTIONAL communications program and no features, commands or functions have been disabled or crippled in any way. *WinRamp Lite* is NOT free software, and we strongly encourage you to register your copy. With registration you will be entitled to numerous special benefits, including:

- the latest version of the software
- an optional printed manual
- one FREE major version upgrade

- full access to the ***WinRamp Lite*** **Support BBS**
- a full rebate of the registration fee when purchasing the commercial version of WinRamp directly from Vironix N.A., Inc.

Dual Channel Distribution Policy

A number of Shareware publishers, having attained a reasonable degree of success, have "graduated" away from the Shareware marketing and distribution concept, often citing the low rate of user registration as an obstacle to economic growth. Despite the obvious economic risks of publishing quality software under the Shareware concept, and due mainly to a strong belief in the future and sheer power of the "super data-highway" as a medium for publishing and distributing software, Vironix N.A. has deliberately embraced a Dual Channel distribution policy.

The elements of this Dual Channel distribution policy are:

1. To publish and distribute a fully functional application capable of competing (and beating) feature-for-feature with commercial/retail packages often costing FOUR times as much as *WinRamp Lite*. It is Vironix NA's policy to continue development and upgrading of *WinRamp Lite* in parallel to development of an even more powerful commercial product: **WinRamp**™, and to continue to commit to distributing *WinRamp Lite* as shareware.

2. To publish and market through standard retail channels a fully integrated, fully customisable communications environment, namely **WinRamp** with superior features - whilst offering registrants of the shareware *WinRamp Lite* a FULL REBATE of their registration fee when purchasing **WinRamp** directly from Vironix NA, Inc.

How this policy benefits you

Vironix NA's Dual Channel distribution policy directly benefits you, the end-user, as you are able to fully evaluate WinRamp Lite BEFORE spending a cent. Furthermore should you wish to upgrade to WinRamp, you will have TWO full products for the price of one.

REGISTRATION

Registering *WinRamp Lite* licenses you to use the product after the 30 day evaluation period. Registered users of *WinRamp Lite* will receive the latest version of the product, with a serial number, technical support and a bound user manual. A two-tiered pricing structure exists for registration: $35 (US) for a licensed copy of *WinRamp Lite*, free technical support and manual on disk; or $45 (US) which includes a licensed copy of *WinRamp Lite*, free technical support and a printed and bound manual. Bulk purchases are subject to discount. Please call our toll free sales number for details.

Prices and terms are subject to change without notice.

Methods of ordering:

1. Call direct to 1-800-VIRONIX (VISA, Mastercard)
   (USA & Canada only)

2. Fax order form: to 508-354-8559

3. Compuserve
   GO SWREG ID#: 3034

4. Vironix NA Support BBS - Dial 508-374-7125

5. Send cheque or postal order to
   Vironix N.A., Inc.
   P.O. Box 1570
   Haverhill
   MA 01831-998

WinRamp Lite may be purchased directly from the following International Distributors:

South Africa and Southern African countries:

Vironix Corporation (Pty). Ltd.
Suite 4: Buckhurst, Essex Gardens
Durban, 3630
Rep. of South Africa Tel: +27 31 266-8930

Fax or Post Order Form

WinRamp Lite Order Form

Name:	______________________________
Company:	______________________________
Address:	______________________________
	______________________________
	______________________________
Telephone number:	______________
Fax number:	______________
E-mail address:	______________

Item	**Unit Price**	**Qty**	**TOTAL**
WinRamp Lite Basic Registration..................	$35.00	___	$________
WinRamp Lite Registration with Manual..........	$45.00	___	$________
Add $1 per copy for 3.5" disks			$________
Shipping/Handling - USA/Canada	$10.00		$________
Shipping/Handling - Outside USA	$15.00		$________
Add applicable State & County Sales Tax			$________
Total in US Funds drawn on a US Bank			**$________**

(Check, money order, or credit card payments accepted).

For Credit Card Users Only:
Type of credit card [ ] VISA [ ] MasterCard [ ] American Express

Card number:	________________________________
Name (as it appears in card):	________________________________
Expiration date of card:	____/____/____
Signature:	________________________________

Where did you hear about or obtain WinRamp Lite from?

____

____

WHAT HAPPENS AFTER REGISTRATION

After you have registered *WinRamp Lite*, the following will occur:

1. Upon bank clearance of your registration fee (or credit card verification), you immediately will be issued with a TEMPORARY Registration Serial Number Key, which when typed at the Registration dialog will prevent any further "nagscreens" from appearing whilst running *WinRamp Lite.*

2. The next step depends on how you registered:

- If you have phoned through your credit card registration via one of the voice contact numbers, your temporary key will be read to you over the phone. This is subject to the verification of your credit card.

- If you have registered through the Automated Registration on the WinRamp Lite Support BBS, you will also immediately be issued with a temporary key whilst online.
- If you registered via email, a reply message will contain your temporary key.
- If you registered on CompuServe (GO SWREG), your temporary key will be sent to you via private CompuServe email.

3. A diskette set will be posted to you immediately, containing the full registered version of WinRamp Lite, in addition to a permanent serial number.

PRINTING THIS MANUAL

This manual has been prepared using the Microsoft Windows **Write** application. Applying the following settings should ensure the manual prints correctly:

- Left and right margins set at 1.25" and top and bottom margins set at 1". These are set in *Page Layout*, accessed from the Windows Write *Document* Menu.
- All Tabs stops cleared. Check this by using the *Tabs* option under the *Document* menu and clicking the *Clear All* button.
- Printing page size set for **letter 8 1/2 x 11**, under *Print Setup* accessed from the *File* menu.

Page breaks have been set for printing according to the above specifications. Should you choose to print on a different page size, formatting of the document will be misaligned, and page numbering in the Table of Contents will be incorrect. However, if you do choose to print using a different page size, it is recommended that you at least repaginate the document so that blank pages are not printed. Use the *Repaginate* command from the Write *File* menu, with the *Confirm Page Breaks* check box marked so that you can keep page breaks between chapters, but move the others to a more logical breaking point.

Remember that if you would like a more comprehensive printed and bound manual - register now!

TABLE OF CONTENTS

1. INTRODUCTION TO WR-SCRIPT

A script file is a short program written in the scripting language provided with *WinRamp Lite* - **WR-Script**. It is generally used to automate tasks you find you are performing repeatedly. In this way scripts are similar in concept to macros, except that a script can execute any number of keystrokes and other more complicated actions.

