Contents





SALT Language Overview



SALT Syntax



Script Structure



Compiler Preprocessor



Built-In Functions



System Variables



ASCII Character Set



Extended Key Scan Codes



Color Values



SIMPLE Language



SALT Editor

SALT II Overview

The Telix for Windows SALT II Language

Telix for Windows has a built-in programming language called SALT II (Script Application Language for Telix version II). SALT II is based on the powerful SALT script language used in Telix for DOS, and it will allow you to perform almost any communications related applications within Telix. SALT II is designed to be as compatible with SALT as can be accomplished when changing operating systems. SALT II looks similar to the C language, however if you have used any programming language (such as Pascal, BASIC, etc.), you should feel quite at home with SALT II. While SALT II was designed to be easy to learn, it is quite complete, so it is recommended that you read the chapters of the Telix for Windows manual dealing with SALT II. Hereafter in this help file, SALT II will be referred to as SALT.

What Can be Accomplished With SALT?

A SALT script is basically a sequence of instructions for Telix to follow, using a specific syntax. You may use any text editor to produce this script file, as long as its output is normal ASCII text (this means that if you use your word processor, you must usually explicitly tell it to write out the file using ASCII format and to not embed any special codes in the file). You may give any name you wish to a SALT source file, although we recommend that you always use the extension .SLT for clarity. Telix for Windows provides an integrated script editor (hereafter referred to as the SALT Editor) that has many features designed to aid you in developing scripts. It is recommended that you use this editor to develop your scripts.

Creating SALT Programs

Like a program in any programming language, a SALT program (also called a 'script') is typically used to perform a needed task or function. The task can range from the very simple to the very complicated. For example, a SALT script can be linked to a dialing directory entry, so that when you have established a connection to that service, it automatically sends your user name and password to the remote service. A much more complicated SALT script is used as the basis for the Host Mode included with Telix.

Once you have written your script file and saved it to disk, it must be compiled. Using the SALT Editor, you need only to select the Compile command from the menu, or if using an external editor, select Compile from the Telix for Windows Script menu. The compiler then reads your 'source' script file, and compiles it to a form which Telix can understand. The compiled script can then be loaded more quickly by Telix, and is also usually smaller. The output file is written with the same name as the source file except that the extension .SLC is used.

When the script compiler finds an error in your source file, it will abort the compile process and give you the line number on which the error occurred, as well as the type of error. If you are using the SALT Editor, the cursor will be moved to the line that the error occurred on and the error message will be displayed on the status bar. The error should then be fixed and the source re-compiled. This is repeated until the compiler detects no more errors in your source file. The compiled script can then be executed in Telix using one or all of several methods; using the Execute menu item on the Script menu, as a linked script in a dialing directory entry, or called from another script.

SALT Syntax

SALT Case & Item Placement

Case is not important in command, function, preprocessor directives or variable names. The only time case matters is inside a string constant (e.g., "Hello" is not the same string as "hello"). Whitespace (such as the space, the tab, the Carriage Return, or the Line Feed character) is not important. The script compiler does not care where you place items, so you may arrange the program as you see fit. For example,

The only time whitespace matters is when it would split up keywords or function names, or in a string. For example, the keyword **while** must not be split up if it is to be recognized. The same applies to other keywords or function names. Also, there must be space between the letters of a command and other letters. For example, "**while** abc" is not the same as "whileabc". In the interest of clarity, it is recommended that you try to make your code easy to understand, by indenting where appropriate, and by using space effectively. There is no reason, for example, to put more than one statement on a line, even if it is perfectly legal. A good example of program style can be found by looking at the sample scripts included with Telix for Windows.

String Constants

A string constant is a sequence of ASCII characters enclosed in quotes, for example, "Hello", "Goodbye", or "Telix". It is often necessary for a string constant to include special characters that can not easily be typed from the keyboard, or can not be easily displayed. This is done with something called the escape character, which is the caret (^) symbol. When the SALT compiler is reading a string constant and comes to the ^ symbol, it replaces it with a certain ASCII code based on the character following the ^. Translations are as follows:

- ^c Where c is any letter, the Control representation of c is inserted into the text. Therefore ^M represents Ctrl-M, ^j represents Ctrl-J, etc. Whether the letter c is upper or lower case is not significant. Note that what is really happening here is that 64 is being subtracted from the ASCII value of c, so for example, the Escape character can be represented as ^[.
- ^^ An actual caret (^) symbol is placed into the text.
- ^" An actual double quote symbol (") is placed into the text. Strings are always delimited by the double quote symbol, so one cannot be readily embedded in a string. Using the ^" control

representation will allow the inclusion of double quote characters in strings. If the plain " symbol were to be used, the compiler would think that the string was terminated at that point. For example, the string "He said, ^"Hello^"." is translated to 'He said, "Hello".'.

An actual single quote symbol (') is placed into the text.

^nnn Where nnn is up to 3 digits representing the ASCII value of the character which should be placed into the text. A maximum of three digits is read, or up to the first non-digit character. For example, the compiler would read in the string "S^65LT" and interpret it as the string "SALT", since 65 is the ASCII value of 'A'. Note that if nnn is less than 3 digits you may have to pad it with one or two leading zeros if there are digits immediately following it in the string, so that the wrong value is not read in. For example the string "^79 Park Avenue" would translate to "O Park Avenue" since 79 is the ASCII value of 'O'. If you actually wanted Ctrl-G (ASCII code 7) followed by "9 Park Avenue", you would use the string "^0079 Park Avenue".

When this character is sent to a modem device, it will cause it to pause for 1/2 a second. This character is cumulative, so sending multiple will increase the delay length.

Integer Constants

An integer constant is a sequence of digits representing an integer value in the range of -2,147,483,648 to 2,147,483,647. An integer constant must start with a digit from 0 to 9 or the negative sign (-) followed by a digit. The following are all valid integer constants:

```
10
-400067
999
```

An integer constant may also be entered in hexadecimal form (base 16, where each digit may be from '0' to '9' or 'a' to 'f', to represent 16 values). Hex values must be preceded by 0x for the compiler to interpret them as such, and case is not important. The following are all valid integer constants enter in hexadecimal form:

```
0xff00
0Xa2
0x7D
0x1AbCdEf
```

Comments

A comment in a source file is text that does not affect what the program does, and is meant purely for explaining or describing something. In a SALT source file, whenever the symbol // is encountered on a line, all the characters from that point on until the end of the line are considered to be a comment and are ignored. For example:

New to SALT II to is another method of commenting, the symbols /* and */. The first symbol indicates the start of a comment and the second indicates the end of it. Using this method of commenting will allow the comment to continue across lines, unlike the // method. This is used for large comment sections and often for temporarily removing code from a script. For example:

```
/* This comment is three lines long,
```

```
but we need only two markers to
indicate its beginning and end. */

prints("Start of script.");
/* prints("These lines would");
prints("neither be compiled,");
prints("nor executed."); */
prints("End of script.");
```

SALT Structure

A SALT script has the following format:

and so on. Basically, a script file consists of definitions of global variables (variables which are available to any part of the script file after which they are defined, and function definitions (functions are lines of code clustered together in a group, so that they can be called by a name). A script file does not have to have any global variables or functions, but to run it must at least have one function called 'main'. The following, for example, is a complete script file:

```
main()
  {
   prints ("hello");
  }
```

When compiled and executed, this script would print the string "hello" to the screen.

See also

Variables, Expressions and Operators, Functions, Statements

Variables

A variable is a location in memory where something is stored. The contents of a variable can be changed by program code (hence the name). In SALT, there are two types of variables, integer variables, and string variables. The former holds an integer value (e.g., 485624, or -627), while the latter holds a text string (e.g. "Telix", or "SCRIPT"). Depending on where it is defined, a variable is either global or local. If a variable is global, it means that it can be used by any part of the script after the point where it is defined. If a variable is local, it means that it can only be used by the part of the script to which it is *local*, for example, the function inside which it is defined. A variable name can be up to 31 digits long, and may include the letters 'A' to 'Z' or 'a' to 'z', the digits '0' to '9', or the underscore character (_). The name may not start with a digit. For example, 'his_name2' and '_his_name2' are legal as variable names, while '2his name' is not.

An integer variable is defined in the form

```
int <varname>;
```

where <varname> is the name to be given to the variable. An alternate definition is

```
int <varname1>, <varname2>, ..., <varnameN>;
```

which allows you to define more than one integer variable in one statement. An original value can be assigned to the integer variable by using the form

```
int <varname> = <int const>;
```

where <int_const> is an integer constant. Similarly, an original value can be assigned in the multiple definition above by placing the assignment before the comma. Some examples are:

```
int maximum;
int start = 0;
int level, i, count = 20, loop;
```

A string variable is defined in the form

```
str <varname>[<max>];
```

where <*varname*> is the name to be given to the variable. <*max*> is the maximum number of characters that the string can hold, and must be in the range of 0 to 32767. An alternate definition is

```
str <varname>[<max>], <varname2>[<max>], ..., <varnameN>[<max>];
```

which allows you to define more than one string variable in a statement. An original value can be assigned to the string variable by using the form

```
str <varname>[<max>] = <str const>;
```

where *<str_const>* is a string constant. Similarly, an original value can be assigned in the multiple definition above by placing the assignment before the comma. Some examples are:

```
str password[80];
str password[40] = "mypass", name[30];
```

The string length field may be left empty if an original value is specified, in which case the length of the

string variable is assumed to be that of the assigned text. For example:

```
str name[] = "John";
name = "Matthew";
```

would declare a variable called name with a maximum length of four (five counting the 0 (NULL) terminator) and an initial value of "John". On the second line, it is assigned a new value, but because the maximum length is only four, the variable now contains the string "Matt", not "Matthew" as it might appear to the unsuspecting.

If a variable is outside of a function, it is global. If it is defined inside a function, it is local to that function and will only be recognized there. If a variable defined inside a function uses the same name as a global variable, any reference to that name while in the function will access the local variable. After the function has completed, the local variable is removed and a reference to that name will access the global variable.

Expression and Operators

An expression is a mixture of symbols which resolves to a value when evaluated. In other words, an expression is basically a formula. An expression can consist of constants, variables, function calls, and operators. An expression can be very simple, or very complicated. For example, some expressions are:

```
10 + 3 - 5

9 * 7 / 63 - 30

result = 10 * max(a, b)

month >= 10

200

command == "bye"

prints("Hello")
```

In an expression, the data being acted upon are constants, variables, and functions calls, while the operators (+, *, etc.) are the symbols that do things with the data. There are many different operators, of which there are two basic types. Binary operators (such as +, *, /) perform a calculation on the expression on either side of them. Unary operators appear before a single expression and work on that. The following table lists the operators available in SALT:

Symbol	(Un/Bin)ary	What it is/does
-	unary	Arithmetic negation
!	unary	Logical NOT
not	unary	Logical NOT (alternate)
++	unary	Increment
	unary	Decrement
*	binary	Multiplication
/	binary	Division
%	binary	Remainder (Mod)
+	binary	Addition
_	binary	Subtraction
<	binary	Less than
>	binary	Grater than
<=	binary	Less than or equal to
>=	binary	Greater than or equal to
==	binary	Equality
!=	binary	Inequality
&	binary	Bitwise AND
1	binary	Bitwise OR
۸	binary	Bitwise Exclusive OR
&&	binary	Logical AND
and	binary	Logical AND (alternate)
II	binary	Logical OR
or	binary	Logical OR (alternate)
=	binary	Assignment
+=	binary	Addition and Assignment
-=	binary	Subtraction and Assignment
*=	binary	Multiplication and Assignment
/=	binary	Division and Assignment

Note that the hyphen symbol (-) can be either an arithmetic negation or a subtraction depending on its use. Note that '!' is equivalent to 'not', '&&' is equivalent to 'and', and '||' is equivalent to 'or'. The first form is preferred as you do not have to leave whitespace around it for the compiler to recognize it, but beginners may have an easier time remembering the second form. Also, do not confuse '=' (the

assignment operator) with '==' (the equality operator). The former is used to assign a value to a variable, while the latter is used to compare two values. Assuming you have the two expressions, <expr1> and <expr2>, <expr1> = <expr2> would assign one to the other, while <expr1> == <expr2> would test the two to see if they are equal. For example

```
num = 10;
```

would assign the value 10 to the variable called 'num', while

```
num == 10
```

would resolve to a value of non-zero (TRUE) if num was equal to 10, and 0 (FALSE) if num was not equal to 10. There is also a difference between the Logical operators and the Bitwise operators. The Logical operators (such as and, &&, or, ||, etc), work with TRUE or FALSE values and result in a TRUE or FALSE value, while the Bitwise operators (&, |, $^{\circ}$) work with the actual bits of the data they are handling. The Bitwise operators almost never have to be used in a Telix script, unless it is needed to get at the actual bits in a data byte.

Every operator resolves to a value, which is the result of the operation performed (e.g, 10 * 7 would resolve to 70). The conditional or equality operators such as ==, >, <=, etc., resolve to a 0 (FALSE)) or non-zero (TRUE) value based on the results of the expression. Even the assignment operator = resolves to a value. The result of the expression

```
num = 10;
```

would be 10.

All the operators have something called precedence, which is their importance, and determines the order in which they are evaluated. For example, 7 + 3 * 9 is equal to 34, because 3 * 9 is evaluated first, and then added to 7 (* has a higher precedence than +). All the operators are listed below in order of decreasing precedence. All the operators on the same line have the same precedence, and are resolved in the order that they are encountered.

```
- !
++ --
* / %
+ -
< > <= >=
== !=
&
|
and &&
or ||
```

If a certain evaluation order is required that does not follow these rules of precedence, parentheses may be used. Thus, 99 + 1 * 10 equals 109, while (99 + 1) * 10 equals 1000.

If you are writing an expression of any sort, and are not sure of the exact precedence of the operators you are using, **use parentheses!**

The operators +=, -=, *=, /= are new to SALT in Telix for Windows. They are a shorthand method of performing basic math functions on a single variable. To illustrate their use, the following examples show the longhand and shorthand equivalents:

```
int i;
```

```
i = 5;

i = i + 3;  // longhand. i == 8 now.
i += 3;   // shorthand. i == 11 now, NOT 3.
i = i - 5;  // longhand. i == 6.
i -= 2;   // shorthand. i == 4.
i = i * 3;  // longhand. i == 12.
i *= 2;   // shorthand. i == 24.
i = i / 6;  // longhand. i == 4.
i /= 4;   // shorthand. i == 1;
```

Functions

A function is a way of grouping together some lines of code. A Telix script consists of one or more functions. There are quite a few advantages to using functions:

- One function can be called from another, to do a certain task. The calling function does not have to know anything about the called function other than what it does. This allows a script to be split up into modular units, and makes code writing and debugging easier.
- As mentioned above, what a function does it private. This means that data variables defined in a function are local to that function, and therefore you do not have to worry about another part of the script unintentionally modifying them.



A library of functions can thus be built. Later, you do not have to rewrite old code.

Functions are defined in the following format:

```
<funcname>(<arg1>, <arg2>, ..., <argN>)
{
    <variable_def>
...
    <variable_def>
    <statement>
...
    <statement>
}
```

<function. It follows the same rules of other identifiers in SALT.</p>
There can only be one function that uses a given name, however.

<arg1> through <argN> are the declarations of the arguments (parameters) that have been passed to the function by its caller (sometimes, to accomplish its task, a function needs to have some values passed to it). Each argument is defined in the form <type> <name> where <type> is int or str, and <name> is the name it should be called by. At present, a function is not allowed to have more than 12 values passed to it.

<variable_def> is a variable definition, as described in the above section on that topic. Any number
of variables may be declared at this part of the function. All such variables will be local variables and
available only to this function.

<statement> is an actual line of code. There may be as many lines of statements in the function as needed. The format of a statement is described below. First though, here is an example of a complete function:

```
max ( int a, int b )
{
  int result;
  if (a > b)
    result = a;
```

```
else
   result = b;

return result;
}
```

This function returns the larger (maximum) of the two values passed to it. It could have been written much more simply (without the use of the variable), but was written this way so that all the function elements would be there.

Statements

A statement is the basic element of code. A statement **ALWAYS** ends with a semicolon character (;). In any location where a statement is acceptable, you may use a group of statements, by enclosing them all in curly braces ({ and }). There are many types of statements, including: expression, if, while, do...while, for, return, break, continue, goto and switch statements. Each type has several different parts.

See also

<u>Expression Statements</u>, <u>If Statements</u>, <u>While Statements</u>, <u>Do...While Statements</u>, <u>For Statements</u>, <u>Return Statements</u>, <u>Break Statements</u>, <u>Continue Statements</u>, <u>Goto Statements</u>, <u>Switch Statements</u>

Dead Parrot

Special Thanks go to Crawford Dales for the outstanding job of proofreading.

Expression Statement

The expression statement is the simplest and most common type of statement. Its format is

```
<expression>;
```

where <expression> is any expression. Examples are:

```
result = 20;
password = "Beef";
pause(20);
num = 20 * max(a, b);
```

Do not forget the semicolon character at the end of the statement. If you do, the compiler will think that the next statement is part of the current one, and will report some unexpected error.

If Statement

An if statement is used when a statement or group of statements should be evaluated only if a condition is true. The format for an if statement is as follows:

```
if (<expression>)
  <statement>
```

<statement> is any statement as described above and below (that is, an expression, if, while, do...while, for, return, break, switch or continue statement), and will only be executed if <expression> evaluates to non-zero. By using curly braces around them, a whole group of statements may be conditionally evaluated. Some examples are:

```
if (result == -1)
    prints("ERROR!");

if (num_tries > maximum)
    return 0;

if (month > 10 && day < 20)
    {
     clear();
     prints("In range.");
     return 1;
    }

if ((num < 10) && (!error) && (read != 0))
     prints("Continuing...");

An alternate form of the if statement is:

if (<expression>)
     <statement1>
    else
     <statement2>
```

In this case, if <expression> evaluates to non-zero (TRUE), <statement1> is executed, otherwise <statement2> is executed. Again, multiple statements may be used instead by grouping them in curly braces. Some examples are:

```
if (stat == -1)
  prints("Error status returned.");
else
  prints("Function finished without problems.");

if (level < 10)
{
  alarm(1);
  prints("Warning!");
}
else
  prints("Everything's ok.");</pre>
```

Since the statement to be executed conditionally can be of any type, that means that any number of if statement can be nested if needed. For example:

```
if (num < 10)
  if (!error)
   if (read != 0)
    return 1;</pre>
```

This also means that something like the following is legal:

```
if (value == 10)
   do_this();
else if (value == 100)
   do_that();
else if (value == 1000)
   do_something_else();
else
   do whatever();
```

What is really happening here is that each if statement is being nested after the else portion of the previous one. The above example could also be written as:

```
if (value == 10)
  do_this();
else
  if (value == 100)
    do_that();
else
    if (value == 1000)
       do_something_else();
    else
       do_whatever();
```

Any amount of nesting is theoretically legal, but the compiler does have a limit due to memory constraints.

While you may write the code in any way which suits you, it is recommended that you use indenting, for clarity. Indenting your code at the proper places makes it a lot easier to read. A very common error to watch out for is accidentally placing a semicolon after the parenthesis ending the expression. For example, if the following is run:

```
if (num == 10);
  prints("Num is equal to 10);
```

the string would always be printed, no matter what num was equal to. This is because the semicolon after the parenthesis ending the expression signifies the end of the statement. In the above case, it would just be a null (empty) statement.

WHILE Statement

The while statement is used to loop continuously while a certain condition is true. It has the form

```
while (<expression>)
  <statement>
```

<statement> would continue to be repeated over and over while <expression> evaluated to non-zero (TRUE). Note that if the expression evaluates to 0 (FALSE) from the beginning, the statement will never be executed. Again, multiple statements may be used by surrounding them in curly braces. A few examples are:

```
while (stat != -1)
    stat = myfunc();

while (num < 100)
{
    printn(num);
    prints("");
    num = num + 1;
}

while (1)
{
    if (func1())
    return 0;
    func2();
}</pre>
```

Again, be careful to not place a semicolon after the parenthesis ending the expression.

DO ... WHILE Statement

The do...while statement is similar to the while statement and has the form:

```
do
    <statement>
while (<expression>);
```

<statement> will be executed at least once and will continue to be executed repeatedly until the expression becomes 0 (FALSE). A few examples are:

```
do
    stat = func1();
while (stat != -1);

do
    {
    prints("hello");
    num = num + 1;
    }
while (num < 100);</pre>
```

FOR Statement

The for statement is used to loop continuously while a certain condition is true. The advantages over the while statement is that a control variable can be initialized and incremented quite easily. The for statement has the form:

```
for (<expression1>; <expression2>; <expression3>)
  <statement>
```

The first expression is the one that should initialize the count variable. For example, if you wanted to count from 1 to 100, and were keeping the count in a variable called num, the first expression would be num = 1. The second expression is the conditional test. As long as it evaluates to non-zero (TRUE), the statement will be executed. Following the above example, this expression would be num < 100. The third expression is the one that is used to increment the count variable. For the above example, it would therefore be num = num + 1. This for statement differs in format from that in most other languages, but doing it this way is actually gives the programmer a lot of power and flexibility. Note that any of the expressions can be left empty, in which case they evaluate to non-zero (TRUE). Some examples are:

```
for (count = 0; count < 100; count = count + 1)
{
   printn(count);
   prints("");
}

for (c = 1000; c > 0; c = c - 1)
   do_this(c);
```

The following would execute an infinite loop:

```
for (;;)
  prints("Hello!");
```

In some programming languages, the for statement is used to execute a loop for a specific number of iterations. In SALT however, you have more control over the condition that causes the loop to terminate. The conditional test is almost always related to the control variable, however any expression that will evaluate to a true or false value may be used. For example, the following is quite legal:

```
for (c = num = 0; ((c < 100) && (stat != -1)); c = c + 1)
{
  stat = func(num);
  num = func2();
}</pre>
```

The statements would only be executed if c was smaller than 100 and stat, which is not related to the control variable, didn't equal -1.

RETURN Statement

At some time, every function must be exited. If the end of the function is reached, control will automatically return to the calling function. Very often however, it is necessary to leave a function somewhere while only halfway through it, perhaps based on a conditional test. Also, it is often necessary that a function returns a value to the caller. The format of the return statement is:

```
return <expression>;
```

If the return statement is encountered anywhere in the function, control immediately returns to the function that called this function. The expression is the value that should be returned. If no expression is supplied, a dummy value is returned. The expression should match they type of value that the caller of this function is expecting. That is, if an int type is expected, the expression should resolve to an integer value. If a str type is expected, the expression should resolve to a string value. Due to memory constraints, a local string variable may **NOT** be returned from a function. Some examples are:

```
return;
return 1;
return level;
return (sum + 25);
return "hello";
return (func() + 20);
```

Notice that when a complex expression is returned it is usually surrounded by parentheses. This is done only for clarity and is not necessary. Also, it should be clear that what is returned is not the expression but what it evaluates to.

BREAK Statement

Often while using a looping statement (while, do...while, for), it is necessary to break out of (exit) the loop. The break statement serves this purpose. When the break statement is encountered, execution of the innermost while, do...while, for, or switch statement is terminated, and execution continues immediately after the terminated statement. It is an error for a break statement to appear outside of a loop. The format of the break statement is:

break;

For example, assuming you had the following code:

```
int num = 0;
while (1)
{
  num = num + 1;
  if (num > 100)
     break;
}
prints("Done");
```

Ordinarily, since there will always be a non-zero (TRUE) value in the conditional part of this <u>while</u> statement, it would execute forever. However, when the 'num' variable is greater than 100, the break statement is executed to exit from the loop, at which point the next statement would be executed (the function call to prints).