With **WR-Script**, a script can be recorded, edited, debugged and run. The process of creating or editing a script requires knowledge of a few basic programming principles, to be expounded on next.

2. WR-SCRIPT BASIC CONCEPTS

The basic programming construct is called a statement. A statement does one unit of work - it might calculate the sum of two numbers, declare a variable, or wait for some response from a remote system. Statements usually act on variables. A variable is a placeholder used to store results of a calculation. Every variable has a type that specifies what kind of result the variable can contain. WinRamp's scripting language, **WR-Script**, supports the following types:

Integer	A "whole" number, e.g. 128, or -421
Real	A number with a decimal point: e.g. 2.6423
Character	A single character, e.g. 'A'
String	A set of characters stored in order, e.g. "Hello World"

Single statements usually don't do very much. Calculating the sum of two numbers is interesting, but not very useful. It would be more useful to calculate, for example, your monthly bond repayments. With this concept in mind, statements are grouped into procedures and functions. A procedure (or function) is a logical grouping of statements that work together to perform some task. This grouping is entirely arbitrary - you can make a procedure do as much (or as little) as you see fit. The **WR-Script** language has several built-in procedures and functions to ease programming. For example, there is a function that will download a file.

Programs are built out of functions and procedures. Control always starts at a special procedure, called "Main".

To summarize, here is a script that works out the square root of a largish number using the built-in "SQRT()" function. The result is stored in a variable called "TheSquareRoot".

```
Procedure Main ()
   Integer MyNumber = 256 { Define variable MyNumber equal to 256 }
   Real TheSquareRoot = SQRT( MyNumber ) { Take it's square root }
End
```

You will notice that the above program includes a textual description of each instruction. These comments are ignored by the compiler - they serve only to make the code more readable to humans. Any character between the left brace '{' and right brace '}' are comments. Nested comments, like "{ This is a comment { and this is another } }" are permissible.

3. CREATING A SCRIPT

The most common method of creating a script is to record one, i.e. "teach" *WinRamp Lite* a sequence of events to be used again. You can record an entire communications session or just those sections that you repeat often, e.g. logging on, checking mail etc. A WR-Script AutoLearn script is similar to a Macro in that they both record the actions that you perform during a communication session and replicate these actions when you execute them. They can also both be used to perform an automatic logon. The main difference, however, is that a macro records only the actions you perform whereas a AutoLearn script records both the actions you perform as well as how the BBS responds. Due to this difference, an AutoLearn script may be more reliable than a macro.

In order to use the **AutoLearn** facility, first establish a connection with your desired host. Select the WR-Script AutoLearn option from the Scripts menu OR click the WR-Scripts AutoLearn button. The script dialog box is displayed, where you can select an existing script file from the appropriate directory or type the name of a new file. It is then just a case of performing the actions you would like to record in your script. When you have finished, deselect the WR-Scripts AutoLearn option from the Scripts menu or click the WR-Scripts AutoLearn button. This will turn the AutoLearn facility off.

In global settings, there is an option to set whether you wish script files created using Autolearn to be automatically compiled or not. If you have set this option to no, i.e. manual compile, then you must open the WR-Script IDE and compile the script from there (refer to the section on the WR-Script IDE) or use the Compile Script option from the Scripts menu in WinRamp.

An alternate method for creating a script file is to develop it from scratch, using the functions and operations available in the WR-Script Interactive Development Environment (**WR-Script IDE**). This environment is described next.

4. WR-SCRIPT INTEGRATED DEVELOPMENT ENVIRONMENT

4.1 Introduction

The WR-Script Integrated Development Environment (**WR-Script IDE**) provides tools for editing, debugging and testing WR-Script programs you have created to automate repetitive tasks.

There are two basic methods of creating a script:

1. You can use the "Auto-Learn" facility to capture a communications session, or part thereof, thereby automatically generating a script; or
2. You can use the **WR-Script IDE** to create a new script from scratch.

Once a WR-Script source code file (*.wrs) has been created, you can use the edit and debug facilities of the **WR-Script IDE** to finetune and correct errors in the script. Once tested, the script can be compiled and used as an integral part of *WinRamp Lite*. To launch the **WR-Script IDE**, select WR-Script IDE from the Debug menu.

Whenever you load a WR-Script source code file, the Script toolbar (initially activated on the bottom of the IDE Window) or menu item can be used to perform a variety of testing and debugging functions.

4.2 Debugging Scripts with the WinRamp IDE

The debugging tools available in the **WR-Script IDE** are:

Run/Resume Program

Clicking on the Run/Resume option from the Debug menu, or clicking on the Run/Resume Program button will execute the script as if it was being run in the live *WinRamp Lite* environment. The script will thus run at approximately its normal speed. This debugging tool is ideal to test the complete script, especially where flow is complex and contains many conditions and loops.

Run to Cursor

Clicking on this menu option or button will cause the **WR-Script IDE** to execute the script up to the position of the cursor. Choosing the Run to Cursor option will automatically set a breakpoint at the position of the cursor. This function is most useful in allowing you to test specific segments of the script execution.

Trace Into

This function, along with the Step Over function is probably the most useful tool for debugging. It allows you to execute the script one line at a time. When you choose Trace Into from the Debug menu, or click on the Trace Into button, the **WR-Script IDE** executes the currently highlighted instruction, moves the highlight to the next instruction, and pauses. If the highlighted instruction was a function created by the user, the IDE would "step into" the function, so the "next" instruction would be the first line of the function.

Step Over

This function is similar to the Trace Into function described above, except that it "skips" over procedures and functions. It thus only traces through the main script code, stepping over subroutines. Step Over is useful when you are trying to trap errors that you know occur in the main script code and not any of the procedures. When a script has no procedures, the Step Over option acts the same the Trace Into option.

Reset Program

Reset Program is the equivalent of halt or stop, where the current action you have chosen in the IDE is halted and the program reset to normal.

Toggle Breakpoint

Breakpoints are used to mark points in the script where you wish execution of the script to halt for debugging purposes. This may be because you are sure an error occurs somewhere before the breakpoint, or if you want to test portions of the script. A breakpoint should be defined when using the Run to Cursor. To define a breakpoint, place the cursor in the required position and select Toggle Breakpoint. To remove select Toggle Breakpoint again.

Find Execution Point

Selecting Find Execution Point from the Debug menu, or clicking on the associated button will cause the **WR-Script IDE** to jump to the last point of execution of the script, typically where the program has paused, e.g. during a Trace Into. This function is useful in long, complicated scripts where the line being executed currently not visible in the edit window because the script is too large.

Inspect Symbol

To Inspect a symbol or variable, you must be in the process of stepping over instructions or any other process where the script has paused (indicated by the instruction being highlighted). With a particular instruction highlighted, select the Inspect option from the Debug menu or click the relevant icon on the toolbar. The Inspect Variable dialog box will then be displayed, where you can enter a variable or select one from the drop-down list box. The value for that variable is then displayed. If you have marked a section of the program, the IDE will try and generate a variable name from the marked section, and automatically inspect that variable.

This dialog describes a variable's state at the current execution point in the script. Shown are the variable's scope (local or global), its type, whether or not its a constant, and its value. If the variable does not exist, or is out of scope, the dialog will describe the variable as "Unknown" and will not be able to report any details. All unknown fields are filled with "[???]".

Fake Device Input
In debugging mode, a script is not actually connected to any communications device. Real uploads and downloads are therefore impossible. If you wish to "pretend" that a device receives a string, click on the Fake Device Input option from the Debug menu. The Force Input dialog will allow you to simulate a upload or download. Select "Yes" to allow the upload or download to succeed, or "No" to pretend it failed.

4.3 Using the WR-Script IDE

A number of tools are available in the **WR-Script IDE** to facilitate editing and debugging. These are found under the Edit and Search menu options. Practically any action is reversible, using the Undo option from the Edit menu. Further, portions of scripts can be reused when creating or editing new ones that require similar functions. This is effected using the Cut, Copy and Paste options from the Edit menu. These functions also allow you to reorder sections of code within a script and copy sections where just a few instructions require modification.

The Search menu allows you to find instructions, phrases or words within the script using the Find option. Further, these can be replaced with a new instruction, phrase or word by selecting the Replace option. This will bring up a dialog box where you enter the string to search for and what to replace it with. The checkboxes should be marked if you require the search to match case, if you require all matched occurrences to be replaced or if you wish to be prompted before the matched string is replaced. The Search Again option will find the next match.

4.4 File Operations

All file operations are accessed through the File menu. To open a WR-Script source code file (*.wrs), select the Open option. This will bring up the Open dialog box where you can select the correct script source file and directory from the list provided. The source file could have been created using the Auto-learn feature available in the main *WinRamp Lite* program, or during a previous session using the IDE. Use the New option to create a new script file.

After debugging, editing or testing your script, use the Save option from the File menu. You will be prompted to enter a new file name and directory if this is the first time you have saved the script source. If you have saved previously, this option will overwrite the old version automatically. Should you wish to save the source under a different file name or directory, choose the Save As option. Exiting the WinRamp Script IDE is done using the File Exit option.

4.5 Compiling

Any script must be compiled before it can be executed. Compiling a script converts it from normal text into a compressed format more suitable for interpretation by *WinRamp Lite*. The compiler checks that all symbols used are defined, and that the script does not have any typecasting errors. If possible, the compiler automatically inserts code to convert types where necessary.

Why pre-compile the script? This approach has several advantages and almost no drawbacks. A compiled script is faster to interpret, and will have fewer run-time errors because the compiler will have rooted out most, if not all, of the problems.

A script can be compiled from the **WR-Script IDE** or from the Scripts menu. A script that has been created using the WR-Script AutoLearn facility may or may not need to be compiled depending on the setting of the AutoLearn Automatic Compile on Close option in the Script Interpreter section of the Misc Global Preferences. To compile in the **WR-Script IDE**, choose Compile or Make from the Compile menu. The make function will recompile the script only if it has changed since it was last compiled; the compile function always recompiles the script regardless of whether it has changed or not.

4.6 Error messages

While working in the WR-Script IDE, there are a number of error messages you may encounter. These include:

Unrecoverable Runtime Error
Runtime errors can be caused by corrupt script images (".wso" files), an actual coding error in the script (for example, a

division by zero), or an internal error in the scripting system (for example, running out of memory).

If the error is not caused by your program, try to recompile the script. This will usually correct corrupt script images.

Error Loading Executable Image
The script image (".wso" file) is corrupt on the disk. Attempt to recompile its source.

Cannot Save File
The IDE cannot save the chosen file. Please confirm that there is sufficient disk space, and that you can write to the drive.

Out of Memory
The WR-Script IDE has run out of memory. Please close any unused applications and try again.

Rebuild Script Object?
The IDE has detected that the source of this script has been modified since it was last compiled. Choose "Yes" to recompile the script now.

4.7 Operators and Operator Precedence

Operators available in WR-Script are:

Logical Operators

AND: Applies only to integers. True if and only if both operands are true. Example: A AND B.
NOT: Applies to integers. Any non-zero value is converted to zero, and zero is converted to some non-zero value. Example: NOT A.
OR: Applies only to integers. True if either one of the operands is true. Example: A OR B.

XOR: Logical exclusive or. Applies only to integers. True if one and only one of the operands is true. Example: A XOR B.

Sign Conversion Operators

Unary Positive: Applies to integers and reals. Equivalent to multiplying by 1. Example: +A.
Unary Negative: Applies to integers and reals. Equivalent to multiplying by -1. Example: -A.

Multiplication and Division

Multiplication: Applies to integers and reals. Example: A * B.
Division: Applies to integers and reals. This operator can cause the "Division by Zero" runtime error. Example: A / B.

Modulus Arithmetic

Modulus: Applies to integers. The modular arithmetic operator calculates the remainder of an integer division. Example: A%B.

Exponentiation

Exponentiation: Applies to integers and reals. Raises the first operand to the power of the second. This operator can cause a runtime error, as it is illegal to raise certain numbers to certain powers. Example: A^B.

Addition and Subtraction

Addition: Applies to all types. Example: A + B.
Subtraction: Applies to all integers and reals. Example: A - B.