CONTINUE Statement

The continue statement is used within a loop (<u>while</u>, <u>do...while</u>, or <u>for</u> statement). The continue statement has the form:

```
continue;
```

It is illegal for a continue statement to appear outside of a loop body. When a continue statement is encountered, program control is immediately transferred to the end of the body of the innermost enclosing while, or for statement. The effect in a while or do...while statement is that the condition part of the loop is evaluated, and the next iteration occurs. For example:

```
num = 0;
while (num < 100000)
{
  num = num + 1;
  if (num > 100)
     continue;
  prints("Hello");
}
```

The effect of the continue statement in the above loop would be that 'Hello' would only be printed while num was less than or equal to 100, as the continue statement is executed when num is greater than 100, which causes the rest of the loop body to be skipped. An example of the use of continue in a for statement would be:

```
for (num = 0; num < 100000; num = num + 1)
{
  if (num > 100)
    continue;
  prints("Hello");
}
```

The effect in this case would be the same. While num is less than or equal to 100, the entire loop body executes. If num is greater than 100 however, the continue statement is executed. This causes the rest of the loop body to be skipped, so the 'Hello' is then not printed.

GOTO Statement

The goto statement is used to branch (jump) from one place to another, within a function. The use of the goto statements is generally considered bad style. They can make code very hard to understand, and are in fact almost never necessary. For example, Telix for Windows is written mainly in the Pascal language, which has a goto statement, yet except for a few pieces of speed-critical, prewritten code, the goto statement was never used or needed. On the other hand, used very sparingly and properly, it can sometimes make some code clearer and perhaps faster. The goto statement consists of two parts, the 'label' or marker, which is where execution will jump to, and the actual goto itself. A label is defined in the form

<identifier>:

where <identifier> follows the same rules as for variable names. Note that a colon follows the name, not a semicolon. The colon character must immediately follow the label name, with no intervening spaces. A label does not have to be on a line by itself, and is not considered a statement by itself. The goto takes the form

```
goto <label>;
```

where < label> is a label elsewhere in the function defined as described above. Execution of the script will immediately continue following the label.

An example is:

```
start:
  prints("Hello");
  goto start;
```

This would print the word "hello" over and over, forever. There is no restriction on the placement of a label, so the above can be written as:

```
start: prints("Hello");
goto start;
```

As mentioned above, there are usually better ways than using a goto statement. For example:

```
int i = 0;
do
   i = i + 1;
while (i < 100);</pre>
```

is clearer than the equivalent:

```
int i = 0;
loop:
    i = i + 1;
    if (i < 100)
    goto loop;</pre>
```

One good use of a goto statement is to get out of a deeply nested while statements, without having to do a lot of extra checking.

SWITCH Statement

The **switch** statement is new to SALT in Telix for Windows, and follows the form of the switch statement in the C language exactly. A **switch** statement is used when you must test for a number of possible conditions. The switch statement has the form:

where <*value*> is an integer variable and n1, n2, etc. are integer constants. The switch statement compares the integer constants you list in each of the case labels against the actual value in <*value*>. When a match is found, control is passed to the <*statement*> following the case label that was equal to <*value*>. Execution then continues until the <u>break</u> statement is encountered, indicating the end of processing, and exit from the switch statement. This is most commonly used as a replacement for multiple <u>if.else</u> statements. An example is:

```
int i;
i = 5;
switch (i)
{
   case 1: prints("i equals 1");
        break;
   case 2: prints("i equals 2");
        break;
   case 3: prints("i equals 3");
        break;
   case 4: prints("i equals 4");
        break;
   case 5: prints("i equals 5");
        break;
   default: prints("i cannot be determined");
}
```

This example could be written using multiple if statements as:

```
int i;
i = 5;
if (i == 1)
   prints("i equals 1");
else
if (i == 2)
   prints("i equals 2");
else
   if (i == 3)
        prints("i equals 3");
else
```

```
if (i == 4)
  prints("i equals 4");
else
  if (i == 5)
    prints("i equals 5");
else
    prints("i cannot be determined");
```

But, as you can see, it becomes much more difficult to read, as well as being slightly slower in execution time.

Preprocessor

Telix for Windows now includes what is commonly referred to as a *Preprocessor*. The preprocessor allows control over the compiler in matters such as what will be compiled under what conditions, inclusion of other script files, and other related items. The preprocessor will prove to be invaluable, especially if you develop large scripts, or scripts that target both the Windows and DOS versions of Telix.

See also

Preprocessor Commands, Predefined Conditional Symbols

Preprocessor Commands

The Telix for Windows script preprocessor uses the following identifiers:

#DEFINE
#UNDEF
#IFDEF
#IFNDEF
#ELSE
#ENDIF
#INCLUDE
#INCLUDEDIR
#COMPILETO
#STACK
#CONST
#DEBUGON
#DEBUGOFF

#DEFINE Directive

Example

#DEFINE *id*

Description

This directive defines a conditional symbol with the name given in id. The symbol is recognized for the remainder of the compilation, even in subsequently included files, until an $\frac{\#\text{UNDEF}}{\text{directive}}$ directive with the same id appears. The #DEFINE directive has no effect if id is already defined.

See also

#UNDEF, **#IFDEF**, **#IFNDEF**

Close Print Copy

#DEFINE Example

```
#DEFINE TESTING

main()
{
   prints("This is always displayed.");
   #IFDEF TESTING
     prints("This is displayed if TESTING is defined");
   #ELSE
     prints("This is displayed if TESTING is not defined");
   #ENDIF
}
```

#UNDEF Directive

Example

#UNDEF *id*

Description

This directive undefines a previously defined conditional symbol with the name given in *id*. The symbol is forgotten for the remainder of the compilation or until it reappears in a <u>#DEFINE</u> directive. The **#UNDEF** directive has no effect if *id* is already undefined.

See also

#DEFINE, **#IFDEF**, **#IFNDEF**

Close Print Copy

#UNDEF Example

```
#DEFINE TESTING
main()
{
   prints("This is always displayed.");
   #IFDEF TESTING
     prints("This is displayed if TESTING is defined");
   #ELSE
     prints("This is displayed if TESTING is not defined");
   #ENDIF
   #UNDEF TESTING
     #IFDEF TESTING
     prints("This is never displayed.");
   #ENDIF
}
```

#IFDEF Directive

Example

#IFDEF id

Description

This directive compiles the source code that follows it if the name given in id is defined. The compilation continues until an $\underline{\#ELSE}$ or $\underline{\#ENDIF}$ directive is reached. If id is not defined, the source code following the #IFDEF directive is ignored until an $\underline{\#ELSE}$ or $\underline{\#ENDIF}$ directive is reached.

See also

#DEFINE, #UNDEF, #IFNDEF, #ELSE, #ENDIF



```
#DEFINE TESTING
main()
{
   prints("This is always displayed.");
   #IFDEF TESTING
     prints("This is displayed if TESTING is defined");
   #ELSE
     prints("This is displayed if TESTING is not defined");
   #ENDIF
   #UNDEF TESTING
   #IFDEF TESTING
     prints("This is never displayed.");
   #ENDIF
}
```

#IFNDEF Directive

Example

#IFNDEF id

Description

This directive compiles the source code that follows it only if the name given in id is not defined. The compilation continues until an $\#\underline{\mathsf{ELSE}}$ or $\#\underline{\mathsf{ENDIF}}$ directive is reached. If id is defined, source code following the $\#\mathsf{IFNDEF}$ directive is ignored until an $\#\underline{\mathsf{ELSE}}$ or $\#\underline{\mathsf{ENDIF}}$ is found.

See also

#DEFINE, #UNDEF, #IFDEF, #ELSE, #ENDIF

```
#IFNDEF Example
```

```
#DEFINE TESTING
main()
{
   prints("This is always displayed.");
   #IFNDEF TESTING
     prints("This is NOT displayed if TESTING is defined");
   #ELSE
     prints("This is displayed if TESTING is defined");
   #ENDIF
   #UNDEF TESTING
   #IFNDEF TESTING
     prints("This is always displayed.");
   #ENDIF
}
```

#ELSE Directive

Example

#ELSE

Description

This directive switches between compiling and ignoring the source code delimited by the last #IFDEF or #IFNDEF and the next #ENDIF.

See also

#DEFINE, #UNDEF, #IFDEF, #IFNDEF, #ENDIF

```
#ELSE Example
```

```
#DEFINE TESTING

main()
{
   prints("This is always displayed.");
   #IFDEF TESTING
      prints("This is displayed if TESTING is defined");
   #ELSE
      prints("This is displayed if TESTING is not defined");
   #ENDIF
}
```

#ENDIF Directive

Example

#ENDIF

Description

This directive ends the conditional compilation initiated by the last $\underline{\#IFDEF}$ or $\underline{\#IFNDEF}$ directive.

See also

#DEFINE, #UNDEF, #IFDEF, #IFNDEF, #ELSE

#ENDIF Example

```
#DEFINE TESTING

main()
{
   prints("This is always displayed.");
   #IFDEF TESTING
     prints("This is displayed if TESTING is defined");
   #ELSE
     prints("This is displayed if TESTING is not defined");
   #ENDIF
   prints("This is always displayed.");
}
```

#INCLUDE Directive

#INCLUDE "<filename>"

Description

This directive instructs the compiler to include the file **<filename>** in the compilation. In effect, the file is inserted in the compiled text at the point where the **#INCLUDE** directive appears. If **<filename>** does not specify a directory for the file, it will be searched for in the following order:

Directories specified in the <u>#INCLUDEDIR</u> directive.



Path of the script being compiled.



Telix for Windows default script directory.



Current directory.

See also

#INCLUDEDIR, #COMPILETO

#INCLUDEDIR Directive

#INCLUDEDIR "<path>"

Description

This directive instructs the compiler to include the directories specified in **<path>** when searching for files named in an #INCLUDE directive, <u>call</u> or <u>calld</u> function. The compiler will search for specified files if a script is used that does not have a path specified. The search is performed in the following order:

<u>Directories</u> specified in the <u>#INCLUDEDIR</u> directive.



Path of the script being compiled.



Telix for Windows default script directory.



Current directory.

Multiple directories may be specified in <path>, and are separated by the semicolon (;) character.

See also

#INCLUDE, #COMPILETO

#COMPILETO Directive

#COMPILETO "<filename>"

Description

This directive tells the compiler to name the resulting compiled script as the filename specified in **<filename>**, rather than based on the source filename. By default, the compiler uses the source filename, and changes the extension to .SLC. Using this directive, the compiled script may be given a completely different filename, including extension. Also, compiled script files may also be placed in another directory, as **<filename>** may contain drive and/or directory specifications.

See also

#INCLUDE, #INCLUDEDIR

#STACK Directive

#STACK bytes

Description

This directive instructs the compiler to reserve the number of bytes specified in *bytes* for the script's stack space. The default stack size for Telix for Windows' scripts is 1024 bytes. The minimum stack size is 64 bytes and the maximum is 32768.

The stack is an area of memory used to store various pieces of information during the execution of a script. Such information includes variables local to functions, parameters passed to functions, and other items that are transparent to the script programmer. The default stack size is sufficient for most scripts, but if you are writing exceptionally large scripts, you may find you need to increase this size.

#CONST Directive

#CONST id value

Description

This directive defines a symbol specified in *id* and assigns the value specified in *value* to it. This symbol can then be referred to as if it were a normal variable, with the exception that it's value cannot be changed. The *value* parameter must be an integer, as string constants are not supported at this time.

#DEBUGON Directive

#DEBUGON

Description

This directive instructs the compiler to include information in the compiled script that can aid you in finding errors in it. The resulting script will be larger than one compiled without debugging information, but during development of a script the information can be invaluable. A script compiled with debugging information will report any errors that occur in the same manner as one without debugging information, but in addition it will report the filename of the script and the line number the error occurred on. In the interests of size and efficiency, debugging information cannot exceed 64k, which translates to roughly 13,000 lines of code. If your script exceeds 13,000 lines, you can selectively turn the debugging information on and off by using the #DEBUGOFF preprocessor directive.

See also

#DEBUGOFF

#DEBUGOFF Directive

#DEBUGOFF

Description

This directive instructs the compiler not to include debugging information in the compiled script. By default, scripts do not include debugging information, but if you use the <u>#DEBUGON</u> preprocessor directive, you may need to use this directive in certain situations. In the interests of size and efficiency, debugging information cannot exceed 64k, which translates to roughly 13,000 lines of code. If your script exceeds 13,000 lines, you can selectively turn the debugging information off with **#DEBUGOFF**. This will allow you to include debugging information in the sections of your script where errors may occur, and exclude it in sections that do not need it.

This directive is not required if you have specified the <u>#DEBUGON</u> directive. The compiler will include debugging information from the point it encounters the <u>#DEBUGON</u> directive until it reaches the maximum amount it can include in the script file. Most scripts will never need this directive, as few will approach the 13,000 line limit.

See also

#DEBUGON

Predefined Conditional Symbols

SALT II defines the following standard conditional symbols:

SALTII Always defined in Telix for Windows v1.00. If this symbol is defined, the version of the SALT script compiler is compatible with SALT II, first introduced with this version of

Telix.

WINDOWS Defined for Windows versions of Telix, indicating that the version of Telix being used to

compile the script is a native MS-Windows application.

Built-In Functions

Telix for Windows' SALT II language has quite a large number of built-in functions. These functions are called just as you would call your own SALT functions. Each performs a certain task (print something to the screen, manipulate strings, access disk files, etc.) and is called with parameters in a certain format and returns an integer or string value (the return value does not have to be used and can often be ignored).

Select the Quick List to view a listing of the functions grouped by the type of action they perform. Following the Quick List is a complete reference of each function in alphabetical order (accessed by the Next (>>) and Previous (<<) buttons above), including a summary of the calling format, a description of what it does, and the return value of the function. An example of actual usage of the function is also often given to illustrate usage. Note that the examples are fragments of program code for the most part, and may not explicitly declare all needed variables. So that you may find related functions, each function description has a 'See Also' section, which lists related functions. Simply click on any item in the 'See Also' section to move to that part of the help. For users of Telix for DOS, a New & Changed section is included that lists the SALT functions that are new or have had their behavior modified in some way.

Function Quick List

Character Handling:

Checks to see if a character has a value of 0-255 <u>IsAscii</u> **IsAlNum** Checks to see if a character is a letter A-Z or a digit **IsAlpha** Checks to see if a character is a letter of the alphabet **IsCntrl** Checks to see if a character is a control character Checks to see if a character is a numeric digit 0-9 **IsDigit** <u>IsLower</u> Checks to see if a character is a lower case letter **IsUpper** Checks to see if a character is an upper case letter Converts a character to lower case if it is not already **ToLower ToUpper** Converts a character to upper case if it is not already

Comm Port Operations:

Determines whether a carrier is present on the port **Carrier** Counts the received chars that have not been handled clnp Cnt <u>cGetC</u> Returns the next character from the receive buffer Waits a set time for the next character to be received <u>cGetCT</u> <u>cPutC</u> Sends a single character to the connect device buffer Sends an integer value to the port as an ASCII string <u>cPutN</u> **cPutS** Sends a string of chars to the connect device buffer cPutS_TR Sends a string through the emulation to the port **FlushBuf** Throws away any chars waiting in the receive buffer Returns the connect devices DCE or DTE rate Get Baud Get DataB Returns the connect devices current data bits setting Get_Parity Returns the connect devices current parity setting Get_Port Returns the connect devices current port number 1-8 Get_StopB Returns the connect devices current stop bit settings Terminates the current connection if any, like ALT-H Hangup Sends a break signal to the modem, like CTRL-END Send Brk Changes the current connect device to a new device Set ConnectDevice Set CParams Sets new speed, data bits, parity, and stop bits Function no longer used, use Set ConnectDevice Set Port

File Input/Output Operations:

<u>fClearErr</u> Clears the error flag associated with an open file **fClose** Closes a file previously opened with fOpen **fDelete** Deletes a specified file; the file may not be open **fError** Checks for file errors (i.e. writing to a read only file) Checks a file pointer for the end of an open file <u>fEOF</u> **fFlush** Clears a file buffer (writes all cached writes) **fGetC** Retrieves the next character from an open file **fGetS** Retrieves string of specified length from an open file **FileAttr** Checks a specified file for being hidden, a subdir, etc. FileFind Determines if a file matching a mask is on disk

Returns the size of a given file in bytes **FileSize**

Extracts parts of a filename from a complete path <u>fnStrip</u> Opens a specified file for reading or writing **fOpen**

Writes a single character to a file opened for writing fPutC Writes a given string to a file opened for writing **fPutS** Reads a block of characters from an open file **fRead** <u>fRename</u> Renames a file to a specified new file name

fSeekMoves the file pointer of open files to a new locationfTellRetrieves the file pointer position of an open filefWriteWrites a block of characters to a write-opened file

File Transfers and Logs:

<u>Capture</u> Opens, closes, pauses, or unpauses specified log file <u>Capture Stat</u> Retrieves the state of the capture log (open, paused)

<u>Printer</u> Toggles the Printer Log on and off

Receive Download specified files with a specified protocol
Send Upload specified files with a specified protocol
Set_DefProt Change the connect devices default protocol
UsageLog Opens and closes the default Usage Log

<u>Usage Stat</u> Determines whether the Usage Log is open or closed

<u>UStamp</u> Enter text, and optionally the date and time, into the usage log file

Input String Matching:

TrackTells Telix to watch for a string to be matchedTrack AddChrAdds a character to the tracked-string bufferTrack FreeTells Telix to stop watching for a matched stringTrack HitAsks Telix whether a tracked string has been foundWaitForWait for a string for a specified period of time

Keyboard Operations:

InKeyRetrieves a key, if available from the keyboard bufferInKeyWRetrieves a key, waiting if needed, from the keyboardKeyGetReturns the contents of a given keyboard macro

KeyLoadLoads a specified keyboard macro tableKeySaveSaves the current keyboard macro tableKeySetSet a specific key to contain a new macro

Miscellaneous Functions:

DosOpens a DOS window and executes a DOS commandDialDials a PhoneBook number or group of numbersDosFunctionNo longer used included for Telix for DOS usersExitTelixExits Telix immediately; may or may not hang upGetEnvReturns a variable value in the master environment

HelpScreen Displays the Telix for Windows Help file

LoadFonLoads a different PhoneBook into Telix for WindowsNewDirChanges the current directory to the one specifiedRandomGenerates a random number in a specified rangeRedialRedials the currently selected PhoneBook entries

RunRuns a DOS or Windows applicationSet_TerminalSets the Terminal Device to a new deviceTelixVersionReturns a value indicating the version of Telix

<u>Terminal</u> Processes any characters waiting in the receive buffer <u>TransTab</u> Load or clear a specified translation table device <u>Update Term</u> Updates Telix with changes to colors and formats

Script Management:

<u>ArgCount</u> Reports the number of parameters passed

<u>CallD</u> Load and execute a script

<u>Load Scr</u>
<u>ScriptVersion</u>
TelixForWindows

Determines if a given script is already loaded
Loads a script into memory but does not run it
Reports the version of SALT running this script
Determines if Telix is the DOS or Windows version

<u>Unload Scr</u> Unloads a script from memory

Sound Functions:

<u>Alarm</u> Generates standard alarm sound via .WAV file <u>PlayWave</u> Plays the specified Windows .WAV sound file

Tone Generates the specified tone through the PC Speaker

String Handling:

<u>CopyChrs</u> Copies characters from one string to another

<u>CopyStr</u> Copies one string into another string at a given point

DelChrs Deletes a number of characters from a string **GetS** Gets a string from the users keyboard via prompt **GetSXY** Prompts the user to input a string at a certain location **InputBox** Standard Windows edit prompt w/ OK and CANCEL **InsChrs** Insert characters from one string into another string <u>ItoS</u> Converts an integer value to a string of ASCII digits Puts a character into a string at a given position <u>SetChr</u> **SetChrs** Puts a string into another string at a given position Converts a string of digits into an integer value Stol **StrCat** Concatenates one string to the end of another string **StrCmpl** Compares a pair of strings, ignoring upper/lower case Determines the length of a null-terminated string StrLen <u>StrLower</u> Converts a string of characters to all lower case Returns the maximum number of chars a string holds **StrMaxLen**

StrPos Search for a given string in another string

StrPoslSearch for a string in another string, ignoring caseStrUpperConverts a string of characters to all upper caseSubChrReturns the character found at a given string positionSubChrsReturns a substring of given length from a stringSubStrReturns a substring within a string stopping at NULL

Time, Date and Timer Operations:

<u>CurTime</u> Returns the current time of day

<u>Date</u> Returns the current date

Delay Delays Telix for a number of tenths of seconds Delay_Scr Delays scripts but not Telix for n tenths of a second <u>tDay</u> Returns the day portion of the date as a number 1-31 Returns the hour part of the time as a number 0-23 **tHour** Returns the minute part of the time as a number 0-59 <u>tMin</u> Returns the month part of the date as a number 1-12 <u>tMonth</u> Returns the seconds part of the time as a value 0-59 **tSec** Returns the year of the date as a number 1970-2019 **tYear**

<u>Time</u> Returns the current time of day

<u>Time_Up</u> Checks to see if a set timer has elapsed
<u>Timer_Free</u> Releases a set timer handle; turns off a timer

<u>Timer_Restart</u> Restarts a timer at a given count

<u>Timer_Start</u> Starts a timer in a value of tenths of a second

<u>Timer_Total</u> Returns tenths of a sec. since a time was (re)started

Video Operations:

Box Creates a box on the screen, in color

<u>CNewLine</u> Sends a carriage return to the terminal device

Cursor OnOff Turns the cursor on or off

<u>Clear Scr</u> Clears the current terminal window

GetTermHeight
GetZermWidthReturns the current terminal height in chars
Returns the current terminal width in chars
Returns the current character column position
Returns the current character row position
Moves the cursor to the given (x,y) char position
MsqBoxGetZermWidth
Returns the current character row position
Moves the cursor to the given (x,y) char position
Present a standard Windows dialog, specified buttons

NewLine Sends a LineFeed to the terminal device

PrintC Print a character on the screen, but do not send it PrintC_Trm Print a character on the screen, including ^M-types **PrintN** Print the specified integer on the screen as a string Print the specified number on the screen via the term PrintN Trm **PrintS** Prints a string on the screen, but does not send it Prints a string on screen thru the terminal emulation PrintS Trm **PrintSC** Prints a string on the screen, does not move cursor Prints a string thru emulation, does not move cursor PrintSC Trm **PStrA** Prints a string to the screen with given color attributes **PStrAXY** Prints a string at a given location, in a given color

Scrolls or clears a region on the screen

Status_Wind Pops up a timed, information window with your text Gets the character and its color from a given location <u>vGetChr</u> Gets a string of characters from a screen position <u>vGetChrs</u> Gets a string and its attributes from a screen location vGetChrsA Puts a character and color into a screen position <u>vPutChr</u> Puts a string of characters onto the screen **vPutChrs** Puts a string and attributes onto the screen **vPutChrsA** Restores a previously saved buffer to the screen vRstrArea Saves a screen region to a buffer for later restoration <u>vSaveArea</u>

New & Changed SALT Functions

To help you quickly locate functions that have been added to SALT II and review changes that have been made to existing functions, use the lists below.