Relational Operators

Greater than: Applies to all types. True if the first operand is greater than the second one. Example: A < B.
Less than: Applies to all types. True if the first operand is less than the second one. Example: A > B.
Greater than or equal: Applies to all types. True if the first operand is greater than or equal to the second one. Example: A >= B.
Less than or equal: Applies to all types. True if the first operand is less than or equal to the second one. Example: A <= B.
Not equal: Applies to all types. True if the first operand is not equal to the second one. Example: A <> B.
Equal: Applies to all types. True if the first operand is equal to the second one. Example: A = B.

WR-Script operators have an order of precedence - when several operations take place within the same program statement, certain kinds of operations will be performed before others. If the operations are of the same level of precedence, the first to be executed will be the leftmost, and the last, the rightmost. The following is the order in which operations are evaluated:

1. NOT
2. Sign Conversion
3. Multiplication, Modulus Arithmetic, Exponentiation and AND
4. Addition, Subtraction, OR and XOR
5. Relational Operators

4.8 WR-Script Type Declarations

WR-Script supports the following types:

Integer A "whole" number in the range -32768 to 32767, e.g. 128 or -421. You may wish to use hexadecimal to define integers - do so by preceding them with the ampersand '&' character, e.g. &A7 is equivalent to 167.

Real A number with a decimal point, e.g. 2.623. Allowable values range from $3.4x10^{-38}$ to $3.4x10^{38}$

Character A single character. Represented in WR-Script by enclosing the character in single quotes, e.g. 'A'.

String A set of characters stored in order. Represented in WR-Script by enclosing the string in double quotes, e.g. "Hello World". To include a double quote character in a string, precede it with the backslash character, for example, "*Hello \"World\"*" is actually the string *Hello* "*World*".

Constant

The constant keyword declares a variable to be unmodifiable for the duration of its scope.

Declaring Variables

When declaring variables, use this syntax:

```
constant type
      variable1 = initialvalue1,
      variable2 = initialvalue2,
      ...,
      variable9 = initialvalue9
```

The constant keyword is optional, as are the initial values. If no initial value is given, the variable will be set to 0 for numbers and characters, and "" for strings.

Examples:

```
constant integer True = 1

string j, k = "Hello"

character CR = Char ( 13 ), LF = Char ( 10 )
```

5. WORKING WITH WR-SCRIPT FUNCTIONS

5.1 Functional and Procedure Declaration

Function Declaration

```
Function FunctionName ( Parameter list ) Returns Type
  function body
End
```

Procedure Declaration

```
Procedure ProcedureName ( Parameter list )
  procedure body
End
```

Parameter lists can be empty, or can consist of a list of typed variables separated by commas. Parameters can be passed by **value** (any changes made to the parameter are NOT reflected in the calling procedure) or by **reference** (any changes made to the parameter are permanent).

By Value
`Type` *`ParameterName`* or `Type` *`ParameterName`* `By Value`

By Reference
`Type` *`ParameterName`* `By Ref`

Return Statement

```
Return ( Value )
Return
```

When used with a procedure, Return returns control to the calling procedure or function.

When used with a function, Return sets the value that the function will pass back to the calling procedure or function, then returns control to it.

5.2 Main() Function and Procedure

Syntax

```
Procedure Main ()
Function Main () Returns Integer
```

Main() is a special procedure or function at which execution starts. It may be defined as a function to allow scripts to return a value.

5.3 WR-Script Function Reference

Abs() Function

Purpose
Returns the absolute value of a number.

Syntax

```
Abs( real Value ) returns real
```

Remarks
This function converts a number to its absolute value (which ignores the sign).

Example

```
procedure Main ()
```

```
    real j = ABS( -2.3 )      { j = 2.3 }
end
```

Beep() Function

Purpose
Beeps.

Syntax
`Beep()`

Remarks
This function causes the computer’s speaker to make a beeping noise. This is usually used in a script to denote the end of an download or upload, or to indicate that an error has occurred. The beep function will have no effect if the [Windows] section of your win.ini specifies Beep=no.

Example
```
procedure Main ()
      Beep ()
      Delay ( 50 )
      Beep ()
end
```

Char() Function

Purpose
Converts an integer to a character.

Syntax
Char(*integer Value* **)** returns character

Remarks
This function will convert a given integer value into the equivalent character. This is useful for building strings with unprintable characters like carriage return and linefeed.

For example, `string UserName = "Fred" + Char ( 13 )` would define the string UserName as "Fred" followed by ASCII code 13, which is a carriage return.

Example
```
procedure Main ()
      SendChar( Char ( 13 ) )
end
```

CloseConnection() Function

Purpose
Closes any currently active connections on the device associated with the script.

Syntax
`CloseConnection()`

Remarks
Closing an already closed device has no effect.

Example
```
Procedure Main ()
      CloseConnection ()
End
```

CloseTerm() Function

Purpose
Closes the current terminal.

Syntax
`CloseTerm ()`

Remarks
Since scripts are associated with a terminal, closing a terminal causes the script to terminate.

Example
```
procedure Main ()
     CloseTerm()
end
```

Date() Function

Purpose
Returns the current date in a specified format.

Syntax
`Date(` *`string Format`* **`)`** `returns string`

Remarks
Provides the current system date. The string *Format* determines how the result is formed. A *y*, or group of *y*'s will be substituted with the current year. Similarly, *m*'s are months, and *d*'s are days. For example, "`yyyy/mm/dd`" would return `1994/08/01` if it happened to be the first of August, 1994.

Example
```
procedure Main ()
     string TimeString = Time ( "hh:mm:ss" )
     string DateString = Date ( "yyyy-mm-dd" )

     MessageBox ( "It is now " +
                  TimeString + ", " +
                  DateString + ".", "Time", 0 )
end
```

Delay() Function

Purpose
Delays for a given time.

Syntax
`Delay(` *`integer Time`* **`)`**

Remarks
The script will stop executing for the specified time (in milliseconds).

Example
```
procedure Main ()
     SendChar ( 'a' )
     Delay ( 1000 )
     SendChar ( 'b' )
end
```

Download() Function

Purpose
Downloads a file using a specified protocol.

Syntax
```
Download( string Filename, string Protocol ) returns integer
```

Remarks
Download returns one if the download succeeds and zero if it fails. The protocol must match one of the protocol names, as defined in WinRamp's protocol configuration. If the protocol does not exist, the download will fail. Not all protocols require a filename, for example ZModem autodetects the filename. If the protocol does autodetect a filename, it will override any given filename.