New Functions:

<u>ArgCount</u>	<u>GetTermWidth</u>	<u>PlayWave</u>	Set_ConnectDe
CNowl inc	InnutDov	DrintC Trm	vice
<u>CNewLine</u> CPutN	<u>InputBox</u> <u>MsgBox</u>	PrintC_Trm PrintN_Trm	ScriptVersion TelixForWindow
<u> </u>	<u></u>		<u>S</u>
<u>ConnectDevice</u> Name	<u>NewLine</u>	PrintS_Trm	<u>TelixVersion</u>
<u>GetTermHeight</u>	NumConnectD evices	Random	

Changed Functions:

<u>Alarm</u>	<u>fRead</u>	<u>KeySet</u>	Terminal
<u>Call</u>	<u>fWrite</u>	Run	WaitFor
<u>Dial</u>	Get_Baud	Set_Port	
Dos	<u>GotoXY</u>	Show_Director	
		¥	
<u>FileFind</u>	<u>KeyGet</u>	Status_Wind	

Alarm Function

Example

Alarm(int times);

The alarm function plays the wave sound defined in the Sounds configuration as Alarm.

Argument Description
The number of times to play the wave sound.

Return Value

The number of times the alarm was sounded.

See also

PlayWave, Tone, alarm on, sound on

Alarm Example

```
// Sounds an alarm until a key is pressed.
while (!inkey())
  alarm(1);
```

ArgCount Function

ArgCount();

The argcount function determines the number of parameters that were passed to a script that was executed via the $\underline{\text{call}}$ or $\underline{\text{calld}}$ function.

Return Value

The number of parameters passed.

ArgCount Example

```
if (ArgCount() < 1)
  prints("Called script requires at least one parameter.");</pre>
```

Box Function

Example

Box(int x, int y, int x2, int y2, int style, int hollow, int color);

The box function is used to create a box on the screen. The box must fit within the confines of the screen.

Argument	Description	
int x, int y int x2, int y2 int style	Upper left corner of the box. Lower right corner of the box. Specifies what kind of border	
	to use:	
	0	
	Spaces	
	 Single lines 	
	2 Double lines	
	 Single vertical lines, 	
	double horizontal lines	
	4 Double vertical lines, single horizontal lines	
int hollow	If non-zero, the inside of the box will not be cleared.	
int color	The desired <u>color</u> of the box.	
Return Value		
A zero is always	s returned.	

See also

<u>Scroll</u>

Box Example

```
box(10, 10, 70, 20, 1, 0, 112);  // draw box with black characters on
  // white background. The upper left
  // corner is 10, 10 and the lower
  // right corner is 70, 20.
```

Call Function

Call(str scriptname, arg1, arg2, ..., argn);

The call function now acts in exactly the same manner as $\underline{\text{calld}}$, and it is included only for compatibility with Telix for DOS.

See also

CallD, Load Scr, Unload Scr, Is Loaded

CallD Function

Example

CallD(str scriptname, arg1, arg2, ..., argn);

The calld function is used when one script file must call (jump into and then return from) another.

Argument	Description
str	The name of the script file to call. If no
scriptname	extension is given, .SLC is assumed.
arg1, argn	The parameters to pass to the called scripts
	main function.

Return Value

The value returned by the called scripts **main** function. It can be either an integer or a string, although called scripts cannot return string variables that are local to itself. If the script cannot be loaded, or it is aborted by the user, a value of -1 is returned.

See also

Load Scr, Unload Scr, Is Loaded

CallD Example

```
stat = calld("TEST");
if (stat == -1)
  prints("Called script could not be loaded or was aborted!");
```

Capture Function

Example

Capture(str filename);

The capture function is used to open, close, pause, and unpause the Telix capture file. Depending on what the string variable *filename* contains, different actions will take place.

Argument	Description
str filename	If a valid filename (which can include a path),
	Telix opens and starts capturing data to the
	file.
	If <i>filename</i> is *CLOSE* and the capture file is
	open, it is closed.
	If <i>filename</i> is *PAUSE* and the capture file is
	open, capturing is suspended.
	If filename is *UNPAUSE* and the capture file
	is paused, capturing is resumed.
	If filename is an empty string (), Telix takes the
	same action as if the user had selected
	Capture Log from the File menu.

Return Value

A value of -1 is returned if there is a problem performing the indicated function, otherwise a non-zero (TRUE) value is returned.

See also

Capture Stat, Printer, capture fname

Capture Example

```
if (capture("TELIX.CAP") == -1)
  prints("Error opening capture file!");
    ...
capture("*PAUSE*");
capture("*UNPAUSE*");
capture("*CLOSE*");
```

Capture_Stat Function Example

Capture_Stat();

The capture_stat function determines the state of a capture file.

Return Value

Returns an integer value representing the current status of the capture file, as follows:

Capture File is closed. 1 Capture File is open. 2 Capture File is open and paused.

See also

Capture, Usage_stat



Capture_Stat Example

Carrier Function

Example

Carrier();

The carrier function determines whether there is a carrier present or not by checking the Carrier Detect signal of the modem. Note that some modems by default override the real state of the signal and always send a high. For this function to work, the modem must be told to supply the real signal. This function may be used to check if Telix is connected to a remote service over the modem, as the Carrier Detect signal should be on if there is a connection. Note also that if you are connecting two computers via a null-modem cable, the value returned will depend on the wiring of the cable being used.

Return Value

Returns a non-zero (TRUE) value if the Carrier Detect signal coming from the modem is on (high), otherwise it returns a zero (FALSE) value.

Carrier Example

```
if (carrier())
  prints("We are online.");
else
  prints("We are offline.");
```

cGetC Function

Example

cGetC();

The cgetc function returns the first character waiting in the received data communications buffer. The cinp_cnt function may be used to see if there are any characters waiting in the buffer.

Return Value

Returns the first character waiting in the communications buffer. If there are no characters in the buffer, a value of -1 is returned.

See also

cGetCT, clnp_Cnt

cGetC Example

```
// If there are characters in the buffer, get the first and put it in chr.
int chr;
if (cinp_cnt())
   chr = cgetc();
```

cGetCT Function

Example

cGetCT(int timeout);

The cgetct functions returns a character from the communications port, waiting up to a specified time to receive one. If a character is already waiting in the communications buffer, it is immediately returned.

Argument	Description
int timeout	The amount of time to wait in tenths of
	seconds.

Return Value

Returns the first character received in the communications buffer in the specified time. If no character is received within the timeout period, a value of -1 is returned.

See also

cGetC, clnp_Cnt



cGetCT Example

```
// Wait for up to 10 seconds to receive a character and put it in chr.
int chr;
if ((chr = cgetct(100)) == -1)
   prints("Timeout!");
else
   printc(chr);
```

ChatMode Function

ChatMode(int echo_remote);

The chatmode function is not supporte in this version of Telix for Windows.

temp

Beta Note:

This function has not yet been implemented.

The chatmode function enters the built-in chat mode.

Argument	Description
int	If non-zero (TRUE), characters typed
echo_remote	by the remote user are echoed back to the remote.
Return Value	
A zero is always	returned.

cInp_Cnt Function

cInp_Cnt();

The cinp_cnt function determines the number of characters waiting in the communications buffer.

Return Value

Returns the number of characters waiting in the received data communications buffer.

See also

cGetC, cGetCT

clnp_Cnt Example

\bigcirc	0	—	-1!
Clear	Scr	⊢⊓n	CTION
Cicai	$\mathcal{O}_{\mathcal{O}_{\mathbf{I}}}$	ı aıı	Otioi

Clear_Scr();

<u>Scroll</u>

The clear_scr function clears the screen and places the cursor in the upper left corner at position 0,0.

	Return Value	
	A zero is always returned.	
See also		

CNewLine Function

CNewLine();

The cnewline function is used to send a carriage return to the terminal. No line feed is sent.

Return Value

A non-zero (TRUE) value is returned unless the character can not be sent for some reason, in which case a value of -1 is returned.

See also

<u>NewLine</u>

ConnectDeviceName Function

Example

ConnectDeviceName(int num, str buffer);

The connectdevicename function is used to retrieve the name of a specified connect device.

Argument	Description
int num	The number of the connect device to retrieve.
str buffer	The variable to hold the name of the connect device.

Return Value

The buffer string is returned.

See also

NumConnectDevices, Set_ConnectDevice



ConnectDeviceName Example

```
int num;
str buff[30];

for (num = 1; num <= numconnectdevices; ++num)
   prints(connectdevicename(num, buff));</pre>
```

CopyChrs Function

CopyChrs(str source, str target, int pos, int count);

The copychrs function copies a number of characters from one string into another and returns *target*. Note that string indexes begin at 0, not 1 as in some languages.

Argument	Description
str source	The string to copy characters from.
str target	The variable to copy characters to.
int pos	The index of <i>target</i> string to begin copying characters to.
int count	The maximum number of characters to copy. If <i>target</i> is not large enough to hold all characters, only as many as will fit are copied.

Return Value

The *target* string is returned.

This function is very similar to <u>copystr</u>, except that it is not string oriented, and therefore does not stop copying characters when a 0 value (NULL) character is encountered.

See also

CopyStr, SubChrs, SubStr

CopyStr Function

CopyStr(str source, str target, int pos, int count);

The copystr function copies from the source string into target string and returns target. Characters are copied until a 0 (NULL) value is encountered, normally at the end of every string, or *count* characters are copied. A 0 (NULL) is always copied to the end of the target string, but is not included as part of *count*.

Argument	Description
str source	The string to copy characters from.
str target	The variable to copy characters to.
int pos	The index of <i>target</i> string to begin copying characters to.
int count	The maximum number of characters to copy. If <i>target</i> is not large enough to hold all characters, only as many as will fit are copied.

Return Value

The *target* string is returned.

See also

CopyChrs, SubStr, SubChrs

cPutC Function

Example

cPutC(int character);

The cputc function sends a character to the communications port.

Argument Description
Int character The ASCII value of the character to be sent.

Return Value

A non-zero value is returned unless the character can not be sent for some reason, in which case a value of -1 is returned.

See also

cPutN, cPutS

cPutC Example

cPutN Function

Example

cPutN(int number);

The cputn function sends an integer number to the communications port.

Argument Description
int number The integer value to be sent.

Return Value

A non-zero (TRUE) value is returned unless the number can not be sent for some reason, in which case a value of - 1 is returned.

See also

cPutC, cPutS

cPutN Example

```
int i;
i = 23;
cputn(27);
cputn(i);
```

cPutS Function

Example

cPutS(str outstr);

The cputs function sends the specified string out over the communications port. A carriage return and line feed are **not** appended to the string.

Argument str outstr Description The string to be sent.

Return Value

A non-zero (TRUE) value is returned unless the character can not be sent for some reason, in which case a value of -1 is returned.

See also

cPutC, cPutN, cPutS_TR



cPutS_Tr Function

Example

cPutS_Tr(str outstr);

The cputs_tr function sends the specified string to the communications port, but pays attention to two output string translation characters, ^ and ~, described in the Telix for Windows manual and in the <u>SALT Syntax</u> section of this help file. This function is really only useful for sending the modem control strings that the user has defined in the configuration.

Argument	Description
str outstr	The string to sent.
Return Value	

A non-zero (TRUE) value is returned unless the character can not be sent for some reason, in which case a value of -1 is returned.

See also

<u>cPutS</u>

cPutS_Tr Example

```
cputs_tr(_modem_init);
cputs_tr("good-bye~yes^M");
```

Cursor_OnOff Function

Cursor_OnOff(int state);

The cursor_onoff functions turns the blinking cursor on or off.

Argument	Description
int state	If non-zero (TRUE),
	the cursor is disabled. If zero (FALSE), the cursor is enabled.

Return Value

A zero is always returned.

CurTime Function

Example

CurTime();

The curtime function returns the current date and time as the number of seconds since January 1, 1970. The date and time value in this format is used by many SALT functions.

See also

Date, Time, tYear, tMonth, tDay, tHour, tMin, tSec

CurTime Example

```
// Print the current date
int t;
str s[64];

t = curtime();
date(t, s);
prints(s);
```

Date Function

Example

Date(int timeval, str buffer);

The date function converts a Telix date value into a date string of the form specified by Windows (controlled through Windows' Control Panel).

Argument	Description
int timeval	The date, represented as the number of seconds since January 1, 1970.
atu la villa u	This format is returned by <u>curtime</u> and <u>filetime</u> , among others.
str buffer	A variable to receive the properly formatted string.
Return Value	

A zero is always returned.

See also

Time, Curtime, FileTime

Date Example

```
str s[16];
printsc("The current date is ");
date(curtime(), s);
prints(s);
```

Delay Function

Delay(int duration);

The delay function pauses the current script for a length specified amount of time. During this pause, everything is shut off except the asynchronous reception of characters from the communications port and mouse movement. Received characters are stored in the input buffer, and are **NOT** processed until after the delay.

Argument
int durationDescriptionThe amount of time to delay in tenths
of seconds.

Return Value

The duration parameter is returned.

See also

Delay Scr

Delay_Scr Function

Delay_Scr(int duration);

The delay_scr function pauses only the execution of the current script file for a specified amount of time. During that time, characters coming in from the serial port are printed on the terminal screen, and user keystrokes are also processed.

Argument int duration Description
The amount of time to delay in tenths of seconds.

Return Value

The duration parameter is returned.

See also

Delay

DelChrs Function

Example

DelChrs(str s, int pos, int num);

The delchrs function is used to remove or delete a number of characters in a string at a certain position.

Argument	Description
str s	The string to delete
	the characters from.
int pos	The index of the string
	from which to begin
	deleting characters.
int num	The number of
	characters to be
	deleted.

Return Value

The *target* string is returned.

See also

InsChrs

DelChrs Example

```
// remove all but the first and last characters in a string str s[] = "0123456789"; delchrs(s, 1, strlen(s) - 2);
```

Dial Function

Example

Dial(str dialstr, int maxtries, int no link);

The dial function allows you to dial entries in a variety of manners, and it allows control over the number of attempts and whether or not to execute linked scripts.

Argument	Description
str dialstr	A string containing the entries to be dialed. Entries can be specified as the PhoneBook entry number, the PhoneBook entry name, or partial name, enclosed in pipe () symbols, or a
	manually entered number prefaced by an m. If an actual pipe symbol is required, use a double pipe (). If <i>dialstr</i> is empty () then the PhoneBook is displayed.
int maxtries	The maximum number of dialing attempts to make. This is the total number of attempts, regardless of the number of entries being dialed. For instance, if three entries are to be dialed and
int no_link	maxtries is 6, each entry will be attempted twice. If maxtries is 0, dialing will continue until a connection is established. If non-zero (TRUE) and the entry connected has a linked script file, that script will NOT be executed. If no_link is zero (FALSE), linked scripts will be executed.

Return Value

If a connection is established, the dial function returns the entry number that was connected to (or 1 if a manual number was dialed). If no connection was established, 0 is returned. If *dialstr* is incorrectly formatted, -1 is returned.

When a connection is successfully established, several items of information regarding the entry are placed in <u>System Variables</u>. These variables are fully described in the <u>entry info</u> section.

See also

Redial, entry enum, entry logonname, entry name, entry num, entry pass



Dial Example

```
int stat;
str number list[255];
// The first dial will dial entry #10, the first entry with "delta" in it,
// entry #15, and My Pipe | BBS until a connection has been established,
// but do not execute any linked scripts.
dial("10 |delta| 15 |My Pipe || BBS|", 0, 1);
// Dial the number "967-1111" a maximum of 5 times. A no_link parameter is
// not required since a manual number is not going to have a linked script.
dial("m967-1111", 5);
// Construct the dial list in the variable number_list, dialing no more
// than 10 times total, and executing any linked scripts attached to the
// entry that establishes a connection.
number_list = "10 2 5 |deltaComm Online| 8";
stat = dial(number_list, 10, 0);
```

Dos Function

Example

Dos(str command, int mode);

The dos function calls the DOS command interpreter, usually COMMAND.COM, to open a DOS window, or execute a DOS based program. The command processor is found by attempting to run the DOSPRMPT.PIF program information file.

This function differs from the <u>run</u> function in that it is used **ONLY** to launch DOS programs. The <u>run</u> function should be used, when possible, as it can handle either Windows or DOS programs. The only situations that you may find the dos function useful are performing actual COMMAND.COM built-in functions (such as DEL, COPY, REN). Note that most of DOS' built-in functions have equivalent SALT functions, thus reducing the need for this function further.

Argument	Description
str command	Parameters to be passed to the command
	interpreter. If empty (), Telix will open a DOS
	window and await further action. If you specify a command or program that expects user input, make sure you are on hand to provide it.
int mode	This parameter does not apply to Telix for
	Windows, and it is included only for compatibility with Telix for DOS.

Return Value

The dos function returns a -1 if the command processor can not be found or there is not enough memory to load it, otherwise a 0 is returned.

See also

Run, DosFunction

Dos Example

// copy all files from the A: drive to the C: drive. dos("copy a:*.* c:", 0);

DosFunction Function

DosFunction();

The dosfunction function calls up the File menu, as if the user had pressed Alt-F while in terminal mode. This function is included for compatibility with Telix for DOS and is not recommended for use.

See also

Dos, Run

ExitTelix Function

Example

ExitTelix(int returncode, int hangup);

The exittelix function closes any currently open log file and exits Telix as if the user had pressed Alt-F4 while in terminal mode.

Argument	Description
int returncode	The value that should be returned
	to Windows. This value seems to
	be ignored by most Windows shell
	programs, but is included for
	compatibility with Telix for DOS.
int hangup	If non-zero (TRUE), Telix will hang up before exiting, otherwise the connection will not be disturbed.

Return Value

This function causes the termination of the script, thus there will never be a return value.

ExitTelix Example

```
// Exit Telix and break any connection which may be established.
exittelix(0, 1);

// Exit Telix without disturbing any connection which may be established.
exittelix(0, 0);
```

fClearErr Function

Example

fClearErr(int fh);

The fclearerr function clears the error flag and the End Of File (EOF) flag associated with an opened file.

Argument Description

The file handle representing the file to clear the error flags for.

Return Value

A zero is always returned.

See also

fError, fEof



fClearErr Example

```
int f;
f = fopen("test.dat", "r");

// Perform reading operations here.
if (ferror(f))
  fclearerr(f);
```

fClose Function

Example

fClose(int fh);

The fclose functions closes a file previously opened for reading or writing with the <u>fopen</u> function. If the file was opened for writing, any data which is still buffered and waiting to be written out to disk is written before the file is closed.

Argument	Description
int fh	The file handle
	representing the file to
	be closed.

Return Value

A -1 is returned if there is a problem closing the file, otherwise a zero is returned.

See also

<u>fOpen</u>

fClose Example

```
int f;
f = fopen("test.dat", "w");
// Perform read/write operations.
fclose(f);
```

fDelete Function

Example

fDelete(str filename);

The fdelete function is used to delete a disk file from within a script.

Argument str filename

Description

The name of the file to delete, optionally including a full drive and path. Wildcard characters (* or ?) may **not** be part of *filename*.

Return Value

A value of -1 is returned if there is a problem deleting the file, otherwise a zero is returned.

See also

fRename

fDelete Example

```
if fdelete("C:\UTIL\TLX\TELIX.CAP") == -1  // delete an old capture file
  prints("Error deleting file.");
```

fEOF Function

Example

fEOF(int fh);

The feof function determines if a file has reached the end-of-file position.

Argument	Description
int fh	The file handle
	representing the file to
	operated on.

Return Value

A non-zero (TRUE) value is returned if the file position is at the end of the file, otherwise a zero (FALSE) value is returned.

See also

fClearErr, fError



FEOF Example

fError Function

Example

fError(int fh);

The ferror function checks a file to determine if an error has occurred when using it. A file's error flag stays set until it is cleared with <u>fclearerr</u> or the file is closed using <u>fclose</u>.

Argument	Description
int fh	The file handle
	representing the file to
	checked.

Return Value

A non-zero (TRUE) value is returned if the error flag is set, otherwise a zero (FALSE) is returned.

See also

fClearErr, fEof



fError Example

fFlush Function

fFlush(int fh);

The fflush function flushes the buffer associated with a file. If the file is opened for writing, any characters in the buffer are written. If the file is opened for reading, the buffer is cleared.

Argument	Description
int fh	The file handle
	representing the file to flushed.

Return Value

A value of -1 is returned if there is a problem flushing the buffer, otherwise a zero is returned.

See also

fOpen, fClose

fGetC Function

Example

fGetC(int fh);

The fgetc function retrieves the next character from a specified file. The file must have been opened for reading or for reading and writing using the <u>fopen</u> function.

Argument	Description
int fh	The file from which the
	character is to be
	read.

Return Value

If successful, the character read is returned, or -1 if the end of the file has been reached or an error is encountered.

See also

fOpen, fPutC, fError, fEOF



fGetC Example

fGetS Function

Example

fGetS(str buffer, int n, int fh);

The fgets function reads characters from a file into a string variable. Reading stops when a line feed character (ASCII #10) is read, the end of file (EOF) is encountered, a read error occurs, or a specified number of characters have been read. The line feed character (and the carriage return that usually precedes it on MS-DOS systems) is not kept as part of the string.

Important: The SALT implementation of the fgets() function differs from the C language function of the same name. While both implementations read until the line feed character, C keeps that character as part of the input string, while SALT doesn't. This change was made because in almost every case, the line feed is not needed, and would otherwise have to be manually stripped by the script after every read.

Argument	Description
str buffer	The variable to receive
	the string read from
	the file.
int n	The maximum number
	of characters to be
	read.
int fh	The file handle from
	which to read the
	string.

Return Value

A value of -1 is returned if there is a read error, or if the end of file (EOF) is encountered before any characters can be read. If successful, a zero is returned.

See also

fOpen, fputs



fGetS Example

FileAttr Function

Example

FileAttr(str filespec);

The fileattr function determines the DOS attributes of a specified file.

Argument	Description
str filespec	The name of the file
	that may optionally
	include a drive and
	directory, as well as
	the DOS wildcard
	characters * and ?. If
	empty (), the attributes
	of the last file found
	with the <u>filefind</u> ,
	<u>filesize</u> , or <u>filetime</u>
	function is returned

The following is a brief description of the attributes that may be returned. For further explanation, see your DOS manual.

Return Value	Description
-1	The specified file
	could not be found.
1	Read only file.
2	Hidden file.
4	System file.
8	Volume label. This is
	the volume name of
	the disk.
16	Subdirectory.
32	Archive bit.

See also

FileFind, FileSize, FileTime



FileAttr Example

```
int attr;
str filename[64];
gets(filename, 64);
attr = fileattr(filename);
if (attr & 6)
               // system and hidden added together
  prints("This file is marked as hidden and system");
else
  if (attr == -1) then
   prints("File not found.");
  if (attr & 1) then
   prints("Read-Only");
  if (attr & 2) then
   prints("Hidden");
  if (attr & 4) then
   prints("System");
  if (attr & 8) then
    prints("Volume Label");
  if (attr & 16) then
    prints("Subdirectory");
  if (attr & 32) then
   prints("Archive");
```

FileFind Function

Example

FileFind(str filespec, int attrib[, str buffer]);

The filefind function is used to search for the existence of one or more files or subdirectories. The size, date, time, and attributes of the matched file can be determined with the $\underline{\text{filesize}}$, $\underline{\text{filetime}}$, and $\underline{\text{fileattr}}$ functions, respectively.

Argument	Description
str filespec	The file specification which may include a drive and path as well as a filename, and may use the DOS wildcard characters * and ?. If empty, filefind searches for the next file matching the filespec used in the previous call to
	filefind.
int attrib	File attributes that must be set for files to match. See <u>fileattr</u> for more information.
str buffer	The variable to place matching filenames in (drive and path are NOT included). This parameter is optional and need not be included if only testing for file existence.