Example
```
procedure Main ()
     Download( "", "ZModem" )  {ZModem needs no filename - it can autodetect}
end
```

FindWindow() Function

Purpose
Finds a window and focuses it.

Syntax
```
FindWindow( string WindowTitle ) returns integer
```

Remarks

Locates and sets focus to a specified Window. Identification is by the title of the Window. If this function returns zero, no window exists that has the title *WindowTitle*.

Example

```
procedure Main ()
     constant integer ShowNormal = 1

     If  FindWindow( "Notepad - MYINFO.TXT" ) = 0 Then
          WinExec ( "notepad.exe MyInfo.txt", ShowNormal )
     End
end
```

Fix() Function

Purpose
Truncates the fractional part of a real number.

Syntax
`Fix(` *`real x`* **`)`** `returns integer`

Remarks
This function converts a given real number into an integer by ignoring the numbers to the left of the decimal point.

Example

```
procedure Main ()
     integer k = Fix ( 2.6 )      { k = 2 }
     integer j = Round ( 2.6 )    { j = 3 }
end
```

GetChar() Function

Purpose
Waits for a character from the communications device.

Syntax
`GetChar( int TimeOut ) returns character`

Remarks
The function will return either when it times out, or any character is received from the communications device. *TimeOut* is in milliseconds, and zero means no timeout (wait forever). If the function times out, it returns the zero character.

Example

```
Procedure Main ()
     Character x = GetChar ( 0 )

     If x = 'a' Then
          MessageBox ( "'A' received!", "Input", 0 )
     Else
          If x = 'b' Then
               MessageBox ( "'B' received!", "Input", 0 )
          Else
               MessageBox ( "Unknown received!", "Input", 0 )
          End
     End

End
```

GetConnectionState() Function

Purpose
Checks the current status of the communications device.

Syntax
`GetConnectionState () returns integer`

Remarks
If no connection is currently active, GetConnectionState returns zero. An active connection returns one.

Example

```
Procedure Main ()
     If GetConnectionState () = 0 Then
          MessageBox ( "No session connected","Session", 0 )
     Else
          MessageBox ( "Currently connected", "Session", 0 )
     End
End
```

GetFilter() Function

Purpose
Retrieves the current escape-sequence filter setting.

Syntax
`GetFilter()` `returns integer`

Remarks
Escape-sequence filtering determines if a script can "see" special control sequences reserved for the current terminal emulator (e.g. ANSI escape sequences).

If the filtering is on (non-zero) all script commands that process data from the terminal or the current device will not receive any of these control sequences. If it is off (the default), all characters will be accessible.

Example
```
procedure Main ()
     integer OldFilter = GetFilter()
end
```

GetLine() Function

Purpose
Waits for a line of text from the communications device.

Syntax
GetLine(*int TimeOut* **)** returns string

Remarks
The function will return either when it times out, or the ASCII codes carriage return or linefeed are received. These codes usually separate lines. *TimeOut* is in milliseconds, and zero means no timeout (wait forever). The return string may be blank (when the function times out).

Example
```
Procedure Main ()
      Integer DoneFlag

      String Input = GetLine ( 0 )
      If Left ( Input, 3 ) = "END" Then
            DoneFlag = 1
      Else
            DoneFlag = 0
      End
End
```

GetPassThrough() Function

Purpose
Checks if a device is passing its input to a terminal.

Syntax

GetPassThrough() `returns integer`

Remarks

GetPassThrough() returns the current passthrough setting. When passthrough is set (or non-zero), any string coming from the communications device is relayed through the script to the terminal window, and any input from the terminal will be passed to the communications device. When it is clear (or zero), all input and output between the communications device and the terminal window is blocked.

Example

```
procedure Main ()
     integer P = GetPassThrough()
end
```

IntToString() Function

Purpose

Converts an integer into a string.

Syntax

IntToString(*integer Value* **)** `returns String`

Remarks

This function converts a given integer value into the equivalent string value. Negative numbers will convert to strings that begin with the minus character '-'.

Example

```
procedure Main ()
```

```
     string x = IntToString( 20 )
end
```

Left() Function

Purpose
Returns the leftmost part of a string.

Syntax
`Left( string s, integer Length ) returns String`

Remarks
Returns the leftmost *Length* characters of a string.

Example
```
procedure Main ()
     string HW = "Hello There World"
     string H = Left ( HW, 5 )      { H = "Hello" now }
     string W = Right ( HW, 5 )      { W = "World" now }
     string T = Mid ( HW, 7, 5 )     { T = "There" now }
end
```

Len() Function

Purpose
Returns length of a string.

Syntax
```
Len( string Str ) returns integer
```

Remarks
The Len functions returns the number of characters in a given string.

Example
```
procedure Main ()
     integer Length = Len( "Hello World" )      { Length = 11 }
end
```

LoginName() Function

Purpose
Returns the current dialing directory login name.

Syntax
```
LoginName() returns string
```

Remarks
If there is no login name specified in the dialing directory, or the terminal was not opened from a dialing directory entry, then this string is empty.

Example
```
procedure Main ()
     WaitFor ( "user-id", 0, 0 )
     Send ( LoginName () )
     WaitFor ( "password", 0,  0 )
     Send ( Password () )
end
```

MessageBox() Function

Purpose
Creates and manages a window with a user defined message and title. The window can have any combination of predefined icons and buttons.

Syntax
`MessageBox(` *`string Caption`*`,` *`string Title`*`,` *`integer Options`* **`)`** `returns integer`

Remarks
The layout of the message box depends on the Options parameter. Add values from the following list to determine how the window looks:

Priority:
0   - Message box is application modal.
4096 - Message box is system modal.
8192 - Message box is task modal.

Buttons:
0   - Window will have the "OK" button.
1   - Window will have the "OK" and "Cancel" buttons.
2   - Window will have the "Abort", "Retry", and "Ignore" buttons.
3   - Window will have the "Yes", "No" and "Cancel" buttons.
4   - Window will have the "Yes" and "No" buttons.
5   - Window will have the "Retry" and "Cancel" buttons.
256 - Second button should be the default.
512 - Third button should be the default.

Icons:
16 - Stop-sign icon.
32 - Question mark icon.
48 - Exclamation mark icon.
64 - Information icon.

Return Values:
0 - Could not create message box.
1 - "OK" button selected.
2 - "Cancel" button selected.

3 - "Abort" button selected.
4 - "Retry" button selected.
5 - "Ignore" button selected.
6 - "Yes" button selected.
7 - "No" button selected.

For example, to define a message box with the title "Error", the message "Do you want to continue?", a "Yes" button, a "No" button, and a question mark, use:

```
MessageBox ( "Do you want to continue?", "Error", 4 + 32 )
```

The script will stop executing while waiting for a response to a message box.

Example

This example program opens a message box asking the user if he wishes to continue with some action. The box will contain the "Yes" and "No" buttons, and a question mark icon.

```
constant integer MB_YESNO = 4
constant integer MB_ICONQUESTION = 32
constant integer ID_YES = 6, ID_NO = 7

procedure Main ()
      integer WillContinue

      integer Result =
        MessageBox("Do you want to continue?",
                    "Question",
                     MB_YESNO + MB_ICONQUESTION )

      if Result = ID_YES then     { Did user select "Yes" button? }
            WillContinue = 1
      else if Result = ID_NO then  { Did user select "No" button? }
```

```
          WillContinue = 0
     end

end
```

Mid() Function

Purpose
Returns part of a string.

Syntax
```
Mid( string s, integer Start, integer Length ) returns String
```

Remarks
Returns *Length* characters of a string starting at character position *Start*. Character positions start from one and go up to the length of the string.

Example
```
procedure Main ()
     string HW = "Hello There World"
     string H = Left ( HW, 5 )      { H = "Hello" now }
     string W = Right ( HW, 5 )      { W = "World" now }
     string T = Mid ( HW, 7, 5 )      { T = "There" now }
end
```

Password() Function

Purpose

Returns the current dialing directory password.

Syntax

`Password()` `returns string`

Remarks

If there is no password specified in the dialing directory, or the terminal was not opened from a dialing directory entry, then this string is empty.

Example
```
procedure Main ()
     WaitFor ( "user-id", 0, 0 )
     Send ( LoginName () )
     WaitFor ( "password", 0,  0 )
     Send ( Password () )
end
```

RealToString() Function

Purpose
Converts a real number into a string.

Syntax
```
RealToString( real Value ) returns String
```

Remarks
This function converts a real value into the equivalent string.

Example
```
procedure Main ()
     string x = RealToString( 1.234 )
end
```

ResetTerm() Function

Purpose

Resets the terminal.

Syntax
`ResetTerm ()`

Remarks
The exact effect of this command depends on the terminal emulation in use.

Example
```
procedure Main ()
      ResetTerm ()
end
```

Right() Function

Purpose
Returns the rightmost part of a string.

Syntax

Right(*string s, integer Length* **)** returns String

Remarks

Returns the rightmost *Length* characters of a string.

Example

```
procedure Main ()
     string HW = "Hello There World"
     string H = Left ( HW, 5 )        { H = "Hello" now }
     string W = Right ( HW, 5 )      { W = "World" now }
     string T = Mid ( HW, 7, 5 )      { T = "There" now }
end
```

Rnd() Function

Purpose

Returns a random number between zero and one.

Syntax

Rnd() returns real

Remarks

This function returns a random number greater than or equal to zero and less than one.

Example

```
procedure Main ()
     integer j = Fix ( Rnd () * 6 )    { number between 0 and 6 }
```

```
end
```

Round() Function

Purpose
Rounds a real number to the nearest integer.

Syntax
`Round(` *`real x`* **`)`** `returns integer`

Remarks
The Round function converts a given real number into an integer by rounding up or down the fractional portion of a number.

Example
```
procedure Main ()
     integer k = Fix ( 2.6 )      { k = 2 }
     integer j = Round ( 2.6 )    { j = 3 }
end
```

Send() Function

Purpose
Sends a string to a communications device.

Syntax
`Send(` *`string SendString`* **`)`**

Remarks
The Send function relays a given string to the communications device, e.g. the modem.

Example
```
{This example demonstrates basic input processing}

Procedure Main ()
  While ( 1 )                           { Repeat forever }
     WaitFor ( "Bob is paging you", 0, 0 )
     Send( "/p Bob Not now, I'm busy." + Char ( 13 ) )
  End
End
```

SendChar() Function

Purpose
Sends a character to a communications device.

Syntax
SendChar(*char C* **)**

Remarks
This function relays a specified character to the communications device, e.g. the modem.

Example
```
procedure Main ()
     SendChar ( 'x' )
end
```

SetDuplex() Function

Purpose
Sets a terminal to full or half duplex.

Syntax
`SetDuplex(` *`integer DuplexMode`* **`)`** `returns integer`

Remarks
The SetDuplex function sets the duplex mode. *DuplexMode* should be zero for half duplex or one for full duplex. It returns the previous duplex setting.

Example
```
Constant Integer Half = 0, Full = 1

Procedure Main ()
     Integer OldDuplexMode = SetDuplex( Full )
     Send( "Hello" + Char ( 13 ) )
     SetDuplex( OldDuplexMode )
End
```

SetFilter() Function

Purpose
Turns the escape-sequence filtering on or off in a script.

Syntax
```
SetFilter( integer Flag ) returns integer
```

Remarks
Escape-sequence filtering determines if a script can "see" special control sequences reserved for the current terminal emulator (e.g. ANSI escape sequences).

If the filtering is on (non-zero) all script commands that process data from the terminal or the current device will not receive any of these control sequences. If it is off (the default), all characters will be accessible.

Example
```
procedure Main ()
     constant integer True = 1
     SetFilter( True )
end
```

SetPassThrough() Function

Purpose
Allows a device to pass its input to a terminal.

Syntax
```
SetPassThrough( integer OnOff ) returns integer
```

Remarks
SetPassThrough returns the current passthrough setting. When passthrough is set (or non-zero), any string coming from the communications device is relayed through the script to the terminal window, and any input from the terminal will be passed to the communications device. When it is clear (or zero), all input and output between the communications device and the terminal window is blocked.

SetPassThrough returns the previous passthrough setting.

Example
```
procedure Main ()
     constant integer True = 1

     SetPassThrough ( True )
end
```

Sqrt() Function

Purpose
Calculates the square root of a number.

Syntax
```
Sqrt( real Value ) returns real
```

Remarks
This function returns the square root of a given number.