Return Value

A non-zero (TRUE) value is returned if a matching file is found, otherwise a value of zero (FALSE) is returned

See also

FileSize, FileTime, FileAttr



FileFind Example

```
// Constants to provide easier more readable function calls.
#CONST faNormal
                0
#CONST faReadOnly
                1
#CONST faHidden
#CONST faSystem
                4
#CONST faVolumeID
                8
#CONST faDirectory
               16
#CONST faArchive
// The first example is a simplified, but quite functional, //
// to determine if a file exists.
// EX: if FileExist("C:\AUTOEXEC.BAT") ....
FileExists(str fname)
 int result;
 result = filefind(fname, faNormal);
 if (result)
   return 1;
 else
   return 0;
// Show all normal files in the specified directory
// EX: ShowDirectory("C:\DOS\");
ShowDirectory(str path)
 str buf[16], fspec[16];
 // Set local variable equal to the passed directory.
 fspec = path;
 // Tack the *.* wildcard specification onto the end of the directory.
 strcat(fspec,"*.*");
 // Continue to loop as long as we find files. Note also that we have
 // given a third paramter, buf, to put the files found into.
 while (filefind(fspec, faNormal, buf) != 0)
  prints(buf);
              // show file found
   fspec = "";
              // so we can continue searching for files
  }
}
// show all files with both the hidden and system attributes in the //
```

FileSize Function

Example

FileSize(str filespec);

The filesize function determines the size in bytes of a file.

Argument

Description

str filespec

The name of the file that may include a drive and directory, as well as the DOS wildcard characters * and ?. If empty (), the size of the last file found with the filefind function is returned.

Return Value

An integer value representing the size of the indicated file is returned, or a value of -1 is returned if the specified file could not be found.

See also

FileFind, FileTime, FileAttr

FileSize Example

FileTime Function

Example

FileTime(str filespec);

The filetime function determines the date and time stamp of a file. A date and time value in this format can be used by the <u>date</u>, <u>time</u>, <u>tyear</u>, <u>tmonth</u>, <u>tday</u>, <u>thour</u>, <u>tmin</u>, <u>tsec</u>, as well as other functions

Argument	Description
str filespec	The name of the file
	that may include a
	drive and directory, as
	well as the DOS
	wildcard characters *
	and ?. If empty (), the
	size of the last file
	found with the <u>filefind</u>
	function is returned.

Return Value

The value returned represents the files modification date as the number of seconds since January 1, 1970. If the file cannot be found, a value of -1 is returned.

See also

FileFind, FileSize, FileAttr



FileTime Example

```
main()
  int ftime;
  str s[16];
  ftime = filetime("TELIX.EXE");
  if (ftime == -1)
    prints("'TELIX.EXE" could not be found!");
  else
   printsc("TELIX.EXE was created at ");
   time(ftime, s);
   printsc(s);
    printsc(" on ");
   date(ftime, s);
   printsc(s);
  // this example assumes both files exist
  if (filetime("FILE1") < filetime("FILE2"))</pre>
    prints("FILE1 is older than FILE2");
  else
    prints("FILE1 is newer than FILE2");
```

FlushBuf Function

FlushBuf();

The flushbuf function flushes (throws away) any characters that may be waiting in Telix's input (receive) buffer. One use for this command is to discard characters before starting a file transfer.

fnStrip Function Example

fnStrip(str filename, int specifier, str target);

The fnstrip function allows specific parts of a filename to be extracted. In the MS-DOS and Windows operating systems, a filename can consist of up to four parts: the drive, the path, the name, and the extension.

Argument	Description
str filename	The filename to be
	processed.
int specifier	Determines what
	portions of the
	filename are to be
	extracted, according to
	the table below.
str target	The variable to hold
	the specified sections
	of the filename.

Specifier	Description
0	Full file name.
1	All except the drive.
2	Drive, name, and
	extension.
3	Name and extension.
4	Drive, path, and name
	(no extension).
5	Path and name (no
	extension).
6	Drive and name (no
	extension).
7	Name only (no
	extension).
12	Drive and path only.
13	Path only.
14	Drive only.

Return Value

A zero is always returned.

See also

<u>FileFind</u>

fnStrip Example

fOpen Function

Example 4 1

fOpen(str filename, str mode);

The fopen function is used to open or create a disk file for reading and/or writing. Only eight files may be open at one time, therefore it is very important to close files when they are no longer needed. If a file is opened for both reading and writing (when "r+", "w+", or "a+" are used as the mode), an <u>fseek</u> operation is necessary before switching from reading and writing.

Argument	Description
str filename	The file to be opened
	or created.
str mode	Specifies what type of
	operations are to be
	allowed. See the
	following table for legal values.
	values.

ading.
iting,
current
or
ting at
If the
ist, it is
eading
itial
е
e file,
st.
eading
stroying
itents.
eading If the ist, it is

Return Value

The value returned is a **handle**, which is simply an integer number, by which the file is to be referred to in all subsequent file operations. A value of zero (FALSE) is returned if the file can not be opened.

See also

fClose



fOpen Example

```
int f;
str s[200];
f = fopen("test.dat", "r");
if (!f)
 prints("Could not open data file.");
else
 {
  while (!feof(f))
                           // print out contents of text file
    fgets(s, 200, f);
    if (!fError(f))
     prints(s);
    else
     prints("An error has occurred.");
     break;
   }
  fclose(f);
 }
```

fPutC Function

Example

fPutC(int c, int fh);

The fputc function writes a character to a file.

Argument	Description
int c	The character to write
	to the file.
int fh	The file handle
	specifying the
	previously opened file
	to write to.

Return Value

The character written is returned, unless there is an error, in which case a value of -1 is returned.

See also

fPutS, fGetC



FPutC Example

fPutS Function

Example

fPutS(str s, int fh);

The fputs function writes a string to a file. Characters are written from the string until a zero (0) value is encountered, but the zero is not written. All SALT strings end with a zero.

Argument	Description
str s	The string to be
	written. It must not
	be more than 512
	bytes in length.
int fh	The file handle
	specifying the
	previously opened file
	to write to.

Return Value

A 0 value is returned if the write is successful, a non-zero value if it is not.

See also

fPutC, fGetS

fPutS Example

fRead Function

Example

fRead(var buf, int count, int fh);

The fread function reads a specified number of bytes from a file into a variable.

Argument	Description
var buf	Either a string or
	integer variable. The
	variable must be large
	enough to hold values
	read.
int count	The number of bytes
	to read from the file.
	If <i>buf</i> is an integer
	type, <i>count</i> must be 4
	or less. Greater
	values will be
	truncated.
int fh	The file handle
	specifying the
	previously opened file
	to write to.

Return Value

The number of bytes actually read is returned, which may be less than asked for if an error occurs or an end of file (EOF) is encountered. The <u>ferror</u> and <u>feof</u> functions should be used to distinguish an error from an end of file (EOF) condition.

See also

fWrite



fRead Example

```
int f, i;
str buffer[40];

f = fopen("test.dat", "r");
fseek(f, 1000, 0);  // goto offset 1000 in file
fread(buffer, 40, f);  // and read 40 bytes of data into buffer
fread(i, 4, f);  // read an integer into i
fread(i, 2, f);  // read least significant bytes of an integer into i
fclose(f);
```

fRename Function

Example

fRename(str oldname, str newname);

The frename function is used to rename a disk file. Renamed files can not change drives or paths, only file names and extensions.

Argument	Description
str oldname	The name of the file to
	be renamed, which
	may include a drive
	and path.
str newname	The new name of the
	file, which should not
	include a drive and
	path.

Return Value

If successful, frename returns a 0 value, otherwise a non-zero value is returned.

See also

fDelete

fRename Example

```
if (frename("C:\TFW\TELIX.CAP", "OLDTLX.CAP") == 0)
  prints("C:\TFW\TELIX.CAP renamed to C:\TFW\OLDTLX.CAP");
else
  prints("Could not rename C:\TFW\TELIX.CAP");
```

fSeek Function

Example

fSeek(int fh, int offset, int origin);

The fseek function sets the position for reading and/or writing in an open file. The pointer can be positioned anywhere in the file, and even past the end of the file (which will extend it). It is illegal to try to position the pointer before the beginning of the file however.

Argument	Description
int fh	The file handle
	specifying the
	previously opened file
	to position.
int offset	The signed offset from
	the location specified
	in <i>origin.</i>
int origin	Beginning of file.
	 Current file
	position.
	2 End of file.

Return Value

If successful, fseek returns a 0 value, otherwise a non-zero value is returned.

See also

<u>fTell</u>



fSeek Example

fTell Function

fTell(int fh);

The ftell function returns the current file position of a file. This is usually the position where the next read or write operation will take place, however for a file opened in Append mode, the value returned will not necessarily return the position of the next write, since Append mode will force writes to the end of file regardless of the current file position.

Argument	Description
int fh	The file handle
	specifying the previously opened file
	to determine the
	position of.

Return Value

A -1 value is returned if an error occurs, otherwise a zero value is returned.

See also

<u>fSeek</u>

fWrite Function

Example

fWrite(var buf, int count, int fh);

The fwrite function writes bytes from a variable or constant to a file.

Argument	Description
var buf	An integer or string
	value to be written to
	the file.
int count	The number of bytes
	to write to the file. If
	buf is an integer type,
	count must be 4 or
	less. Greater values
	will be truncated.
int fh	The file handle
	specifying the
	previously opened file
	to write to.

Return Value

The number of bytes actually written are returned, which may be less than requested if an error occurred.

See also

<u>fRead</u>



fWrite Example

Get_Baud Function

Example

Get_Baud([int type]);

The get_baud function determines the baud rate in use by the current connect device (300 through 115,200).

Argument	Description
int type	If 1, the DCE is
	returned. If 0 or no
	parameter is passed,
	the DTE is returned.

Return Value

An integer value that represents the currently connected baud rate. Values range from 300 to 115,200, or -1 if the current port is invalid..

See also

Get_Parity, Get_DataB, Get_StopB, Get_Port

Get_Baud Example

```
prints("The current baud rate is ");
printn(get_baud());
prints("");
```

Get_DataB Function

Get_DataB();

The get_datab function determines the data bits setting in use on the current connect device.

Return Value

The current data bits setting, either 7 or 8, or -1 if the current port is invalid..

See also

Get_Baud, Get_Parity, Get_StopB, Get_Port

GetEnv Function

Example

GetEnv(str varname, str target);

The getenv function may be used to access the DOS Environment and get the value assigned to an Environment Variable.

Argument	Description
str varname	The name of the
	environment variable
	to retrieve. Case is
	not significant.
str target	The string variable to
	copy the environment
	variable into.

Return Value

A non-zero (TRUE) value is returned if the function is successful, otherwise a zero (FALSE) values is returned (if the environment variable didnt exist).

GetEnv Example

Get_Parity Function

Get_Parity();

The get_parity function returns an integer value which represents the current parity setting in use on the current connect device.

Return Value

- -1 Invalid port
- 0 No parity
- 1 Even parity
- 2 Odd parity
- 3 Mark parity
- 4 Space parity

See also

Get_Baud, Get_DataB, Get_StopB, Get_Port

Get_Port Function

Example

Get_Port();

The get_port function determines the number of the current communications port being used.

Return Value

The number of the communications port in use, from 1 to 4, or -1 if the current port is invalid.

See also

Get_Baud, Get_DataB, Get_Parity, Get_StopB

Get_Port Example

```
prints("Currently using COM");
printn(get_port());
prints("");
```

Get_StopB Function

Get_StopB();

The get_stopb function determines the stop bits setting in use on the current connect device.

Return Value

The current stop bits settings, either 1 or 2, or -1 if the current port is invalid..

See also

Get_Baud, Get_DataB, Get_Parity, Get_Port

GetS Function

Example

GetS(str buffer, int max);

The gets function allows the user to enter a string with editing capabilities. String entry is complete when the user presses Enter. The user may press Esc to abort string entry, in which case the resulting string will be empty ("").

Argument	Description
str buffer	The variable to store
	the entered string.
int max	The maximum number
	of characters that may
	be entered.

Return Value

The number of characters entered by the user are returned. If the user pressed Esc to abort string entry, a value of -1 is returned. If the cursor has been moved out of the terminal window (with the GotoXY function, for instance), a value of -2 is returned immediately.

See also

GetSXY

```
GetS Example
```

```
int n;
str password[8];
printsc("Enter a password? ");
n = gets(password, 8);
```

GetSXY Function

Example

GetSXY(str buffer, int max, int x, int y, int color);

The getsxy function is similar to the <u>gets</u> function, but the screen location of string entry may be specified, as well as a color attribute. String entry is complete when the user presses Enter. The user may press Esc to abort string entry, in which case the resulting string will be empty ("").

Argument	Description
str buffer	The variable to store
	the entered string.
int max	The maximum number
	of characters that may
	be entered.
int x	The column where
	entry will occur.
int y	The line where entry
	will occur.
int color	The <u>color</u> attribute of
	the editing area.

Return Value

The number of characters entered by the user are returned. If the user pressed Esc to abort string entry, a value of -1 is returned. If the values of *x*, *y* is not a valid screen location, a value of -2 is returned immediately.

See also

<u>GetS</u>

GetSXY Example

```
int n;
str filename[64] = "C:\TELIX\TELIX.EXE";

// allow user to enter filename in black on white
// at current cursor position
n = getsxy(filename, 64, getx(), gety(), 112);
```

GetTermHeight Function

<u>Example</u>

GetTermHeight();

The gettermheight function determines the number of lines available on the terminal screen. This function returns the absolute number of lines without regard to whether they are visible or not (if the terminal window is sized so that scrollbars are required to view all of it). Note that SALT functions that take screen coordinates as parameters use zero based coordinates. See the example for further information.

Return Value

An integer value representing the number of lines available; 24, 25 or 50.

See also

GetTermWidth, GetX, GetY



GetTermHeight Example

```
// Draw a box that covers the entire screen.
// Note that both gettermwidth and gettermheight have 1 subtracted
// from them because screen coordinates are zero based.
box(0, 0, gettermwidth()-1, gettermheight()-1, 1, 0, 30);
```

GetTermWidth Function

Example

GetTermWidth();

The gettermwidth function determines the number of columns available on the terminal screen. This function returns the absolute number of columns without regard to whether they are visible or not (if the terminal window is sized so that scrollbars are required to view all of it). Note that SALT functions that take screen coordinates as parameters use zero based coordinates. See the example for further information.

Return Value

An integer value representing the number of lines available, either 80 or 132.

See also

GetTermHeight, GetX, GetY



GetTermWidth Example

```
// Draw a box that covers the entire screen.
// Note that both gettermwidth and gettermheight have 1 subtracted
// from them because screen coordinates are zero based.
box(0, 0, gettermwidth()-1, gettermheight()-1, 1, 0, 30);
```

GetX Function

GetX();

The getx function determines the current column (horizontal x axis) position of the cursor on the screen. This function returns the absolute number of columns without regard to whether they are visible or not (if the terminal window is sized so that scrollbars are required to view all of it).

Return Value

Returned values will range from 0 for the leftmost column to 131 for the rightmost column.

See also

GetY, GotoXY

GetY Function

GetY();

The gety function determines the current row (vertical y axis) position of the cursor on the screen. This function returns the absolute number of columns without regard to whether they are visible or not (if the terminal window is sized so that scrollbars are required to view all of it).

Return Value

Returned values range from 0 for the upper edge of the screen to 49 for the lower edge.

See also

GetX, GotoXY

GotoXY Function

Example

GotoXY(int xpos, int ypos);

The gotoxy function positions the cursor at the given screen coordinates. Note that 0,0 is the upper left corner. On a 80x25 text screen, the lower right corner would be 79,24. If the given coordinates is a location that is not on screen, output will be suspended until it is placed back on screen. If output is suspended, processing of the communications port does continue, but there will be indication of it in the terminal window. Valid screen ranges can be determined using the GetTermHeight and GetTermWidth functions.

Argument	Description
int xpos	The column at which
int ypos	the cursor should be placed. The line at which the cursor should be placed.
Return Value	

A zero is always returned.

See also

GetX, GetY, GetTermHeight, GetTermWidth

GotoXY Example

```
// go to the top left corner
gotoxy(0, 0);

// go to the bottom right corner
gotoxy(gettermwidth()-1, gettermheight()-1);
```

Hangup Function

Hangup();

The hangup function tries to hang-up the modem, exactly as if the user had selected Hangup from the Actions menu or pressed Alt-H while in terminal mode. This is accomplished by first dropping (turning off) a signal called the DTR line, and if that is unsuccessful, sending the hang-up string defined in the device's configuration.

Return Value

A non-zero (TRUE) value is returned if the hang-up is successful, otherwise a zero (FALSE) value is returned.

HelpScreen Function

HelpScreen();

The helpscreen function displays the Telix for Windows help file.

Return Value

A value of zero is always returned.

InKey Function

Example

InKey();

The inkey function gets a character from the keyboard if one is ready, but does not wait for a key to be pressed. Also, inkey returns extended key code values which are not part of the ASCII character set (for example, the code for Alt-D which is 8192). These values are described in the Telix for Windows manual Appendix and in the Extended Key Scan Code section of this help file.

Return Value

The first character in the keyboard buffer, or a value of 0 if the keyboard buffer is empty.

See also

InKeyW

InKey Example

```
int chr;
prints("Press any key to continue...");
chr = 0;
while (!chr)
    chr = inkey();
```

InKeyW Function

Example

InKeyW();

The inkeyw function gets the next available character from the keyboard, or waits for a key to be pressed if the keyboard buffer is empty. Also, inkeyw returns extended key code values which are not part of the ASCII character set (for example, the code for Alt-D which is 8192). These values are described in the Telix for Windows manual Appendix and in the <u>Extended Key Scan Code</u> section of this help file.

Return Value

The first character to become available in the keyboard buffer.

See also

<u>InKey</u>

InKeyW Example

```
int chr;
prints("Press any key to continue...");
chr = inkeyw();
```

InputBox Function

<u>Example</u>

InputBox(str title, str prompt, str buff);

The inputbox function uses a stock input dialog box to acquire text input from the user. The dialog contains a standard Windows edit control for text input, and an Ok and Cancel button for completing input.

Argument	Description
str title	The caption to be
	displayed on the input
	boxs title bar.
str prompt	The text displayed
	above the edit field. It
	should be used to
	explain what input is
	being requested.
str buff	The string variable to
	hold the users input.
	If this variable
	contains a string when
	the inputbox function
	is called, it will be
	placed in the edit field
	initially.

Return Value

A value of 1 is returned if the Ok button was pressed, otherwise a value of 2 is returned. If Cancel is selected, the *buff* variable will **not** be updated.

See also

Status_Wind, MsgBox



InputBox Example

```
#CONST idOK
#CONST idCancel
#CONST mbOk
#CONST mblconStop 16
int retval;
str buf[80] = "John Doe";
if (inputbox("Query", "Enter your full name:", buf) == idOK)
 printsc("Entered Name = ");
 prints(buf);
else
  // note that buf would still contain "John Doe" at this point.
  prints("Entry was aborted.");
// The following input box will be displayed until Cancel was not
// chosen and the entered string is not empty.
while ((inputbox("Query", "Enter your full name:", buf) == idCancel) ||
strlen(buf) == 0)
  msgbox("Error", "You must enter your name.", mbOK | mbIconStop);
printsc("Entered Name = ");
prints(buf);
```

InsChrs Function

Example

InsChrs(str source, str target, int pos, int num);

The inschrs function is used to insert characters from one string into another at a specific position, shifting existing characters to the right. If there is not enough room for shifted characters, they will be lost.

Description
The string to insert
characters from.
The string to insert
characters into.
The position in the
target string to insert
characters. Note that
SALT strings begin at
0, not 1 as some
languages.
The number of
characters to insert
from the source string.

Return Value

A zero is always returned.

See also

CopyStr, CopyChrs

InsChrs Example

```
str test[24] = "Good-bye", test2[10] = "Hello ";
// add "Hello" to the front of the existing string
inschrs(test2, test, 0, strlen(test2));
```

IsAlNum Function

IsAlNum(int c);

The isalnum function tests an integer value to determine if it is a letter (A-Z, a-z) or numeric (0-9) value. isalnum will only give valid results for integer values in the ASCII character set; that is, values for which isascii is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument Description
The value to be tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlpha, IsAscii, IsCntrl, IsDigit, IsLower, IsUpper

IsAlpha Function

IsAlpha(int c);

The isalpha function tests an integer value to determine if the value is a letter (A-Z, a-z) value. isalpha will only give valid results for integer values in the ASCII character set; that is, values for which <u>isascii</u> is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument Description

Int c The value to be tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAscii, IsCntrl, IsDigit, IsLower, IsUpper

IsAscii Function

IsAscii(int c);

The isascii function tests an integer value to determine if it is an ASCII character (0-255). Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument	Description
int c	The value to be
	tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAlpha, IsCntrl, IsDigit, IsLower, IsUpper

IsCntrl Function

IsCntrl(int c);

The iscntrl function tests an integer value to determine if the value is a control character (0-31, 127). iscntrl will only give valid results for integer values in the ASCII character set; that is, values for which <u>isascii</u> is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument	Description
int c	The value to be
	tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAlpha, IsAscii, IsDigit, IsLower, IsUpper

IsDigit Function

IsDigit(int c);

The isdigit function tests an integer value to determine if it is a digit (0-9). isdigit will only give valid results for integer values in the ASCII character set; that is, values for which <u>isascii</u> is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument Description
int c The value to be tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAlpha, IsAscii, IsCntrl, IsLower, IsUpper

IsLower Function

IsLower(int c);

The islower function tests an integer value to determine if the value is a lower case letter (a-z). islower will only give valid results for integer values in the ASCII character set; that is, values for which <u>isascii</u> is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument Description
int c The value to be tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAlpha, IsAscii, IsCntrl, IsDigit, IsUpper

IsUpper Function

IsUpper(int c);

The isupper function tests an integer value to determine if the value is an upper case letter (A-Z). isupper will only give valid results for integer values in the ASCII character set; that is, values for which <u>isascii</u> is true. Note that the parameter may be specified as either the ASCII integer value, or as a character enclosed in single quotes ('').

Argument Description
int c The value to be tested.

Return Value

A non-zero (TRUE) value if the conditions are met, or a zero (FALSE) if it is not.

See also

IsAlNum, IsAlpha, IsAscii, IsCntrl, IsDigit, IsLower

Is_Loaded Function

Example

Is_Loaded(str filename);

The is_loaded function is used to determine if a SALT script is currently loaded in memory. The script can be in memory if it was explicitly loaded with the $\underline{load_scr}$ function, or is still in memory because it was previously executed and did not finish.

Argument	Description
str filename	The filename of the
	script to check. If it
	does not include an
	extension, .SLC is
	automatically
	appended.

Return Value

A non-zero (TRUE) value is returned if the script file is in memory, otherwise a zero (FALSE) value is returned.

See also

Load Scr, Unload Scr

Is_Loaded Example

```
if (!is_loaded("TESTSCR")) // make sure script is in memory
   load_scr("TESTSCR");
```

ItoS Function

Example

ItoS(int value, str buff);

The itos function converts an integer value to a string representation of the number.

Argument	Description
int value	The number to be
	converted.
str buff	The variable to place
	the resulting string in.

Return Value

The resulting string (buff) is returned.

See also

<u>stoi</u>

```
ItoS Example
```

KeyGet Function

KeyGet(int key, int table, str buffer);

The keyget function is used to retrieve the macro text that is assigned to a key.

Argument	Description
int key	An integer
	representing the key
	as described in the
	Telix for Windows
	manual Appendix and
	in the Extended Key
	Scan Code section of
	this help file).
Int table	This parameter does
	not apply to Telix for
	Windows, and it is
	included only for
	compatibility with Telix
	for DOS. If a value is
	specified, it will be
	ignored.
str buffer	Any macro text
	assigned to the key
	will be placed in this
	string.

Return Value

A zero is always returned.

See also

KeySet, KeyLoad, KeySave

KeyGet Example

```
str s[100];
prints("Text currently assigned to the F1 key in user table is:");
keyget(0x3b00, 0, s);
prints(s);
```

KeyLoad Function

KeyLoad(str filename, int table);

The keyload function is used to load a keyboard macro definition file into Telix.

Argument	Description
str filename	The filename of the
	keyboard definition
	file. If no extension is
	give, .KBD is
	assumed.
int table	This parameter does
	not apply to Telix for
	Windows, and it is
	included only for
	compatibility with Telix
	for DOS.

Return Value

A value of -1 is returned if there is a problem loading the key file, otherwise a non-zero (TRUE) value is returned.

See also

KeySave, KeyGet, KeySet

KeyLoad Example

```
if keyload("SPECIAL", 0)
  prints("Keyboard file loaded.");
else
  prints("Keyboard file could not be loaded.");
```

KeySave Function

KeySave(str filename, int table);

The keysave function is used to save the current macro key definitions to a disk file.

Argument	Description
str filename	The filename of the
	keyboard definition
	file. If no extension is
	give, .KBD is
	assumed.
int table	This parameter does
	not apply to Telix for
	Windows, and it is
	included only for
	compatibility with Telix
	for DOS.

Return Value

A value of -1 is returned if there is an error writing to the file, otherwise a non-zero (TRUE) value is returned.

See also

KeyLoad, KeyGet, KeySet

KeySet Function Example

KeySet(int key, int table, str text);

The keyset function is used to assign keyboard macro text to a key.

Argument	Description
int key	An integer
	representing the key
	as described in the
	Telix for Windows
	manual Appendix and
	in the Extended Key
	Scan Code section
	of this help file).
int table	This parameter does
	not apply to Telix for
	Windows, and it is
	included only for
	compatibility with Telix
	for DOS.
str text	The macro text to be
	assigned to the key.
Return Value	
- Value	

A zero is always returned.

See also

KeyGet, KeyLoad, KeySave

KeySet Example

```
// Assign a name to the F1 key in the user table
// Note that if the current keyboard macro table
// already has a definition for that key it will
// be replaced by this one.
keyset((0x3b00, 0, "Joe Smith^M");
```

LoadFon Function

LoadFon(str filename);

The loadfon function loads a PhoneBook into Telix for Windows.

Argument	Description
str filename	The complete filename
	of the PhoneBook to
	load, including any
	extension (e.gFBK),
	as well as the disk
	drive and directory if
	the file is not in the
	current directory.

Return Value

A non-zero (TRUE) value is returned if the dialing directory file is successfully loaded. If some sort of error occurs (file does not exist, file reading error, etc.) a zero (FALSE) value is returned.

Load_Scr Function

Example

Load_Scr(str filename);

When a script is executed, either by the user manually running it, or from within another script, it is usually loaded from disk. The load_scr function is used to load a script into memory ahead of time, providing a savings in time when the script must be run repeatedly. After a script is loaded in this manner, running it with the <u>calld</u> function makes a copy of the script in memory instead of loading it from disk. The script will remain in memory until released by the <u>Unload_Scr</u> function.

Argument	Description
str filename	The filename of the
	script to load. If no
	extension is
	given, .SLC is
	assumed.

Return Value

If there is an error loading the script file (it is not found or there is not enough memory), a value of -1 is returned.

Otherwise a non-zero (TRUE) value is returned.

See also

Unload Scr, Is Loaded

Load_Scr Example

```
int stat;
stat = load_scr("TEST");  // load TEST.SLC
```

MsgBox Function

MsgBox(str title, str message, int options);

The msgbox function creates, displays, and operates a message-box window. The message box contains the specified message and title, plus any combination of predefined icons and push buttons described below.

Argument	Description
str title	The caption to be
	displayed on the
	message boxs title
	bar.
str message	The message to be
	displayed. The
	Control-M (ASCII #13)
	character may be
	embedded to indicate
	line breaks.
int options	A combination of
	options, listed in the
	table below, that
	specifies the contents
	and behavior of the
	dialog. The options
	are combined using
	the OR () operator.
	See the Example for
	more information.

Option	Description
0	Ok button.
1	Ok and Cancel button.
2	Abort, Retry and
	Ignore button.
3	Yes, No and Cancel
	button.
4	Yes and No button.
5	Retry and Cancel
	button.
16	Hand icon.
32	Question icon.
48	Exclamation icon.
64	Information icon.
256	Second button
	displayed is the
	default.
512	Third button displayed
	is the default.
Return Value	Description
1	Ok button was
-	selected.
2	Cancel button was

	selected.
3	Abort button was selected.
4	
4	Retry button was
	selected.
5	Ignore button was
	selected.
6	Yes button was
	selected.
7	No button was
•	selected.
	sciecteu.

See also

Status_Wind, InputBox



MsgBox Example

```
// Button constants.
#CONST mbOk
#CONST mbOkCancel
#CONST mbAbortRetryIgnore 2
#CONST mbYesNoCancel
#CONST mbYesNo
                           4
#CONST mbRetryCancel
                          5
// Default Button constants.
#CONST mbDefButton1
#CONST mbDefButton2
                          256
#CONST mbDefButton3
                          512
// Info bitmap constants.
#CONST mbIconHand
                          16
#CONST mblconStop
                          16
#CONST mbIconQuestion
                          32
#CONST mbIconExclamation
                          48
#CONST mbIconAsterisk
                           64
#CONST mbIconInformation
                           64
// Return value constants.
#CONST idOk
#CONST idCancel 2
#CONST idAbort 3
#CONST idRetry 4
#CONST idIgnore 5
#CONST idYes
#CONST idNo
int retval;
// Display a simple message with an OK button.
MsgBox("Error:", "Telix was unable to establish a connection.", mbOK);
// Display a more complex message and
// take appropriate action based on response.
retval = MsqBox("Error:", "Telix was unable to establish a connection.",
mbAbortRetryIgnore | mbDefButton2 | mbIconStop);
switch (retval)
  case idIgnore:
   break; // take no action.
  case idRetry:
   // code to attempt reconnection.
   break;
  case idAbort:
    // code to abort attempt.
    break;
 }
```

NewDir Function

Example

NewDir(str directory);

The newdir function is used to change the current drive and/or directory.

Argument	Description
str directory	The drive and/or
	directory to change to.

Return Value

A non-zero (TRUE) value is returned if the function is successful, otherwise a zero (FALSE) value is returned (if the driver or directory is illegal or doesn't exist).

See also

Dos, Run

NewDir Example

newdir("C:\TFW\FONBOOKS");
loadfon("MYPHONE.FBK");

NewLine Function

'	, ,	
NewLine	(٠ ١
INC M TITLE	١.	, ,

The newline function is used to send a carriage return and line feed to the terminal.

Return Value
A zero is always returned.

See also

CNewLine

NumConnectDevices Function

Example

NumConnectDevices();

The numconnectdevices is used to determine how many connect devices the user has installed. Connect devices are installed and configured from Telix's Configure menu option, Connect Devices.

Return Value

The number of connect devices is returned.

See also

ConnectDeviceName, Set_ConnectDevice



NumConnectDevices Example

```
int num;
str buff[30];

for (num = 1; num <= numconnectdevices; ++num)
   prints(connectdevicename(num, buff));</pre>
```

PlayWave Function

Example

PlayWave(str filename);

The playwave function plays a waveform sound from a file or an entry in the [SOUNDS] section of the WIN.INI file.

Argument

Description

str filename

A complete filename or an entry in the [SOUNDS] section of WIN.INI file. If the sound can't be found, the default sound specified by the SystemDefault entry in the [SOUNDS] section of WIN.INI is played. If there is no SystemDefault entry or if the default sound can't be found, the function makes no sound.

Return Value

A non-zero (TRUE) value is returned if the function is successful, otherwise a zero (FALSE) value is returned (if filename was not a valid wave file or entry in the [sounds] section of WIN.INI).

See also

Alarm, Tone, alarm on, sound on



```
playwave("c:\win\chimes.wav");  // Plays the specified file
playwave("SystemHand");  // Plays the Windows Critical Stop sound
```

PrintC Function

Example

PrintC(int chr);

The printc function prints a character represented by an ASCII value to the terminal screen.

Argument Description
The ASCII value of the character to be printed.

Return Value

The value to be printed is returned.

See also

PrintC_Trm, PrintN, PrintN_Trm, PrintS, PrintS_Trm, PrintSC, PrintSC_Trm

PrintC Example

PrintC_Trm Function

<u>Example</u>

PrintC_Trm(int chr);

The printc_trm function prints the character represented by an ASCII value to the terminal screen. This function is the same as <u>printc</u>, except that the character passes through the current terminal emulator, so terminal escape sequences can be used.

Argument	Description
int chr	The ASCII value of the
	character to be
	printed.

Return Value

The value to be printed is returned.

See also

PrintC, PrintN, PrintN_Trm, PrintS, PrintS_Trm, PrintSC, PrintSC_Trm

PrintC_Trm Example

Printer Function

Printer(int state);

The printer function is used within a script file to turn the printer log on or off, as if the user had selected Printer Log from the file menu or pressed the appropriate key in terminal mode.

Argument	Description
int state	If zero (FALSE), the
	printer log will be
	turned off. If non-zero
	(TRUE), it will be
	turned on.

Return Value

A zero (FALSE) value is returned if the log cannot be started, otherwise a non-zero (TRUE) value is returned.

See also

Capture

PrintN Function

Example

PrintN(int num);

The printn function prints an integer number to the screen. The cursor is **not** advanced to the beginning of the next line.

Argument Description
The number to be printed.

Return Value

The value to be printed is returned.

See also

PrintC, PrintC Trm, PrintN Trm, PrintS, PrintS Trm, PrintSC, PrintSC Trm

PrintN Example

```
printsc("Current baud rate is ");
printn(get_baud(1));
```

PrintN_Trm Function

Example

PrintN_Trm(int num);

The printn function prints an integer number to the screen. The cursor is **not** advanced to the beginning of the next line. printn_trm works in the same manner as <u>printn</u>, except that the number passes through the current terminal emulator, so terminal escape sequences may be used.

Argument	Description
int num	The number to be printed.

Return Value

The value to be printed is returned.

See also

PrintC, PrintC Trm, PrintN, PrintS, PrintS Trm, PrintSC, PrintSC Trm

PrintN_Trm Example

```
printsc("Current baud rate is ");
printn_trm(get_baud(1));
```

PrintS Function

Example

PrintS(str outstr);

The prints function prints a string at the current cursor position on the screen, followed by a Carriage Return and Line Feed (which advances the cursor at the beginning of the next line).

Argument	Description
str outstr	The string to be printed.
Return Value	
A zero is alway	ys returned.

See also

PrintC, PrintC_Trm, PrintN, PrintN_Trm, PrintS_Trm, PrintSC, PrintSC_Trm



```
// Print Hello at the current position and advance to beginning of next line.
prints("Hello");

// Print Goodbye on the line after the one Hello was printed on.
prints("Goodbye");
```

PrintS_Trm Function

Example

PrintS Trm(str outstr);

The prints function prints a string at the current cursor position on the screen, followed by a Carriage Return and Line Feed (which advances the cursor at the beginning of the next line). prints_trm works the same as <u>prints</u>, except that the string passes through the current terminal emulator, so terminal escape sequences may be used.

Argument	Description
str outstr	The string to be printed.
Return Value	
A zero is always	returned.

See also

PrintC, PrintC_Trm, PrintN, PrintN_Trm, PrintS, PrintSC, PrintSC_Trm

PrintS_Trm Example

```
// Print Hello at current position and advance to next line.
prints_trm("Hello");

// Print Hello at current position and advance two lines.
prints_trm("Hello^M^J");

// go to top left corner in VT102 emulation and print Hello.
prints_trm("^[[H Hello");
```

PrintSC Function

Example 4 1

PrintSC(str outstr);

The printsc function prints a string at the current cursor position on the screen, but does not advance the cursor to the next line, hence the 'c', which stands for continuous.

Argument	Description
str outstr	The string to be printed.
Return Value	
A zero is always	s returned.

See also

PrintC, PrintC_Trm, PrintN, PrintN_Trm, PrintS, PrintS_Trm, PrintSC_Trm

PrintSC Example

```
// The following will print "Hello and Goodbye" at the current position.
printsc("Hello");
printsc(" and ");
printsc("Goodbye");
```

PrintSC_Trm Function

Example 4 1

PrintSC_Trm(str outstr);

The printsc_trm function is similar to <u>printsc</u>, except that the string passes through the current terminal emulator, so terminal escape sequences may be used.

Argument	Description
str outstr	The string to be printed.
Return Value	
A zero is alway	s returned.

See also

PrintC, PrintC Trm, PrintN, PrintN Trm, PrintS, PrintS Trm, PrintSC



PrintSC_Trm Example

```
// go to top left corner in VT102 emulation and print "Yello".
printsc_trm("^[[H"); // Sequence for top left corner.
printsc_trm("Hello"); // Print Hello.
printsc_trm("^[[H"); // Back to top left again.
printsc_trm("Y"); // Print Y where H was.
```

PStrA Function

Example

PStrA(str s, int color);

The pstra (Print STRing with color Attribute) function is used to print a string on the screen, similar to the prints and prints functions. This function is much faster however, and should be used when speed is important. The pstra function also allows a color to be specified for the text.

Argument	Description
str s	The string to be
	printed.
int color	The <u>color</u> of the string to be printed.

Return Value

A zero is always returned.

See also

PrintS, PrintSC, PStrAXY



pstra("Enter name:", 112); // print in black on white.

PStrAXY Function

Example

PStrAXY(str s, int x, int y, int color);

The pstraxy (Print STRing with color Attribute at X,Y) function is used to print a string to the screen, similar to the <u>prints</u> and <u>printsc</u> functions. This function is much faster however, and should be used when speed is important. The pstraxy function also allows a <u>color</u> to be specified for the text, as well as a position on the terminal screen.

Argument	Description
str s	The string to be
	printed.
int x	The column to print
	the string at.
int y	The line to print the
	string at.
int color	The <u>color</u> of the
	string to be printed.

Return Value

A zero is always returned.

See also

PrintS, PrintSC, PStrA



pstraxy("Enter name:", 10, 10, 30); // print at 10, 10 in yellow on blue.

Random Function

Random(int range);

The random function returns a random number within a specified range.

Argument	Description
int range	The maximum number of the range plus 1.

Return Value
A number in the range 0 <= X < range.

Receive Function

<u>Example</u>

Receive(int protocol, str name);

The receive function is used to receive (download) one or more files from another system. If a download directory has been defined in the Filenames & Paths configuration, received files will be placed there, unless the *name* string explicitly includes a path to another drive and/or directory.

Argument	Description
Int protocol	The protocol to
	receive the file(s) with.
	See table below.
str name	The name for the
	file(s) being received.
	For protocols which
	pass filenames, such
	as ZModem, YModem
	(batch), and others,
	this should be an
	empty string ("").

Protocol	Description
'A'	ASCII
'X'	XModem
'1'	XModem-1k
'G'	XModem-1k-G
Ύ'	YModem
'E'	YModem-G
'Z'	ZModem

Return Value

A value of -1 is returned if the transfer was aborted, -2 if the carrier (connection) was lost, or 0 if successful.

See also

Send, down dir

Receive Example

Redial Function

Example

Redial(str dialstr, int maxtries, int no link);

Redial is used much the same as the <u>dial</u> function. It redials the previously entered dialing queue, optionally adding new numbers to it, and it allows control over the number of attempts and whether or not to execute linked scripts.

Argument	Description
str dialstr	A string containing the entries to be added to the dialing queue. Entries can be specified as the PhoneBook entry number, the PhoneBook entry name, or partial name, enclosed in pipe () symbols, or a manually entered number prefaced by an 'm'. If an actual pipe symbol is required, use a double pipe (). If dialstr is empty (""), then no entries are added to the queue before dialing.
int maxtries	The maximum number of dialing attempts to make. This is the total number of attempts, regardless of the number of entries being dialed. For instance, if three entries are to be dialed and <i>maxtries</i> is 6, each entry will be attempted twice. If <i>maxtries</i> is 0, dialing will continue
int no_link	until a connection is established. If non-zero (TRUE) and the entry connected has a linked script file, that script will NOT be executed. If no_link is zero (FALSE), linked scripts will be executed.

Return Value

If a connection is established, the redial function returns the entry number that was connected to (or 1 if a manual number was dialed). If no connection was established, 0 is returned. If *dialstr* is incorrectly formatted, -1 is returned.

When a connection is successfully established, several items of information regarding the entry are placed in <u>System Variables</u>. These variables are fully described in the <u>entry info</u> section.

See also

Dial, entry enum, entry name



```
int stat;
str number_list[255];
// The first dial will dial entry #10, the first entry with "delta" in it,
// entry #15, and My Pipe | BBS until a connection has been established,
// but do not execute any linked scripts.
if (dial("10 |delta| 15 |My Pipe || BBS|", 0, 1) > 0)
   connected_func();
while (redial("", 0, 1) > 0)
   connected_func();
```

Run Function

Example

Run(str filename, str comline, int mode);

The run function is used to execute another program. Ensure that if you run a program that expects user input, you are on hand to provide it. This function is similar to the <u>dos</u> function, but it can launch both Windows and DOS programs. Therefore, it is preferable unless a DOS batch file has to be run, or an internal DOS command must be executed, in which case the dos function has to be used.

Argument	Description
str filename	The filename of the
	program to launch. It
	must either be in the
	current directory, the
	path, or must include
	the full path to the file
	(i.e., specify the drive
	and/or directory).
str comline	Parameters to be
	passed to the called
	program.
Int mode	This parameter does
	not apply to Telix for
	Windows, and it is
	included only for
	compatibility with Telix
	for DOS. If a value is
	given, it will be
	ignored.

Return Value

A -1 is returned if the file can not be launched (because it can not be found or there is not enough memory), otherwise 0 is returned.

See also

Dos, DosFunction



```
// Launch Windows' NotePad program with CONFIG.SYS loaded. run("NOTEPAD.EXE", "C:\CONFIG.SYS", 0);
```

Scroll Function

Example

Scroll(int x, int y, int x2, int y2, int lines, int color);

The scroll function is used to scroll or clear a region of the screen.

Description
The left column of the
area.
The upper row of the
area.
The right column of
the area.
The lower row of the
area.
The number of lines to
scroll the area. If
positive, the area is
scrolled up. If
negative, the area is
scrolled down. If
zero, the entire region
is cleared.
The <u>color</u> of empty
lines scrolled into the
area.

Return Value

A zero is always returned.

See also

<u>Box</u>

Scroll Example

// scroll the entire screen up 10 lines with a color of yellow on blue.
scroll(0, 0, gettermwidth()-1, gettermheight()-1, 10, 30);

ScriptVersion Function

ScriptVersion();

The scriptversion function is used to determine the version of the Telix script compiler.

Return Value

If the compiler is Telix for DOS v3.22 or below (SALT), a 0 is returned. If it is Telix for Windows v1.00 or 100% TFW compatible version of Telix for DOS (SALT II), a 1 is returned.

See also

TelixForWindows, TelixVersion, SALTII Symbol, WINDOWS Symbol

Send Function

Example

Send(int protocol, str filename);

The send function is used to send (upload) one or more files from another system. If an upload directory has been defined in the Filenames & Paths configuration, files will be sent from there unless the *filename* string explicitly includes a path to another drive and/or directory.

Argument	Description
int protocol	The protocol to send
	the file(s) with. See
	table below.
str name	The name of the file(s)
	to be sent. It may
	include the DOS
	wildcard characters *
	and ?, in which case
	all matching files will
	be sent, however the
	protocol used must be
	capable of sending
	more than one file at a
	time, e.g. Zmodem,
	Ymodem, etc.

Protocol	Description
'A'	ASCII
'X'	XModem
'1'	XModem-1k
'G'	XModem-1k-G
Ύ'	YModem
'E'	YModem-G
'Z'	ZModem

Return Value

A value of -1 is returned if the transfer was aborted, -2 if the carrier (connection) was lost, -3 if there were no files to upload, or 0 if successful.

See also

Receive, up_dir

```
Send Example
```

```
int result;
// Upload TEST.EXE from the upload directory and C:\CONFIG.SYS using YModem.
result = send('Y', "TEST.EXE C:\CONFIG.SYS");
if (result < 0)
   prints("File transfer failed!");</pre>
```

Send_Brk Function

Send_Brk(int duration);

The send_brk function sends a sustained break signal over the connect device for a period of time.

Argument	Description
int duration	The amount of time to send the break signal in tenths of seconds.
Return Value	
A zero is always returned.	

Set_ConnectDevice Function

Set_ConnectDevice(str devicestr);

The set_connectdevice function is used to select a device for communications use. If the requested device can not be found, the connect device is not changed.

Argument	Description
str devicestr	The complete name of an existing connect device as entered in the Connect Device
	Manager.

Return Value

A value of 1 (TRUE) is returned if successful, otherwise a 0 (FALSE) is returned.

See also

ConnectDeviceName, NumConnectDevices, Set_CParams

Set_CParams Function

Example

Set_CParams(int baud, int parity, int data, int stop);

The set_cparams function is used to set the communications parameters in use on the current connect device.

Argument	Description
int baud	Baud rate to use.
	Legal values are 300,
	1200, 2400, 4800,
	9600, 19200, 38400,
	57600, 115200.
int parity	Parity to use. Legal
	values are 0 for None,
	1 for Odd, 2 for Even,
	3 for Mark, and 4 for
	Space.
int data	Data bits to use.
	Legal values are 7 and
	8.
int stop	Stop bits to use.
	Legal values are 1 and
	2.

Return Value

If all the settings are legal values, a non-zero (TRUE) value is returned, otherwise a value of -1 is returned.

See also

Set_ConnectDevice



Set_CParams Example

```
// Set 2400 baud, No parity, 8 data bits and 1 stop bit.
set_cparams(2400, 0, 8, 1);

// Change to 9600 baud without disturbing the parity, data or stop bits.
set_cparams(9600, get_parity(), get_datab(), get_stopb());
```

Set_DefProt Function

Set_DefProt(int protocol);

The set_defprot function is used to set the default file transfer protocol presented to the user when a file transfer is requested.

Argument int protocol	Description Default protocol. following table.	See
Protocol	Description	
'A'	ASCII	
'X'	XModem	
'1'	XModem-1k	
'G'	XModem-1k-G	
'Y'	YModem	
'E'	YModem-G	
'Z'	ZModem	
Return Value		
A zero is alway	s returned	

A zero is always returned.

See also

Receive, Send

Set_Port Function

Set_Port(int port);

The set_port function is not used in Telix for Windows, and it is included only for compatibility with Telix for DOS. The <u>Set_ConnectDevice</u> function should be used instead.

See also

Set_ConnectDevice, Set_cParams

Set_Terminal Function

Set_Terminal(str terminal_name);

The set_terminal function is used to change terminal emulation.

Argument	Description
str	The terminal type to be
terminal_nam	used. Legal values are:
е	"TTY", "ANSI-BBS",
	"ANSI", "VT220",
	"VT102", "VT100",
	"VT52", "AVATAR", and
	"RIP".