Example
```
procedure Main ()
     real root = Sqrt( 16 )     { root = 4 }
end
```

StartCapture() Function

Purpose
Starts capturing the incoming datastream to a file.

Syntax
```
StartCapture( string Filename ) returns integer
```

Remarks
If there was a problem creating the given file, StartCapture will return a zero. If the operation was a success, StartCapture returns one.

Example

```
procedure Main ()
     StartCapture ( "myfile.cap" )
end
```

StopCapture() Function

Purpose
Stops capturing the incoming data stream.

Syntax
`StopCapture ()`

Remarks
Stops capturing any incoming data and closes the capture file.

Example
```
procedure Main ()
     StopCapture ()
end
```

StringToInt() Function

Purpose
Converts a string into an integer.

Syntax
`StringToInt(` *`String Value`* **`)`** `returns Integer`

Remarks
This function converts a given string value into the equivalent integer value. An invalid number will return zero.

Example

```
procedure Main ()
     integer X = StringToInt ( "10" )
     integer Y = StringToInt ( "&10" )
end
```

StringToReal() Function

Purpose
Converts a string into a real number.

Syntax
StringToReal(*String Value* **)** returns Real

Remarks
This function converts a given string value into the equivalent real number value. An invalid number will return zero. The string should have the following format: `"<Sign><Number>.<Number>"`. Sign is either plus '+', minus '-' or nothing, and the decimal point is optional.

For example, "-1.432", "18" and "+23.32" are valid strings. "Hello", "123abc" and "2e-12" are not and will return zero.

Example
```
procedure Main ()
     integer X = StringToReal( "10.293" )
end
```

SysName() Function

Purpose

Returns the current dialing directory system name of the terminal window associated with the script.

Syntax
`SysName()` `returns string`

Remarks
If there is no system name specified in the dialing directory, or the terminal was not opened from a dialing directory entry, then this string is empty.

Example
```
procedure Main ()
     MessageBox ( "Currently on system " + SysName (),
                  "Logged in", 0 )
end
```

Time() Function

Purpose
Returns the current time in a specified format.

Syntax
`Time(` *`string Format`* **`)`** `returns string`

Remarks
Provides the current system time. The string *Format* determines how the result is formed. An *h*, or group of *h*'s will be substituted with the current hour.   Similarly, *m*'s are minutes, and *s*'s are seconds. For example, `"hh:mm:ss"` would return `03:28:21` if the time happened to be 28 minutes past 3 in the morning.

Example
```
procedure Main ()
     string TimeString = Time ( "hh:mm:ss" )
     string DateString = Date ( "yyyy-mm-dd" )

     MessageBox ( "It is now " +
                  TimeString + ", " +
                  DateString + ".", "Time", 0 )
end
```

ToLower() Function

Purpose
Converts a string to lowercase.

Syntax
`ToLower( string Caption ) return string`

Remarks
The ToLower function converts all characters in the given string to lowercase. For example, 'A' becomes 'a', 'B' becomes 'b', and so on. Only the letters in the string are affected by this function.

Example
```
{Converts "UserName:" to "username:"}

procedure Main ()
     string Name = "UserName:"
     Name = ToLower ( Name )
     MessageBox ( Name, "Case conversion", 0 )
end
```

ToUpper() Function

Purpose
Converts a string to uppercase.

Syntax
```
ToUpper( string Caption ) to string
```

Remarks
The ToUpper function converts all characters in the given string to uppercase. For example, 'a' becomes 'A', 'b' becomes 'B', and so on. Only the letters in the string are affected by this function.

Example
```
{Converts "UserName:" to "USERNAME:"}

procedure Main ()
      string Name = "UserName:"

      Name = ToUpper ( Name )
      MessageBox ( Name, "Case conversion", 0 )
end
```

Upload() Function

Purpose
Uploads a file using a specified protocol.

Syntax
```
Upload( string Filename, string Protocol ) returns integer
```

Remarks
Upload returns one if the upload succeeds and zero if it fails. The protocol must match one of the protocol names, as defined in WinRamp's protocol configuration. If the protocol does not exist, the upload will fail. Similarily, if the file does not exist, the upload will fail.

Example
```
procedure Main ()
      Upload ("newgame.arj", "ZModem" )
end
```

WaitFor() Function

Purpose

Waits for a string from a communications device.

Syntax

`WaitFor(` *`string SendString, integer TimeOut, integer CaseSensitive`* **`)`** `returns integer`

Remarks

If the string is not received in the specified *TimeOut* period (measured in milliseconds) then WaitFor returns zero. If the string is received, it returns a one.

A Timeout value of 0 counts as "no timeout."

If CaseSensitive is 1, only exact characters match. If it is 0, the match is case insensitive, so, for example "a" = "A".

Example
```
{This example demonstrates basic input processing}

Procedure Main ()
  While ( 1 )                              { Repeat forever }
     WaitFor( "Bob is paging you", 0, 0 )
     Send ( "/p Bob Not now, I'm busy." + Char ( 13 ) )
  End
End
```

WaitForConnection() Function

Purpose
Puts the device into auto-answer mode and waits for an incoming connection to be established.

Syntax
`WaitForConnection( int TimeOut ) returns integer`

Remarks
TimeOut is in milliseconds, zero means no timeout (wait forever). The function returns zero for failure (i.e. timed out or unable to establish session) or one if a successful connection was established.

Example
```
Procedure Main ()
      Integer DidConnect = WaitForConnection( 0 )
      If DidConnect = 0 Then
            MessageBox( "Can't establish connection", "Session", 0 )
      End
```

```
End
```

WinExec() Function

Purpose
Runs a program.

Syntax
`WinExec(` *`string Prog, integer State`* **`)`** `returns integer`

Remarks
Launches a specified application. *Prog* should be the program executable name, complete with path (if necessary) and any parameters. The *State* parameter determines the applications initial state:

0	Hides the window, another application is activated.
1	Normal window
2	Minimized (start as icon)
3	Maximized

If the program can be successfully run, the function returns a value greater than 32. Otherwise, it returns one of the following values:

0	System out of memory, or executable file corrupt, or relocations invalid.
2	File not found.
3	Path not found.
5	Sharing or network-protection error.
6	Library required separate data segments for each task.
8	Not enough memory.
10	Incorrect Windows version.
11	Program not a Windows application, or file corrupt.
12	Incorrect operating system for this application.