Return Value

A value of -1 is returned if there is a problem switching to the indicated terminal emulator, otherwise a non-zero (TRUE) value is returned.

SetChr Function

Example

SetChr(str buf, int pos, int c);

The setchr function replaces a character at a specific position in a string.

Argument	Description
str buf	The string to place the
	character in.
int pos	The position in the
	string to place the
	character. Any
	characters at this
	position will be
	overwritten.
int c	The character to place
	in the string.

Return Value

The integer value of the character to be set is returned.

See also

SetChrs, SubChr

SetChr Example

SetChrs Function

Example

SetChrs(str buf, int pos, int c, int count);

The setchrs function is used to set a range of characters in a string to the same value.

Argument	Description
str buf	The string to place the
	characters in.
int pos	The position in the
	string to start placing
	characters. Any
	existing characters
	starting at this position
	will be overwritten.
int c	The character to place
	in the string.
int count	The number of
	characters to be set.

Return Value

A zero is always returned.

See also

SetChr, SubChrs

SetChrs Example

```
str s[100];
// zero out an entire string
setchrs(s, 0, 0, strmaxlen(s));
// set the first ten characters to 'A'
setchrs(s, 0, 'A', 10);
```

Show_Directory Function

Show_Directory(str filespec, int cecho, int carrier);

The Show_Directory function is not used in Telix for Windows, and it is included only for compatibility with Telix for DOS.

Status_Wind Function

Example

Status Wind(str message, int duration);

The status_wind function is used to display a message in a pop up window. The window displayed includes an OK button, which the user can press to remove the window, and has room for up to four lines of centered text.

Argument	Description
str message	The message to be
	displayed. The
	Control-M (ASCII #13)
	character may be
	embedded to indicate
	line breaks.
int duration	The amount of time to
	display the message
	in tenths of seconds.

Return Value

A zero is always returned.

See also

MsgBox, InputBox, Box, PStrA, PStrAXY



Status_Wind Example

```
str msg[100] = "This is line 1.; This is line 2.; This is line3.; This is line
4."
int x;

// Display window for 3 seconds.
status_wind("This is a simple status message", 30);

// Using the string above, replace all semicolons with ASCII 13 (Carriage Return).
x = strchr(msg, ';', 0);
while (x != -1)
{
    setchr(msg, x, 13);
    x = strchr(msg, ';', x);
}

// Now that Carriage Returns are in place, the status window will display
// the text on 4 separate lines.
status_wind(msg, 30);
```

Stol Function

Example

StoI(str s);

The stoi function converts a string to its numeric equivalent. Processing stops at the first non-digit character.

Argument Description

str s The string to convert to a number.

Return Value

The numeric representation of the string is returned. If an empty or invalid string is specified, a value of 0 is returned.

See also

<u>ItoS</u>

Stol Example

```
str s[] = "123";
if (stoi(s) == 123)
  prints("This will always be printed!");
```

StrCat Function

StrCat(str string1, str string2);

The streat function concatenates, or appends, one string to the other.

Argument	Description
str string1	The string to append
	to. If it is not large
	enough to hold the
	resulting string, only
	as many as will fit will
	be added.
str string2	The string to be
	appended.

Return Value
The *target* string is returned.

StrCat Example

```
str s[80] = "hello";
strcat(s, "good-bye");
if (s == "hellogood-bye")
  prints("This will always be printed");
```

StrChr Function

Example

StrChr(str s, int pos, int c);

The strchr function is used to search for a character within a string.

Argument	Description
str s	The string to be
	searched.
int pos	The position in the
	string to begin the
	search.
int c	The character to
	search for.

Return Value

If the character is found, its location is returned, otherwise a value of -1 is returned.

See also

StrPos, StrPosl



StrChr Example

```
// Count how many times a certain char occurs in a string
int i, count = 0;
str s[] = "abcabcabcabcabc";
i = -1;
do
  ++i;
 i = strchr(s, i, 'a');
 if (i != -1)
    count = count + 1;
 }
while (i !=-1);
printsc("The character 'a' occurs ");
printn(count);
printsc(" times in the string '");
printsc(s);
prints("'.");
```

StrCmpl Function

Example

StrCmpI(str string1, str string2);

The strcmpi function is used to compare two strings without regard to case. The strings are compared character by character until a difference is found or the end of either string is found.

Argument	Description
str string1, str	The strings to be
string2	compared.

Return Value

An integer value is returned as follows:

int = 0 The strings are the same.
 int < 0 The first string is less than the second string.
 int > 0 The first string is greater than the second string.

StrCmpl Example

```
if (strcmpi("HeLLo", "hEllO"))
  prints("This will always be printed");
```

StrLen Function

Example

StrLen(str s);

The strlen function determines the number of characters in a string. Since strings are terminated with a 0 (NULL) character, this function really counts the number of characters before a 0 is encountered.

Argument	Description
str s	The string to
	determine the length
	of.

Return Value

The length of characters in the string is returned.

See also

StrMaxLen

StrLen Example

```
str teststr[] = "This is a test string";
printsc("The length of 'teststr' is ");
printn(strlen(teststr));
```

StrLower Function

StrLower(str s);

The strlower function converts all upper case characters in a string to lower case. Characters other than upper case are left unchanged.

Argument Description
Str s The string to be converted.

Return Value

The converted string is returned.

See also

StrUpper

StrMaxLen Function

StrMaxLen(str s);

The strmaxlen function determines the maximum number of characters that a string can hold. This is the same value as used when the string was defined elsewhere in the program (e.g. if the string was defined as 'str hello[16];', a value of 16 would be returned). All strings are really one character larger than defined, as the last character is always a terminating 0 (NULL). However, since this value can not be changed, it is not counted as part of the length of a string.

Argument	Description
str s	The string to
	determine the
	maximum length of.

Return Value

The maximum number of characters that can fit in the string, excluding the terminating 0 (NULL).

See also

StrLen

StrPos Function

Example

StrPos(str string1, str substr, int start);

The strpos function is used to search for one string within another.

Argument	Description
str string1	The string to be
	searched.
str substr	The string to search
	for.
int start	The position in the
	string to begin
	searching.

Return Value

If the substring is found, its location is returned, otherwise a value of -1 is returned.

See also

<u>StrPosl</u>



StrPos Example

StrPosl Function

Example

StrPosI(str string1, str substr, int start);

The strposi function is used to search for one string within another, without regard to case.

Argument	Description
str string1	The string to be
	searched.
str substr	The string to search
	for.
int start	The position in the
	string to begin the
	search.

Return Value

If the substring is found, its location is returned, otherwise a value of -1 is returned.

See also

<u>StrPos</u>



StrPosl Example

StrUpper Function

StrUpper(str s);

The strupper function converts all lower case characters in a string to upper case. Characters other than lower case are left unchanged.

Argument	Description
str s	The string to be
	converted.
Return Value	
The converted	string is returned.

See also

<u>StrLower</u>

SubChr Function

Example

SubChr(str s, int pos);

The subchr function returns a character from a specified location of a string.

Argument	Description
str s	The string containing
	the character.
int pos	The position in the
-	string of the character.

Return Value

The integer value of the requested character is returned.

See also

SetChr, SubChrs



SubChr Example

```
// This will print out the contents of a test string, extracting
// each character individually, and stopping when a 0 is reached
// which marks the end of all proper strings
int i;
str s[] = "This is a test string";
for (i = 0; subchr(s, i) != 0; ++i)
   printc(subchr(s, i));
```

SubChrs Function

SubChrs(str source, int pos, int count, str target);

The subchrs function copies a number of characters from one string into another and returns the target string. Only as many characters as will fit are copied. This function is very similar to substr, except that it is not string oriented, and does not stop copying characters when a 0 value is encountered.

Argument	Description
str source	The string to copy
	from.
int pos	The position in the
	string to begin copying
	from.
int count	The number of
	characters to copy
	from the source string.
str target	The string to copy into.

Return Value

The resulting target string is returned.

See also

SubStr, SubChr, CopyStr, CopyChrs

SubStr Function

Example

SubStr(str source, int pos, int max, str target);

The substr function copies a portion of one string to another and returns the resulting string. Characters are copied until a 0 (NULL) value is encountered (normally at the end of every string) or the maximum number is reached. Only as many characters as will fit are copied.

Argument	Description
str source	The string to copy
	from.
int pos	The position in the
	string to begin copying
	from.
int max	The maximum number
	of characters to copy
str target	The string to copy into.

Return Value

The resulting string is returned.

See also

SubChrs, CopyStr, CopyChrs

SubStr Example

```
str s[] = "horse cat dog", s2[16];
substr(s, 6, 3, s2);
if (s2 == "cat")
  prints("This will always be printed");
```

tDay Function

tDay(int timeval);

The tday function returns an integer value representing the day portion of a date.

Description Argument int timeval The date to retrieve the day from.

Return Value

The day of the month, ranging from 1 to 31.

See also

CurTime, FileTime, tHour, tMin, tMonth, tSec, tYear

```
tDay Example
```

```
int t;

t = curtime();
printsc("This is day number ");
printn(tday(t));
printsc(" of month number ");
printn(tmonth(t));
printsc(" in the year ");
printn(tyear(t));
prints(".");
```

tHour Function

Example

tHour(int timeval);

The thour function returns an integer value representing the hour portion of a date.

Argument Description
int timeval The date to retrieve the hour from.

Return Value

The hour of day, ranging from 0 to 23.

See also

<u>CurTime</u>, <u>FileTime</u>, <u>tDay</u>, <u>tMin</u>, <u>tMonth</u>, <u>tSec</u>, <u>tYear</u>

tHour Example

```
int t;

t = curtime();
printsc("The time is: ");
printn(thour(t));
printsc(":");
printn(tmin(t));
printsc(":");
printn(tsec(t));
prints("");
```

tMin Function

Example

tMin(int timeval);

The tmin function returns an integer value representing the minute portion of a date.

Argument Description
Int timeval The date to retrieve the minute from.

Return Value

The minute, ranging from 0 to 59.

See also

CurTime, FileTime, tDay, tHour, tMonth, tSec, tYear

```
tMin Example
```

```
int t;

t = curtime();
printsc("The time is: ");
printn(thour(t));
printsc(":");
printsc(":");
printsc(":");
printsc(":");
prints(tsec(t));
```

tMonth Function

Example

tMonth(int timeval);

The tmonth function returns an integer value representing the month portion of a date.

Argument Description
Int timeval The date to retrieve the month from.

Return Value

The month of the year, ranging from 1 to 12.

See also

CurTime, FileTime, tDay, tHour, tMin, tSec, tYear

tMonth Example

```
int t;

t = curtime();
printsc("This is day number ");
printn(tday(t));
printsc(" of month number ");
printn(tmonth(t));
printsc(" in the year ");
printn(tyear(t));
prints(".");
```

tSec Function

Example

tSec(int timeval);

The tsec function returns an integer value representing the second portion of a date.

Argument Description
The date to retrieve the second from.

Return Value

The second, ranging from 0 to 59.

See also

<u>CurTime</u>, <u>FileTime</u>, <u>tDay</u>, <u>tHour</u>, <u>tMin</u>, <u>tMonth</u>, <u>tYear</u>

```
tSec Example
```

```
int t;

t = curtime();
printsc("The time is: ");
printn(thour(t));
printsc(":");
printn(tmin(t));
printsc(":");
printn(tsec(t));
prints("");
```

tYear Function

Example

tYear(int timeval);

The tyear function returns an integer value representing the year portion of a date.

Argument Description
Int timeval The date to retrieve the year from.

Return Value

The year, ranging from 1970 to 2019.

See also

CurTime, FileTime, tDay, tHour, tMin, tMonth, tSec

tYear Example

```
int t;

t = curtime();
printsc("This is day number ");
printn(tday(t));
printsc(" of month number ");
printn(tmonth(t));
printsc(" in the year ");
printn(tyear(t));
prints(".");
```

TelixForWindows Function

TelixForWindows();

The telixforwindows function is used to determine if the version of the Telix script compiler is a Windows version. This function is useful if the <u>scriptversion</u> function returns 1 (TFW or compatible TFD), and you need to determine explicitly if the environment is Windows.

Return Value

If a DOS version, a zero (FALSE) value is returned. If a Windows version, a non-zero (TRUE) value is returned.

See also

ScriptVersion, TelixVersion

TelixVersion Function

TelixVersion();

The telixversion function is used to determine the version of Telix that is running.

Return Value

An integer value representing the version is returned (i.e. 100 for version 1.00).

See also

ScriptVersion, TelixForWindows

Terminal Function

Example

Terminal (int SaveKeys, int TrackHits);

The terminal function allows Telix to process characters received from the connect device and print them on the terminal screen, and process user keystrokes. If a function has nothing to do (for example, while waiting for a <u>track</u> hit), it can call terminal to make sure characters and user keystrokes are processed. Note that if a script wants to process every incoming character (e.g., with the <u>cgetc</u> function) the terminal function should never be called.

Argument	Description
int SaveKeys	If zero (FALSE), Telix
	will perform the default
	keyboard processing.
	If non-zero (TRUE),
	keystrokes will be
	stored until the script
	processes them.
int pos	If zero (FALSE), comm
	port processing will
	stop if a <u>track</u> hit
	occurs. If non-zero
	(TRUE), track hits will
	be ignored.

Return Value

A zero is always returned.



Terminal Example

```
// This will wait forever for either of two strings
// to come in from the comm port, and then stop.
int t1, t2, stat;
t1 = track("hello", 0);
t2 = track("good-bye", 0);
                   // loop forever
while (1)
  terminal();
                    // The call to terminal() lets any characters
                    // that come in be looked at by Telix's
                    // internal routines for a match with.
                    // Incoming chars are also printed on the
                    // terminal screen and user keystrokes are
                    // handled
  stat = track_hit(0);
  if (stat == \overline{t1} || stat == t2)
                                   // exit if one of the strings
                                   // came in
  break;
 }
                    // stop Telix for looking for more matches
track free(t1);
track_free(t2);
```

Time Function

<u>Example</u>

Time(int timeval, str buffer);

The time function converts a Telix date value into a string formatted according to the Windows settings. Time values in this format are returned by the <u>curtime</u> and <u>filetime</u> functions, among others.

Argument	Description
int timeval	The date value to
	convert.
str buffer	The string to copy the
	formatted date into.

Return Value

A zero is always returned.

See also

Date, CurTime, FileTime

```
Time Example
```

```
str s[16];
printsc("The current time is ");
time(curtime(), s);
prints(s);
```

Time_Up Function

Example

Time_Up(int thandle);

The time_up function determines if a specified timer has expired. The period of time after which a timer will elapse is specified in the <u>timer_start</u> function.

Argument	Description
int thandle	The handle of the
	timer to be checked as
	returned by the
	timer start function.

Return Value

A non-zero (TRUE) value if a specified timer has elapsed, otherwise a 0 (FALSE) value is returned.

See also

Delay, Timer_Free, Timer_Restart, Timer_Start, Timer_Total

Time_Up Example

```
int t;

t = timer_start(100);  // delay for 10 seconds
while (!time_up(t));

// start a timer and loop for 10 seconds, printing the elapsed time
// in tenths of seconds
timer_restart(t, 100);
while (!time_up(t))
{
   printn(timer_total(t));
   prints("");
}
timer_free(t);
```

Timer_Free Function

Example

Timer_Free(int thandle);

The timer_free function frees a timer variable when it is no longer needed. As only 16 timers are available to scripts, it is important that they be released when they are no longer needed.

Argument	Description
int thandle	The handle of the
	timer to be freed as
	returned by the
	timer_start function.

Return Value

A zero is always returned.

See also

Delay, Time_Up, Timer_Restart, Timer_Start, Timer_Total

Timer_Free Example

```
int t;

t = timer_start(100);  // delay for 10 seconds
while (!time_up(t));

// start a timer and loop for 10 seconds, printing the elapsed time
// in tenths of seconds
timer_restart(t, 100);
while (!time_up(t))
{
   printn(timer_total(t));
   prints("");
}
timer_free(t);
```

Timer_Restart Function

Example

Timer_Start(int thandle, int time);

The timer_restart function performs the same function as $\underline{\text{timer_start}}$, except that it restarts an existing timer.

Argument	Description
int thandle	The handle of the
	timer to be restarted
	as returned by the
	timer_start function.
int time	The time in tenths of
	seconds before the
	timer expires.

Return Value

The timer handle is returned if successful, otherwise a 0 is returned.

See also

Delay, Time Up, Timer Free, Timer Start, Timer Total



Timer_Restart Example

```
int t;

t = timer_start(100);  // delay for 10 seconds
while (!time_up(t));

// start a timer and loop for 10 seconds, printing the elapsed time
// in tenths of seconds
timer_restart(t, 100);
while (!time_up(t))
{
   printn(timer_total(t));
   prints("");
}
timer_free(t);
```

Timer_Start Function

Example

Timer_Start(int time);

The timer_start function is used to start a timer. This timer can then be used to check if a certain period of time has elapsed from when the timer was started.

Argument	Description
int time	The time in tenths of
	seconds before the
	timer expires.

Return Value

An integer value called a timer handle, that is used to refer to this timer when other timer functions are called.

See also

Delay, Time_Up, Timer_Free, Timer_Restart, Timer_Total

Timer_Start Example

```
int t;

t = timer_start(100);  // delay for 10 seconds
while (!time_up(t));

// start a timer and loop for 10 seconds, printing the elapsed time
// in tenths of seconds
timer_restart(t, 100);
while (!time_up(t))
{
   printn(timer_total(t));
   prints("");
}
timer_free(t);
```

Timer_Total Function Example

Timer Total(int thandle);

The timer_total function determines the total elapsed time since a timer was started or restarted.

Description Argument int thandle The handle of the timer to be freed as returned by the <u>timer_start</u> function.

Return Value

The elapsed time in tenths of a second.

See also

Delay, Time Up, Timer Free, Timer Restart, Timer Start

Timer_Total Example

```
int t;

t = timer_start(100);  // delay for 10 seconds
while (!time_up(t));

// start a timer and loop for 10 seconds, printing the elapsed time
// in tenths of seconds
timer_restart(t, 100);
while (!time_up(t))
{
   printn(timer_total(t));
   prints("");
}
timer_free(t);
```

ToLower Function

ToLower(int chr);

The tolower function converts a character to its lower case equivalent.

Argument	Description
int chr	The upper case
	character to be
	converted.

Return Value

The integer value of the converted character.

See also

ToUpper, StrLower, StrUpper

Tone Function

Example

Tone(int frequency, int length);

The tone function causes the PC to emit a sound of a specified frequency for a period of time.

Argument	Description
int frequency	The frequency of the
	tone to generate.
int length	The length of time in
	tenths of seconds to
	generate the tone.

Return Value

A zero is always returned.

See also

Alarm, PlayWave, alarm on, sound on



tone(659, 14);

ToUpper Function

ToUpper(int chr);

The toupper function converts a character to its upper case equivalent.

Argument	Description
int chr	The lower case
	character to be
	converted.

Return Value

The integer value of the converted character.

See also

ToLower, StrLower, StrUpper

Track Function

Example

Track(str trackstr, int mode);

The track function tells Telix to keep track of (watch for) a string to be received from the connect device. Up to 32 tracks may be active at one time.

Argument	Description
str trackstr	The string to be
	tracked.
int mode	If 0, case is significant,
	otherwise, it is not.
	Tracking strings where
	case is significant is
	faster, and should be
	used when many
	tracks are needed.

Return Value

An integer value called a track handle, which is used to refer to the string in other tracking functions.

See also

WaitFor, Track_AddChr, Track_Free, Track_Hit



Track Example

```
// Log-on to a BBS, answering two prompts in any order.
// This will loop forever, so for actual use would have
// to be changed a bit. See sample scripts for examples.
int stat, t1, t2;
t1 = track("Name? ", 0);
t2 = track("Password? ", 0);
                  // loop as long as needed
while (1)
 {
                  // call terminal function to allow Telix
  terminal();
                  // to look at incoming characters for
                  // matches and let Telix process user
                  // keystrokes
  stat = track_hit(0);
                             // see if any matches
  if (stat == t1)
                             // name prompt
    cputs("Joe Smith^M");
                             // send name and continue looping
  if (stat == t2)
                             // password prompt
    cputs("mypass^M");
                             // send password
                             // free track handles
track free(t1);
track free(t2);
```

Track_AddChr Function

Example

Track AddChr(int chr);

While a script is executing, Telix is not in terminal mode and does not have access to incoming characters to scan for matching strings. Therefore, the <u>terminal</u> function must periodically be called to allow Telix to process incoming characters. Alternately, if a script must process all incoming characters itself (with a function like <u>cgetc</u>), and therefore can not call the terminal function, they must still be added to the track routines for string matching to work. The track_addchr function performs this process. When it is called, Telix processes the specified character as if it had been received from the terminal handler, and uses it to scan for matching strings.

Note that all of the Print functions will call track_addchr automatically.

Argument	Description
int chr	The character to be
	processed by Telix's
	track handler.

Return Value

A zero is always returned.

See also

WaitFor, Track, Track Free, Track Hit



Track_AddChr Example

```
// Log-on to a BBS, answering two prompts in any order.
int stat, t1, t2, track1 hit = 0, track2 hit = 0;
t1 = track("Name? ", 0);
t2 = track("Password? ", 0);
               // loop as long as needed
while (1)
 {
 c = cgetc();
                             // use this instead of terminal() to
 track addchr(c);
                             // prevent terminal update
  stat = track hit(0);
                             // see if any matches
  if (stat == \overline{t1})
                             // name prompt
   cputs("Joe Smith^M");
                             // send name and continue looping
   if (track2_hit)
                             // exit loop if password was sent
     break;
   track1 hit = 1;
                             // set flag that name was sent
   }
  else if (stat == t2)
                             // password prompt
   {
                             // send password
   cputs("mypass^M");
   if (track1 hit)
                             // exit loop if name was sent
     break;
   track2 hit = 1;
                             // set flag that password was sent
   }
track_free(t1);
                             // free track handles
track_free(t2);
```

Track_Free Function

Example

Track_Free(int handle);

The track_free function is used to tell Telix to stop tracking a certain string. It is very important to free strings when they no longer need to be tracked, as tracking a large number of strings can slow down Telix's execution.

Argument	Description
int handle	The track handle that
	was returned when the
	string was added with
	the track function. If
	zero, all tracked
	strings are released.

Return Value

A zero is always returned.

See also

WaitFor, Track, Track_AddChr, Track_Hit



Track_Free Example

```
// Log-on to a BBS, answering two prompts in any order.
// This will loop forever, so for actual use would have
// to be changed a bit. See sample scripts for examples.
int stat, t1, t2;
t1 = track("Name? ", 0);
t2 = track("Password? ", 0);
                  // loop as long as needed
while (1)
 {
                  // call terminal function to allow Telix
  terminal();
                  // to look at incoming characters for
                  // matches and let Telix process user
                  // keystrokes
  stat = track_hit(0);
                             // see if any matches
  if (stat == t1)
                             // name prompt
    cputs("Joe Smith^M");
                             // send name and continue looping
  if (stat == t2)
                             // password prompt
    cputs("mypass^M");
                             // send password
                             // free track handles
track free(t1);
track free(t2);
```

Track_Hit Function

Example

Track Hit(int handle);

When track is called, Telix doesn't loop endlessly waiting for the string to come in, but instead returns control back to the script. As characters are received, Telix checks to see if any of the strings to be tracked have been matched, and marks those that have. A script can at any time call the track_hit function to see if a string was received. The marker on a handle is cleared once track_hit has indicated that the string was received.

Argument int handle The track handle that was returned when the string was added with the track function. If zero, the lowest numbered handle of any string that has be matched is returned, or zero if none have.

Return Value

If the string was received, a non-zero (TRUE) value is returned. If it hasn't been received, a zero (FALSE) value is returned.

See also

WaitFor, Track, Track_AddChr, Track_Free



Track_Hit Example

```
// Log-on to a BBS, answering two prompts in any order.
// This will loop forever, so for actual use would have
// to be changed a bit. See sample scripts for examples.
int stat, t1, t2;
t1 = track("Name? ", 0);
t2 = track("Password? ", 0);
                  // loop as long as needed
while (1)
 {
                  // call terminal function to allow Telix
  terminal();
                  // to look at incoming characters for
                  // matches and let Telix process user
                  // keystrokes
  stat = track_hit(0);
                             // see if any matches
  if (stat == t1)
                             // name prompt
    cputs("Joe Smith^M");
                             // send name and continue looping
  if (stat == t2)
                             // password prompt
    cputs("mypass^M");
                             // send password
                             // free track handles
track free(t1);
track free(t2);
```

TransTab Function

TransTab(str filename, int table);

The transtab function is used to load or clear the incoming or outgoing character translation table.

Argument	Description
str filename	If a filename
	specifying a valid
	translation table, it is
	loaded into memory.
	If empty (""), Telix will
	prompt for the table to
	load. If the string is
	"*CLEAR*", the current
	table is cleared.
int table	A value of 0 indicates
	the incoming
	translation table. A
	value of 1 indicates
	the outgoing
	translation table.

Return Value

A value of -1 is returned if there is a problem loading the indicated translate table, otherwise a non-zero (TRUE) value is returned.

Unload_Scr Function

Example

Unload Scr(str filename);

The <u>load_scr</u> function can be used by a script file to load another script into memory before it is actually used. The unload_scr function should then be used to unload or remove this script from memory when it is no longer needed. Note that a script that is currently executing or that is nested (has called the current script) must not be unloaded, since Telix is still executing it or will need to return to it eventually!

Argument	Description
str filename	The filename of the
	script to be unloaded.
	If no extension is
	given, .SLC is
	assumed.

Return Value

If there is a problem unloading the script file, a value of -1 is returned.

Otherwise a non-zero (TRUE) value is returned.

See also

Load_Scr, Is_Loaded

Unload_Scr Example

Update_Term Function Example

Update_Term();

The update_term function is called to ensure that Telix updates certain things relating to the video and terminal page. For example, changes made to the <u>back color</u> and <u>fore color system variables</u> will not take effect until this function is called.

> **Return Value** A zero is always returned.

Update_Term Example

UsageLog Function

Example

UsageLog(str filename);

The usagelog function is used to manipulate the Telix usage log facility.

Argument

Description

str filename

If a valid filename, all usage will be logged to that file. If empty (""), Telix will prompt for the filename to log usage to. If the string is "*CLOSE*", the usage log is closed, and no further logging will occur.

Return Value

A value of -1 is returned if there is a problem performing the indicated operation, otherwise a positive value is returned.

See also

UStamp, Usage Stat, usage fname



UsageLog Example

```
str oldlog[80];
int loginuse;

loginuse = usage_stat();
if (loginuse)
   {
    ustamp("Logging switched to SPECIAL.LOG.", 1, 1);
    oldlog = _usage_fname;
    usagelog("*CLOSE*");
   }

usagelog("SPECIAL.USE"); // Log all usage to the file SPECIAL.USE in the current directory.

// do actions that need logging.

usagelog("*CLOSE*"); // Close the special log.
if (loginuse)
   usagelog(oldlog);
```

Usage_Stat Function

Example

Usage_Stat();

The usage_stat function returns an integer value representing the current status of the Usage Log.

Return Value

If the Usage Log is currently open, a non-zero (TRUE) value is returned, otherwise a value of zero (FALSE) is returned.

Return Value

A zero is always returned.

See also

UsageLog, Capture Stat

Usage_Stat Example

```
str oldlog[80];
int loginuse;

loginuse = usage_stat();
if (loginuse)
{
   ustamp("Logging switched to SPECIAL.LOG.", 1, 1);
   oldlog = _usage_fname;
   usagelog("*CLOSE*");
}

usagelog("SPECIAL.USE"); // Log all usage to the file SPECIAL.USE in the current directory.

// do actions that need logging.

usagelog("*CLOSE*"); // Close the special log.
if (loginuse)
   usagelog(oldlog);
```

UStamp Function Example

UStamp(str text, int new_entry, int add_nl);

The ustamp function is used to place (stamp) text into the Telix usage log. If the usage log is not currently open, this function call is simply ignored.

Description				
The entry to place in				
the usage log.				
If non-zero (TRUE),				
the current date and				
time will preface the				
entry.				
If non-zero (TRUE), a				
carriage return and				
line feed are added				
after the entry. Using				
a value of zero				
(FALSE), will allow				
constructing single line entries with multiple calls to ustamp.				

Return Value

A value of -1 is returned if there is a problem writing to the usage log, otherwise a non-zero (TRUE) value is returned.

See also

<u>UsageLog</u>



UStamp Example

```
str oldlog[80];
int loginuse;

loginuse = usage_stat();
if (loginuse)
{
   ustamp("Logging switched to SPECIAL.LOG.", 1, 1);
   oldlog = _usage_fname;
   usagelog("*CLOSE*");
}

usagelog("SPECIAL.USE"); // Log all usage to the file SPECIAL.USE in the current directory.

// do actions that need logging.

usagelog("*CLOSE*"); // Close the special log.
if (loginuse)
   usagelog(oldlog);
```

vGetChr Function

Example

vGetChr();

The vgetchr function is used to read a character, including color information, at the current cursor position on the terminal screen.

Return Value

The return value contains the character in the first (low) byte, and the color of the character in the high (second) byte. Each component may be extracted using the & and / operators (See the Example).

See also

vGetChrs, vPutChrs, vPutChrs, vPutChrsA

```
vGetChr Example
```

vGetChrs Function

Example

vGetChrs(int x, int y, str buf, int pos, int num);

The vgetchrs function is used to read multiple characters from a position on the terminal screen into a string, saving only the characters and disregarding colors.

Argument	Description
int x	The column on the
	screen to begin
	reading characters.
int y	The line on the screen
	to begin reading
	characters.
str buf	The variable to store
	the characters in.
	Note that a 0 (NULL)
	character is not
	placed at the end of
	the string.
int pos	The position in the
	variable to begin
	storing characters.
int num	The number of
	character to read from
	the screen.

Return Value

A -1 is returned if the function fails, otherwise a zero is returned.

See also

vGetChr, vGetChrsA, vPutChr, vPutChrs, vPutChrsA

vGetChrs Example

```
// copy 20 characters starting from 10,10 on the screen to 20,20
// Don't keep color attributes
str buffer[20];

vgetchrs(10, 10, buffer, 0, 20);
vputchrs(20, 20, buffer, 0, 20);
```

vGetChrsA Function

Example

vGetChrsA(int x, int y, str buf, int pos, int num);

The vgetchrsa function is used to read multiple characters and their color attributes from a position on the terminal screen into a string. Note that to save both characters and colors, two bytes (characters) are required for each. Note also that this function does not put a 0 (NULL, or end of string character) at the end of the sequence of characters retrieved.

Argument	Description
int x	The column on the
	screen to begin
	reading characters.
int y	The line on the screen
	to begin reading
	characters.
str buf	The variable to store
	the characters in.
	Note that a 0 (NULL)
	character is not
	placed at the end of
	the string.
int pos	The position in the
	variable to begin
	storing characters.
int num	The number of
	characters to read
	from the screen.

Return Value

A -1 is returned if the function fails, otherwise a zero is returned.

See also

vGetChr, vGetChrs, vPutChr, vPutChrs, vPutChrsA

vGetChrsA Example

vPutChr Function

Example

vPutChr(int chr);

The vputchr function is used to place a character on the terminal screen at the current cursor position, including <u>color</u> information.

Argument	Description
int chr	The character and it's
	color attribute to place
	on the screen. The
	low (first) byte
	contains the character,
	and the high (second)
	byte contains the color
	attribute.

Return Value

A -1 is returned if the function fails, otherwise a zero is returned.

See also

vGetChrs, vGetChrsA, vPutChrs, vPutChrsA

vPutChr Example

vPutChrs Function

Example

vPutChrs(int x, int y, str buf, int pos, int num, int attr);

The vputchrs function is used to write multiple characters from a string onto the terminal screen at a given position with a specified <u>color attribute</u>.

Argument	Description					
int x	The column on the					
	screen to begin writing					
	characters.					
int y	The line on the screen					
	to begin writing					
	characters.					
str buf	The variable to read					
	the characters from.					
	Note that vputchrs					
	assumes that color					
	values are not					
	included.					
int pos	The position in the					
	variable to begin					
	reading characters.					
int num	The number of					
	characters to read					
	from the string.					
int attr	The color attribute to					
	use when writing the					
	characters to the					
	screen.					

Return Value

A -1 is returned if the function fails, otherwise a zero is returned.

See also

vGetChr, vGetChrs, vGetChrsA, vPutChr, vPutChrsA

vPutChrs Example

```
// copy 20 characters starting from 10,10 on the screen to 20,20
// Don't keep color attributes
str buffer[20];

vgetchrs(10, 10, buffer, 0, 20);
vputchrs(20, 20, buffer, 0, 20);
```

vPutChrsA Function

Example

vPutChrsA(int x, int y, str buf, int pos, int num);

The vputchrsa function is used to write multiple characters from a string onto the terminal screen at a given position.

Argument	Description
int x	The column on the
	screen to begin writing
	characters.
int y	The line on the screen
	to begin writing
	characters.
str buf	The variable to read
	the characters from.
	Note that vputchrsa
	assumes that <u>color</u>
	attributes are included.
int pos	The position in the
	variable to begin
	reading characters.
int num	The number of
	characters to read
	from the string.

Return Value

A zero is always returned.

See also

vGetChr, vGetChrs, vGetChrsA, vPutChr, vPutChrs

vPutChrsA Example

vRstrArea Function

Example

vRstrArea(int vhandle);

The vrstrarea function is used to restore a previously saved portion of the terminal screen.

Argument
int vhandle

The handle to the saved screen information returned by a call to the vsavearea function.

Return Value

A zero is always returned.

See also

<u>vSaveArea</u>

```
vRstrArea Example
```

vSaveArea Function

Example

vSaveArea(int x1, int y1, int x2, int y2);

The vsavearea function is used to save a rectangular portion of the terminal screen to be later restored. Characters, and their <u>color attributes</u>, are saved in an internal buffer.

It is very important that for every call to this function, there is a subsequent call to <u>vrstrarea</u>. If this is not done, the internal buffer used to store the screen area will not be released and can result in a memory shortage.

Argument	Description
int x1	The left column of the
	area to be saved.
int y1	The top line of the
	area to be saved.
int x2	The right column of
	the area to be saved.
int y2	The bottom line of the
	area to be saved.

Return Value

An integer value, commonly called a handle is returned. This handle is used to refer to the saved area in subsequent calls to <u>vrstrarea</u> which will restore the screen area. If Telix encounters a problem trying to save the screen, a value of -1 is returned.

See also

<u>vRstrArea</u>

vSaveArea Example

WaitFor Function

Example

WaitFor(str waitstr1 [... , str waitstr8], int timeout);

The waitfor function is used to wait for given strings to come to be received from the connect device.

Argument	Description
str waitstrN	A string to wait for.
	Case is not
	significant, and the
	string must not be
	longer than 40
	characters.
str waitstrn	Up to 8 strings may be
	monitored in one call
	to waitfor.
int timeout	The maximum amount
	of time to wait for a
	string to be matched.

Return Value

The index of the matched string will be returned, or a zero (FALSE) value if no strings were matched in the given amount of time.

See also

Track, Track_AddChr, Track_Free, Track_Hit



WaitFor Example

```
int i;
i = (waitfor("name?", "password", 180));
if i
{
   if (i == 1)
      prints("The string 'name?' came in from the comm port.");
   else
      prints("The string 'password' came in from the comm port.");
}
else
{
   prints("Neither 'name?' nor 'password' came in from the");
   prints("comm port in 3 minutes!");
}
```

System Variables

Telix for Windows has quite a large number of predefined variables. They are called **System Variables** and are used to store many user preferences. There are both string and numeric system variables, and you can access them just as you would any other variable. To help distinguish them from normal variables, they all start with an underscore (_) character. See the <u>Quick List</u> for a complete alphabetical list of all system variables.

Variable Quick List

<u>add_lf</u>	<u>auto ans str</u>	<u>entry</u> bbstype	no_connect3
<u>alarm_on</u>	<u>_back_color</u>	<u>entry_comment</u>	<u>no connect4</u>
<u>answerback</u> str	<u>_capture_fname</u>	<u>entry enum</u>	<u>redial_stop</u>
<u>asc_rcrtrans</u>	<u>_cisb_auto</u>	<u>_entry_name</u>	<u>scr_chk_key</u>
_asc_remabort	<u>_connect_str</u>	<u>entry_num</u>	<u>script_dir</u>
<u>asc</u> rlftrans	<u>_dest_bs</u>	<u>entry_logonname</u>	<u>sound</u> on
_asc_scpacing	<u>_dial_pause</u>	<u>entry pass</u>	<u>strip_high</u>
<u>asc</u> scrtrans	<u>dial_time</u>	<u>fore_color</u>	<u>_swap_bs</u>
<u>asc secho</u>	<u>_dialpost</u>	<u>image_file</u>	<u>telix</u> dir
_asc_sexpand	<u>dialpref1</u>	<u>local_echo</u>	<u>up_dir</u>
<u>asc_slftrans</u>	<u>_dialpref2</u>	<u>mdm_hang_str</u>	<u>usage_fname</u>
_asc_slpacing	<u>_dialpref3</u>	<u>_mdm_init_str</u>	<u>zmod</u> auto
<u>asc_spacechr</u>	<u>_dialpref4</u>	<u>no_connect1</u>	<u>_zmod_rcrash</u>
_asc_striph	<u>_down_dir</u>	<u>no connect2</u>	<u>zmod</u> scrash

_add_lf Variable

If the _add_If system variable is set to non-zero (TRUE), a Line Feed character is automatically added after every Carriage Return character that comes in.

_alarm_on Variable

If the _alarm_on system variable is set to non-zero (TRUE), alarms are enabled in Telix. Note that if the _sound_off system variable is set to zero (FALSE), alarms will not be heard no matter what the state of this variable.

See also

alarm, sound on

_answerback_str Variable

The _answerback_str system variable holds the string which Telix will send when a Ctrl-E (ENQ) character is received while in terminal mode. If this string is empty, nothing is sent. Note that if Compuserve B transfers are enabled, the answerback string will not be sent, since CIS B uses the Ctrl-E as part of the transfer process. Maximum length is 19 characters.

_asc_rcrtrans - _asc_striph Variables

_asc_rcrtrans determines what Telix does with Carriage Return characters during ASCII receives.

- 0 do nothing
- 1 strip
- 2 add Line afterwards

_asc_remabort is the character which when received from the remote side during an ASCII transfer is a signal to abort the transfer.

asc rlftrans determines what Telix does with Line Feed characters during ASCII receives.

- 0 do nothing
- 1 strip
- 2 add Carriage Return before

_asc_scpacing is the time in milliseconds which Telix should wait before transmitting each character during ASCII sends.

_asc_scrtrans determines what Telix does with Carriage Return characters during ASCII sends.

- 0 do nothing
- 1 strip
- 2 add Line Feed afterwards.

If _asc_secho is set to non-zero (TRUE), Telix will echo each character during ASCII sends.

If _asc_sexpand is set to non-zero (TRUE), Telix will expand blank lines to a space character, during ASCII sends.

_asc_slftrans determines what Telix does with Line Feed characters during ASCII sends.

- 0 do nothing
- 1 strip
- 2 add Carriage Return before

_asc_slpacing is the time in tenths of seconds which Telix should wait before transmitting each line during ASCII sends.

_asc_spacechr is the character which Telix should wait for during ASCII sends, before transmitting each line (0 means no wait).

If _asc_striph is set to non-zero (TRUE), Telix will strip the high (most significant) bit of each character in an ASCII transfer.

_auto_ans_str Variable

The _auto_ans_str system variable holds the string that should be sent to the modem to make it automatically answer the phone when it rings. This string is used by the Host Mode script, among others. The string will possibly include translation characters as described in the Telix manual in the section by that name, and should be sent to the modem with the cputs_tr function. Maximum length is 49 characters.

See also

mdm_init_str

_back_color Variable

The _back_color system variable contains the background color which should be used for text in terminal mode. Allowable values are from 0 - 15. Note that changes to this variable may not be reflected until the screen is cleared.

See also

fore color

_capture_fname Variable

The _	_capture_	_fname system	variable	holds the	default	capture	file filename.	The maximur	n length
is 64 characte	ers.								

See also

capture, usage fname

_cisb_auto Variable

This variable is not yet supported in Telix for Windows, but will be added soon.

See also

_zmod_auto

_connect_str Variable

The _connect_str system variable holds the string which Telix should scan for when dialing, and should take to mean that a connection has been established. For Hayes type modems it is usually set to "CONNECT". Maximum length is 19 characters.

See also

no connect1 - no connect4

_dest_bs Variable

The _dest_bs system variable controls whether a backspace character received by Telix in
Terminal Mode erases the character to the left of the cursor, or just moves the cursor on top of it without
erasing it. If this variable is 0 (FALSE), Telix will treat the backspace as non-destructive, and destructive
otherwise.

See also

_swap_bs

_dial_pause Variable

The _dial_pause system variable holds (in seconds) the amount of time to wait between the end of one dialing attempt and the beginning of another. Most modems don't need more than a 1 second pause.

_dial_time Variable

The _dia	al_time system	variable holds the	e amount of time	Telix should	wait for a	connection	when
dialing, in secon	ds (e.g. 30).						

See also

<u>_dial_pause</u>

_dialpost Variable

The _dialpost system variable holds the string (the dialing suffix) which should be sent to the modem after the number, when dialing. For Hayes type modems, it is usually just a Carriage Return. Maximum length is 19 characters. This string will possibly include some translation characters, as described in the Telix manual, and should be sent to the modem with the cputs_tr function.

See also

<u>dialpref - dialpref3, redial_stop</u>

_dialpref Variables

The _dialpref system variable holds the string which should be sent to the modem before the number, when dialing. For Hayes type modems, it is usually set to "ATDT". Maximum length is 19 characters. This string will possibly include translation characters, as described in the Telix manual, and should be sent to the modem with the cputs_tr function.

The _dialpref2, _dialpref3 and _dialpref4 variables are the other two dialing prefixes that may be defined in Telix.

See also

_dialpost, _redial_stop

_down_dir Variable

The _down_dir system variable holds the default download directory name. When a file is downloaded (received), if the user specifies a drive and/or directory in the name, the file is put there. However, if only a name is specified, the file is placed in the directory indicated by _down_dir. The maximum length is 64 characters, and this string should end with the backslash character, '\'.

See also

up_dir, receive

_entry_enum Variable

The _entry_enum variable is set by the dialing routines. When a connection is established while dialing, the entry number of the dialing directory entry connected to is stored here. If a manual number is connected to, the value 0 is stored here.

See also

entry_name, dial, redial

_entry_bbstype - _entry_pass Variables

All of the _entry_xxxx variables are set by the dialing routines. When a connection has been established, the appropriate value is copied into the variable.

The _entry_bbstype system variable contains the name of the BBS software package that is connected to. This variable is assigned in the dialing directory entry by the user. Telix does not determine this upon connection to a system. The maximum length is 15 characters.

The _entry_comment system variable holds the value of the comment field from the entry connected to. Common uses are for storing mail packet filenames or secondary passwords. The maximum length is 40 characters.

The _entry_name system variable contains the name of the dialing directory entry connected to. The maximum length is 29 characters.

The _entry_num system variable holds the phone number of the entry connected to. The maximum length is 17 characters.

The _entry_logonname variable contains the user name for the entry connected to. This is useful for performing logons. The maximum is 25 characters.

The _entry_pass system variable holds the password from the entry connected to. This is useful for performing logons. The maximum length is 14 characters.

See also

<u>entry</u>enum, dial, redial

_fore_color Variable

The _fore_color system variable contains the foreground $\underline{\operatorname{color}}$ which should be used for text in terminal mode. Allowable values are from 0 - 15. Note that changes to this variable may not be reflected until the screen is cleared, or the update_term function is called.

See also

<u>back_color</u>, <u>Update_Term</u>

_image_file Variable

The _image_file system variable holds the full name of the file that screen images are saved to when the user selects Screen Image from the File menu or presses Alt-I while in terminal mode. If this file already exists, the screen is appended to it.

_local_echo Variable

The _local_echo system variable controls whether or not characters typed in terminal mode are echoed on the screen. If _local_echo is set to non-zero (TRUE), characters are echoed, otherwise they are not.

_mdm_hang_str Variable

The _mdm_hang_str system variable holds the string that should be sent to the modem to hang it up when the user presses Alt-H. Note that this string will only be sent to the modem if Telix can't first hang-up the modem by turning off a signal on the serial port called the DTR line. This string may contain translation characters as defined in the Telix manual, and should be sent to the modem with the cputs_tr function. Maximum length is 19 characters.

See also

mdm init str, auto ans str

_mdm_init_str Variable

The _mdm_init system variable holds the string that should be sent to the modem when Telix is started. It is used to make sure certain settings in the modem are right. This string may contain translation characters as defined in the Telix manual, and should be sent to the modem with the cputs_tr function. Maximum length is 49 characters.

See also

auto ans str, mdm hang str

_no_connect1 - _no_connect4 Variables

These system variables contain the strings that Telix should scan for when dialing that indicate a connection could not be established (i.e., the number was busy or there was no answer). The maximum length for each string is 19 characters.

See also

connect_str

_redial_stop Variable

The _redial_stop system variable holds the string that should be sent to the modem to stop a dialing attempt. It usually just holds a Carriage Return character. This string may contain translation characters as described in the Telix manual, and should be sent to the modem with the cputs_tr function. Maximum length is 19 characters.

See also

<u>dialpref</u>, <u>dialpost</u>

_scr_chk_key Variable

If the _scr_chk_key variable is set to zero (FALSE), Telix will not allow executing scripts to be aborted. If set to non-zero (TRUE), users will be allowed to abort scripts.

_script_dir Variable

The _script_dir system variable holds the full path of the directory where Telix should look for compiled script files when a script is selected to be run. When a script is selected to be run, Telix uses this procedure: if the name includes the drive and/or directory, only that path is searched. If the name includes only the filename, the current directory is first searched for the script file, and then the directory pointed to by the _script_dir variable. This string should end in the backslash (\) character. The maximum allowed length is 64 characters.

See also

<u>telix dir</u>, <u>up dir</u>, <u>down dir</u>

_sound_on Variable

If the _	_sound_	on system	variable	is set to	non-zero	(TRUE)	sound	is enable	d in [*]	Telix,	otherwise
all sound is sh	ut off.										

See also

alarm, _alarm_on

_strip_high Variable

The _strip_high system variable controls what Telix does with the high (most significant) bit of incoming characters while in terminal mode. If this variable is set to a non-zero (TRUE) value, Telix will strip the high bit of incoming characters.