13	MS-DOS 4.0 application.
14	Unknown file type
15	Application developed for an earlier version of Windows.
16	Attempt to load a the application more than once when this is illegal.
19	Executable file compressed.
20	A Dynamic Link Library (DLL) required to run the program is corrupt.
21	Application cannot run without 32 bit extensions.

Example

```
procedure Main ()
     constant integer ShowNormal = 1

     If  FindWindow( "Notepad - MYINFO.TXT" ) = 0 Then
          WinExec( "notepad.exe MyInfo.txt", ShowNormal )
     End
end
```

5.4 WR-Script Control Structures

Control structures are key elements in building flexible scripts. There are three control structures in WR-Script:

FOR .. TO .. NEXT Control Structure

Purpose
Carries out a series of instructions a specific number of times.

Syntax

```
FOR Variable = Start TO End [STEP Increment]
```

```
  Series of Instructions
NEXT [Variable]
```

Remarks

The series of instructions will be performed as many times as it takes the variable to increment from *Start* to *End*. The variable is incremented by the number specified in *Increment* (if there is no step increment stated, the increment is taken as one) each time the `NEXT` statement is carried out. If *Start* is greater than *End*, then *Increment* must be a negative value so that the variable decreases (by the step increment) until it is less than *End*.

`FOR..TO..NEXT` control structures can be nested within each other, as long as the variable is unique for each structure.

Example

```
procedure Main ()
      integer LoopCount
      For LoopCount = 32 To 127 Step 2
            SendChar ( Char ( LoopCount ) )
      Next LoopCount
end
```

IF .. THEN .. ELSE Control Structure

Purpose

Runs instructions conditionally.

Syntax

```
IF condition THEN series of instructions
[ELSE series of instructions] END
```

Remarks

A condition is an expression that is either true or false. As in most basic-like languages, true is evaluated as some non-zero number and false as zero. When the `IF` condition is true, any instructions after the `THEN` are executed. When the condition is false, any instructions after the `ELSE` are executed. The simplest form of this control structure is `IF` *`condition`* `THEN` *`instruction`* `END`, where the instruction is performed if the condition is true.

Example

```
procedure Main ()
      If WaitFor ( "password", 10000, 0 ) Then
            MessageBox ( "Waitfor timed out", "Error", 0 )
      Else
            MessageBox ( "Waitfor successful", "Success", 0 )
      End
end
```

WHILE .. END Control Structure

Purpose

A loop that will execute a series of instructions while a sepecific condition is true.

Syntax

```
WHILE condition
  Series of instructions
END
```

Remarks

The series of instructions will be repeated as long as the condition holds true. If the condition always holds true an endless loop occurs.

Example

```
{This example demonstrates basic input processing}

Procedure Main ()
  While ( 1 )                             { Repeat forever }
     WaitFor ( "Bob is paging you", 0, 0 )
     Send ( "/p Bob Not now, I'm busy." + Char ( 13 ) )
  End
End
```

6. RUNNING SCRIPTS

There are a number of ways to run a script:

- The script can be loaded manually, after you have established a connection.
- You can attach a script to a button and click on it to run the script.
- A script can be attached to a particular dialing directory entry, in which case the script is run whenever that entry is used to dial a remote system.

6.1 Loading a Script

Establish a communication session with your desired host. Select Run Script Object from the Scripts menu OR click the Run Script Object button. You will be presented with the Select Script File dialog box, from which the desired script file must be selected.

6.2 Attaching a Script file to a Button

If a script is used frequently, it may be a good idea to attach a button to it, which will allow you to execute it with a single mouse click. Once you have created a script file either by autolearning it or with the **WR-Script IDE**, you can associate it with a particular button which you can then add to a toolbar. To do this, select the Attach Tool Buttons option from the Scripts menu and then select the Scripts option from the secondary menu. Choose Add to attach a button to a script file that has no buttons attached to it or choose Edit to change the attached button. The Remove option detaches a button from a script file.

In the dialog box that appears, type the name of your script in the Name field, the text you would like to appear as the tool tip for that button in the Tool Tip field, and the path and file name of the script file you would like to attach in the File field. It is then a case of selecting the button you would like to attach to the script file.

Although the script file now has a button attached to it, you will only be able to see and click it if it is configured in a toolbar. To do this, select Toolbars from the Configure menu or click the Toolbar Editor button in the configuration toolbar. Select Edit from the secondary menu and then click the New button at the top of the Toolbar editor window. Enter the name of your new toolbar in the toolbar Name field. After pressing Enter or clicking the OK button, make sure the check box for Scripts is marked (under the Show section of the dialog box.), then select the script file and attached button you would like on the new toolbar in the Available Actions list. Drag it to the Toolbar Buttons list or click the Arrow button between the two lists. This process can be repeated for as many buttons as you would like on the toolbar. Click the OK button once you have finished and the new buttons should now appear in the new toolbar in your *WinRamp Lite* window.

To run a script that is attached to a button, first establish a connection with your desired host. Clicking the button you have attached to the script will cause the script to execute.

6.3 Attaching a Script File to a Dialing Directory Entry

You can create an automatic logon by writing a script using the **WR-Script IDE** or recording it with the WR-Script *AutoLearn* facility and performing the desired actions. In order to attach a script to a dialing directory entry, do the following:

- Ensure that the number, for which you would like a script to execute automatically, is in the Dialing Directory. If not, you will have to add an entry in the Dialing Directory for it.
- Open the Dialing Directory by selecting Dialing Directory from the Communicate menu or by clicking the Dialing Directory button.
- Select the appropriate entry.
- Click the Edit Entry button, select Edit from the Dialing Directory menu or press the space bar to edit the settings.
- Click the Directory tab at the top of the Directory Entry Details dialog box.
- Click the Script radio button under the On Startup section of the dialog box.
- Make sure that the Scripts default path on the top left of the window is the one that your script file is in. If not, change the path so that it is correct.
- Type the name of the script file without the file extension in the field next to the Script radio button.
- Click the OK button.

Whenever you click on that entry from the Dialing directory, it will dial the relevant host, using the script file on start up.

7. STOPPING A SCRIPT

Once the desired script is running in your communication session, you may want to stop it from completing the series of instructions contained in the script file by selecting the Stop Script from the Scripts menu or click the Stop Scripts button when you want to break the execution.