_swap_bs Variable

The _swap_bs system variable controls what Telix sends when the Backspace key is pressed. If this variable is 0, Telix will send a Backspace character when Backspace is pressed, and a DEL character when Ctrl-Backspace is pressed. If this variable is set to 1, Telix will reverse these codes.

See also

_dest_bs

_telix_dir Variable

The _telix_dir system variable holds the full path to reach the Telix program's base directory (e.g. 'C:\TELIX\'). Changing this variable is not recommended, as if a wrong value is used, Telix will probably not be able to find many needed files. The maximum length is 64 characters.

If this variable is changed, it is imperative that a backslash (\) character is found at the end. Telix builds paths to many files by appending certain names to this string. If the backslash is missing, it will cause many problems.

See also

script dir, up dir, down dir

_up_dir Variable

The _up_dir system variable holds the default upload directory name. When a file is to be uploaded (sent), if the user specifies a drive and/or directory in the name, the file is taken from there. However, if only a name is specified, the file is searched for in the directory indicated by _up_dir. This variable should end with a backslash (\ \) character. The maximum length is 64 characters.

See also

_down_dir, send

_usage_fname Variable

The _usage_fname system variable holds the default	Usage Log filename. The maximum length
is 64 characters.	

See also

_capture_fname, usagelog

_zmod_auto Variable

The _zmod_auto system variable controls whether or not Zmodem autodownloads are allowed. If Telix is in terminal mode and receives an auto download request Telix will ignore it if this variable is set to a 0 (FALSE) value. The download will then have to be initiated by the user.

See also

<u>cisb_auto</u>

_zmod_rcrash Variable

The _zmod_rcrash system variable controls whether the Zmodem receive Crash Recovery (resume) option is on. If this variable is set to a non-zero (TRUE) value, Telix will try to resume aborted transfers during a Zmodem download.

See also

150

_zmod_scrash

_zmod_scrash Variable

The _zmod_scrash system variable controls whether the Zmodem send Crash Recovery (resume) option is on. If this variable is set to a non-zero (TRUE) value, Telix will try to tell the other side to resume aborted transfers during a Zmodem upload.

See also

zmod_rcrash

ASCII Character Set

The ASCII character set consists if 128 characters, with each character having an ASCII value, in the range of 0 to 127. The IBM PC uses the IBM Extended ASCII set, which adds a further 128 values, to provide extra symbols. The following table lists the regular ASCII character set. The first column contains the ASCII control characters, which can not normally be printed, and are given by name.

Dec	Hex	Ctrl	Name	Dec	Нех	Chr	Dec H	lex (hr	Dec	Hex	Chr
0	00	^@	NUL	32	20		64	40	@	96	60	`
1	01	^A	SOH	33	21	!	65	41	Α	97	61	a
2	02	^B	STX	34	22	"	66	42	В	98	62	b
3	03	^C	ETX	35	23	#	67	43	С	99	63	С
4	04	^D	EOT	36	24	\$	68	44	D	100	64	d
5	05	^E	ENQ	37	25	용	69	45	Ε	101	65	е
6	06	^F	ACK	38	26	&	70	46	F	102	66	f
7	07	^G	BEL	39	27	•	71	47	G	103	67	g
8	08	^H	BS	40	28	(72	48	Н	104	68	h
9	09	^Ι	HT	41	29)	73	49	I	105	69	i
10	0a	^J	LF	42	2a	*	74	4a	J	106	6a	j
11	0b	^K	VT	43	2b	+	75	4b	K	107	6b	k
12	0 c	^L	FF	44	2c	,	76	4c	L	108	6с	1
13	0d	^M	CR	45	2d	-	77	4d	Μ	109	6d	m
14	0e	^N	SO	46	2e		78	4e	N	110	6e	n
15	0f	^0	SI	47	2f	/	79	4f	0	111	6f	0
16	10	^P	DLE	48	30	0	80	50	P	112	70	р
17	11	^Q	DC1	49	31	1	81	51	Q	113	71	q
18	12	^R	DC2	50	32	2	82	52	R	114	72	r
19	13	^S	DC3	51	33	3	83	53	S	115	73	s
20	14	^T	DC4	52	34	4	84	54	Τ	116	74	t
21	15	^U	NAK	53	35	5	85	55	U	117	75	u
22	16	^V	SYN	54	36	6	86	56	V	118	76	V
23	17	^W	ETB	55	37	7	87	57	W	119	77	W
24	18	^X	CAN	56	38	8	88	58	Χ	120	78	Х
25	19	^Y	EM	57	39	9	89	59	Y	121	79	У
26	1a	^ Z	SUB	58	3a	:	90	5a	Z	122	7a	Z
27	1b	^ [ESC	59	3b	;	91	5b	[123	7b	{
28	1c	^\	FS	60	3с	<	92	5c	\	124	7с	
29	1d	^]	GS	61	3d	=	93	5d]	125	7d	}
30	1e	^^	RS	62	3е	>	94	5e	^	126	7e	~
31	1 f	^_	US	63	3f	?	95	5f	_	127	7f	DEL

Extended Key Scan Codes

The following chart lists keyboard scan codes for special non-ASCII keys, as returned by inkey and inkeyw, and used by the keyget, keyset, keyload, and keysave SALT functions. Normal keys which are within the ASCII set are listed in the preceding topic, <u>ASCII Character Set</u>.

Key	Norm Dec	nal Hex	w / C Dec	trl Hex	w / Dec		w / S Dec	Shift Hex
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11	15104 15360 15616 15872 16128 16384 16640 16896 17152 17408	3c00 3d00 3e00 3f00 4000 4100 4200 4300	24064 24320 24576 24832 25088 25344 25600 25856 26112 26368	5f00 6000 6100 6200 6300 6400 6500 6600	26624 26880 27136 27392 27648 27904 28160 28416 28672 28928		21760 22016 22272 22528 22784 23040 23296	5400 5500 5600 5700 5800 5900 5a00 5b00 5c00 5d00
1 2 3 4 5 6 7 8 9					30720 30976 31232 31488 31744 32000 32256 32512 32768 33024	7800 7900 7a00 7b00 7c00 7d00 7e00 7f00 8000 8100		
Up Down Left Right Home End PgUp PgDn Ins Del	18432 20480 19200 19712 18176 20224 18688 20736 20992 21248	4800 5000 4b00 4d00 4700 4f00 4900 5100 5200 5300	29440 29696 30464 29952 33792 30208	7300 7400 7700 7500 8400 7600				

Color Values

Several SALT functions, such as <u>pstra</u>, use color attribute values. A character on the text screen has a foreground color, and a background color. Possible colors are numbered as follows:

Color	Integer Value
Black	00
Blue	01
Green	02
Cyan	03
Red	04
Magenta	05
Brown	06
Light Gray	07
Dark Gray	08
Light Blue	09
Light Green	10
Light Cyan	11
Light Red	12
Light	13
Magenta	
Yellow	14
White	15

To obtain a color attribute value for a color combination, the formula is:

color attribute value = foreground color value + (16 * background color value)

Therefore, a Yellow character on a Blue background would have a color attribute value of 30 (14 + (16 * 1)).

Note that only the first eight colors are valid background colors. Using light colors for a background yields it's corresponding dark color, and causes the foreground color to blink.

The following is a useful function to add to your toolbox to make calculating color attributes easier.

```
#CONST Black
                 00
#CONST Blue
                 01
#CONST Green
                 02
#CONST Cyan
                 03
#CONST Red
                 04
                 05
#CONST Magenta
                 06
#CONST Brown
#CONST Lt Gray
                 07
#CONST Dk Gray
                 80
#CONST Lt Blue
                 09
#CONST Lt Green
                 10
#CONST Lt_Cyan
                 11
#CONST Lt_Red
                 12
#CONST Lt Magenta 13
#CONST Yellow 14
                 15
#CONST White
MakeColor(int Fore, int Back)
{
```

```
int color;
color = Fore + (Back * 16);
return color;
}
```

Using the constants in conjunction with the MakeColor function, the color attributes are calculated for you, and the resulting code is more easily understood. Consider the following:

```
pstra("Print this string in yellow on blue", 30);
pstra("Print this string in yellow on blue", MakeColor(Yellow, Blue));
```

SIMPLE Language

The Telix SIMPLE Language

Telix for Windows has a built-in programming language called SALT (Script Application Language for Telix). SALT is extremely powerful, and much of that power is due to its semblance to the C programming language. Along with that power comes a degree of difficulty, however. For those not comfortable in a structured programming environment such as SALT, a second scripting language, SIMPLE (Salt IMPLEmentation) is also provided.

Simple takes a loosely structured program resembling a stream of English sentences and transforms it into SALT for you. No programming experience is necessary. Its SIMPLE!

What Can Be Accomplished With SIMPLE?

SIMPLE scripts can be used to automate logins to bulletin boards, automate repetitive tasks such as mail transfers, or can be programmed to watch for multiple strings, offering up the proper response to each no matter the order in which they arrive. SIMPLE offers you much of the power of SALT without the learning curve.

Comparing SIMPLE to SALT

SALT's complexity allows it to do much more than SIMPLE can. For example, the Host+ bulletin board that comes with Telix was written primarily in SALT, but such a task would not be possible in SIMPLE. SALT offers access to most screen-related functions in Telix; SIMPLE offers only a pair of commands to place information on the screen. SALT offers full color control; SIMPLE does not. Other differences make SALT the preferable language for complex scripts.

Creating SIMPLE Scripts

A SIMPLE script is basically a sequence of instructions for Telix to follow, using a loosely defined syntax. You may use any text editor to produce this script file, as long as its output is normal ASCII text (this means that if you use your word processor, you must explicitly tell it to write out the file using ASCII format and not to embed any special codes in the file). You may give any name you wish to a SIMPLE script, although we recommend that you always use the extension .SIM for clarity. For example, a script to log on to the Telix Support BBS might be called TELIX.SIM. If you use the Script Editor that is built in to Telix for Windows, you **must** give the file an extension of .SIM or the compiler will treat the file as a SALT script.

Once you have written your script file and saved it to disk, it must be compiled. Using the SALT Editor, you need only to select the Compile command from the menu, or if using an external editor, select Compile from the Telix for Windows Script menu. The compiler then reads your 'source' script file, and compiles it to a form which Telix can understand. The compiled script can then be loaded more quickly by Telix, and is also usually smaller. The output file is written with the same name as the source file except that the extension .SLC is used.

When the script compiler finds an error in your source file, it will abort the compile process and give you the line number on which the error occurred, as well as the type of error. If you are using the SALT Editor, the cursor will be moved to the line that the error occurred on and the error message will be displayed on the status bar. The error should then be fixed and the source re-compiled. This is repeated until the compiler detects no more errors in your source file. The compiled script can then be executed in Telix using one or all of several methods; using the Execute menu item on the Script menu, as a linked script in a dialing directory entry, or called from another script.

See also

SIMPLE Syntax, SIMPLE Structure, SIMPLE Functions, SIMPLE Program Control

SIMPLE Syntax

Case is not important in command, function, and variable names. The only time case matters is inside a string constant (e.g., "Hello" is not the same string as "hello"). Whitespace (such as the space, the tab, the Carriage Return, or the Line Feed) is not important. The script compiler does not care where you place items, so that you may arrange the program as you see fit. For example,

```
If Online Then Send String25
Else Dial "11" MaxOf 23 RunScript
is equivalent to
If Online Then
  Send String25
Else
  Dial "11" MaxOf 23 RunScript
or even to
Ιf
Online
Then
Send
String25
Else
Dial
"11"
MaxOf
23
RunScript
```

The only time whitespace matters is when it would split up key-words or function name, or in a string. For example, the key-word 'whenever' must not be split up if it is to be recognized. The same applies to other key-words or function names. Also, there must be space between the letters of a command and other letters. For example, 'whenever' is not the same as 'wheneverabc'. In the interest of clarity, it is recommended that you try to make your script easy to understand, by indenting where appropriate, and by using space effectively. There is no reason, for example, to put more than one statement on a line, even if it is perfectly legal. Another poor example, as the last example above illustrates, is one where a complete line is broken up for no good reason. A good example of program style can be found by looking at the sample SIMPLE scripts included with Telix.

SIMPLE Structure

Program Structure

A SIMPLE script has no set format beyond a few easy-to-follow rules. These rules will be discussed as the apply to the individual commands when necessary. Otherwise, simply add commands to your script to do whatever is necessary.

Variables

A SIMPLE script may use up to 255 string "variables", or groups of characters, that you can change as you see fit. You need not do anything special to use a string. Just use the word "StringXX" wherever you need the string, where XX is the number of the string. All SIMPLE strings are exactly 80 characters in length. These will be referred to as StringXX variables throughout this documentation. Examples of StringXX use might be as follows:

To create string number 20, and make it contain the phone number of the Telix BBS, you might have a line:

```
Assign String20, "1-919-481-9399"
```

Note that you do not have to use all 80 characters of a string. SIMPLE knows where to end a string if you don't fill it up.

To create a string that contains today's date, you could simply use:

```
Date String15
```

If you need to use either the quote character itself in a string, or the carat symbol (the shifted-6), both of which have special meaning in Telix, you must dereference them. To dereference the character, precede it with a carat. Examples of this are:

```
Assign String1, "A quote, ^", needs a carat in front."
Assign String2, "A carat, ^^, is represented by two carats."
```

Integer variables are also available, referred to as IntegerXX variables, and conform to the same rules as StringXX variables described above. IntegerXX variables can contain values ranging from - 2,147,483,683 to 2,147,483,648.

System Variables

SIMPLE has four system variables which may be used as part of certain statements. Their use will be explained in greater detail as part of the commands that may access them. These variables are:

BBSNumber: This variable will contain the dialing directory entry number after dialing and connecting to

a system. It changes only when a connection is made.

TransferStatus: This variable is explained in detail under the If directive.

ReturnCode: This variable is explained in detail under the <u>lf</u> directive.

BBSPassword: This variable contains your password for the system you last connected to, as read from the dialing directory. This variable makes it possible to write a script that doesn't have to be recompiled every time you change your password. The script can just use this variable instead of a String variable. All you need to do when changing your password is to edit the dialing directory within Telix, and insert the new password.

SIMPLE Functions

<u>Alarm</u> **Emulate** <u>Assign</u> **End** Begin CaptureLog **EraseFile** ExitScript ChangeDir ClearScreen **ExitTelix** Hangup <u>Date</u> <u>Input</u> **Dial** <u>Message</u> Dos Printer Download RunScript Send Shell Show Sound Time Upload UsageLog Wait

ADD AddString Cursor DIV MUL RenameFile SUB

ADD Function

Example

ADD integer1, integer2

The ADD function adds the value of *integer2* to *integer1*. Neither integer need be defined prior to an ADD. The first value must be an IntegerXX variable, and the second value may be either a variable or literal value. The comma between the two values is required.

See also

Assign, DIV, MUL, SUB



Assign Integer30, 50 ADD Integer30, 25 ADD Integer30, Integer30

AddString Function Example

AddString string1, string2

The AddString function adds, or concatonates, the value of *string2* to *string1*. The *string1* variable must be a <u>StringXX</u> variable, and <u>string2</u> may be a literal string in quotes or a <u>StringXX</u> variable. The comma between the two strings is required.

See also

<u>Assign</u>

AddString Example

Assign String1, "TELIX"
AddString String1, ".REP"

Alarm Function

Example

Alarm integer

The Alarm function plays the Alarm wave sound defined in the Sounds configuration *integer* times.

Alarm Example

```
Send "C:\TFW\UP\MYMAIL.REP" with ZMODEM
if TransferStatus = 0 then
  Begin
   Alarm 1
  Message "Mail packet was not uploaded!"
end
```

Assign Function

<u>Example</u>

Assign string1, string2
Assign integer1, integer2

The Assign function assigns the value of *string2* to *string1*, or the value of *integer1* to *integer2*. Neither string, or integer, need be defined prior to an assign. The first value must be a <u>StringXX</u>, or <u>IntegerXX</u>, variable, and the second value may be either a variable or literal value. The comma between the two values is required.

See also

AddString, ADD, DIV, MUL, SUB



Assign String24, "Telix Support BBS" Assign String40, String24

Assign Integer30, 500 Assign Integer35, Integer30

$\underset{\underline{\text{Example}}}{\text{Begin Command}}$

Begin

The Begin command denotes the start of a group of code that belongs together. It is typically used to keep a group of code together in conjunction with If statements. Every Begin command must have a corresponding End command. Begin and End commands may be nested within each other. It is strongly recommended that indentation be used to help keep track of the Begin and End pairs.

See also

End

Begin Example

```
if Online then
  Begin
    Assign String4, "TELIX.REP"
    Alarm 3
  End
```

CaptureLog Function

Example

CaptureLog string | OFF | ON | PAUSE | UNPAUSE

The CaptureLog function controls the status of the Telix Capture file, in much the same was as Alt-L does from the keyboard in Telix. You may turn on the capture log by passing a <u>StringXX</u> or constant string, **PAUSE**, **UNPAUSE**, or turn **OFF** the capture file entirely. Only one of these actions may be performed per CaptureLog command. CaptureLog **ON** opens the capture log with the filename specified in the Telix Filenames and Paths configuration.

See also

UsageLog



CaptureLog "TEMP.CAP"
CaptureLog Pause
CaptureLog Unpause
CaptureLog Off

CaptureLog On CaptureLog Off

Assign String16, "TELIX.CAP" CaptureLog String16 CaptureLog Off

ChangeDir Function

ChangeDir string1

The ChangeDir function provides access to the DOS "CD" command. You may change to any valid directory with this command. Invalid directories are simply ignored, and Telix will remain in the current directory. String1 may be either a <u>StringXX</u> variable or a literal string in quotes.



Assign String64, "D:\TELIX\DOWN" ChangeDir String64 ChangeDir "C:\TELIX"

ClearScreen Function

Example

ClearScreen

The ClearScreen function acts as if you had pressed Alt-C from within Telix. It clears the terminal window of all characters. ClearScreen does not accept any parameters.



ClearScreen

Cursor Function

Cursor ON/OFF

The Cursor function turns the blinking cursor in the terminal window on or off.

Date Function Example

Date string

The Date function places the current date into *string*. *string* must be a <u>StringXX</u>-type variable.



Date String16

show "Today's date is "
show String16 ENTER

Dial Function

Example

Dial string1 [FROM string2] [MAXOF integer] [RUNSCRIPT]

The Dial function allows nearly complete access to the Telix dialing directory. Telix can be told to dial several entries, optionally from a specific dialing directory file, and can be told whether or not to run a linked script.

Dial must be passed at least one parameter, a <u>StringXX</u> variable or string constant in quotes containing numbers to dial. This string may contain either a list of entries by number, or a manual number preceded with a lowercase "m".

If you wish, you may tell Dial from which directory these numbers are to be read, with an optional **FROM** directive. From must be passed a <u>StringXX</u> or literal string constant in quotes, containing the name of the directory to load. If you use a **FROM** directive, the specified directory will be used throughout the rest of the script, or until a **FROM** directive is encountered in another dial function. To avoid confusion, it is best to either always use **FROM**, or never use it. If **FROM** is not used, the currently loaded directory shall be the source.

You may tell Telix to limit the number of dialing attempts to make by using an optional **MAXOF** directive. **MAXOF** must be followed by an integer, which specifies the number of attempts to make. **MAXOF** must come after **FROM**, if **FROM** is present.

You may tell Telix to execute the script linked to the dialing directory with the **RUNSCRIPT** directive. The default is not to execute such a script. By placing the optional directive **RUNSCRIPT** at the end of the Dial command, Telix will execute a linked script attached to the entry connected to, if there is one, and return control to your SIMPLE script upon completion.

The Dial function places a return value into the system variable ReturnCode as follows:

If there was a connection, ReturnCode shall be the entry number in the dialing directory of the system connected to, or 1 for a manual number.

If there was no connection established, a zero shall be placed into ReturnCode.

If the string passed to dial did not contain a string that could be interpreted as a valid list of numbers to dial, -1 will be placed into ReturnCode.

The use of the ReturnCode is discussed in detail as part of the <u>lf</u> statement.



```
Assign String24, "1 5 6"
Assign String64, "D:\TELIX\FON\LONGDIST.FON"

// The first example dials entries 1, 5, and 6 from LONGDIST.FON until it

// connects to one of them or the user presses escape.

Dial String24 From String64

// The second example dials the Telix Support BBS manually, up to 50 times.

Dial "m1-919-481-9399" MaxOf 50

// The third example dials entries 1, 5, and 6, after loading TELIX.FON. If a

// connection is made, any script linked to the entry will be executed.

Dial String24 From "TELIX.FON" RunScript
```

DIV Function

Example

DIV integer1, integer2

The DIV function divides the value of *integer1* by *integer2*. Neither integer need be defined prior to a DIV. The first value must be an IntegerXX variable, and the second value may be either a variable or literal value. The comma between the two values is required.

See also

Assign, ADD, MUL, SUB



Assign Integer30, 50 DIV Integer30, 2

Dos Function

Example

Dos string [PAUSE]

The Dos function allows you to shell to DOS to execute the program specified in *string*. *String* may be a String XX variable or a literal string in quotes. If you wish Telix to pause prior to returning, simply place the optional directive **PAUSE** after the command to execute.

The Errorlevel that DOS returns after running the command is placed in the system variable <u>ReturnCode</u>. Use of the ReturnCode is discussed with the <u>If</u> statement. Please see your DOS manual for more information regarding the DOS Errorlevel.



Assign String64, "C:\WP51\WP.EXE"
Dos String64
Dos "D:\UTIL\QEDIT.EXE" Pause

Download Function

Example

Download string WITH protocol

The Download function acts just as if you had pressed Alt-PgDn and entered a protocol and filename. It will download the file (or files if a batch protocol is used) indicated by *string*. *String* may be a <u>StringXX</u> variable or a string constant in quotes.

The Download command uses the protocol specified by the **WITH** directive. If no protocol is specified, Telix will prompt for the protocol. Protocols allowed are:

XModem 1K-XModem G-1K-XModem YModem YModem-G ZModem

Remember that downloads usually need to be triggered on the remote site before you can receive the file. You will usually need to <u>Send</u> a start command prior to using the Download command.

The success or failure of the transfer is reported in the special system variable TransferStatus that is described in the <u>If...Then...Else</u> section.



Assign String64, "D:\TELIX\DOWN\WORK\TELIX.QWK"

Send "D;Z" Enter
Send Enter
Download String64 with Zmodem
If TransferStatus = 0 Then
 Message "Download failed!"

Send "D;G" Enter
Send Enter
Download "C:\TELIX\DOWN\TLX320-1.ZIP" With Ymodem-G
If TransferStatus > 0 Then
 Message "Download succeeded."

Emulate Function

Example

Emulate terminal

The Emulate function tells Telix to change the terminal emulation it is using to that specified by *terminal*. Allowable emulations are:

TTY VT52 VT100 VT102 VT220 ANSI X3.64 ANSI-BBS AVATAR

RIPscrip 1.54



Emulate VT102 Emulate ANSI-BBS

End Command

Example

End

The End command denotes the end of a group of code that belongs together. It is typically used to keep a group of code together in conjunction with <u>If</u> statements. Every End command must have a preceding <u>Begin</u> command. Begin and End commands may be nested within each other. It is strongly recommended that indentation be used to help keep track of the Begin and End pairs.

End Example

```
if Online then
  Begin
    Send "U" Enter
    Upload "TELIX.REP" with Zmodem
  End
```

EraseFile Function

Example

EraseFile *string*

The EraseFile function deletes the file specified in *string* from the disk. Be careful using this command as deleted files are usually unrecoverable. *String* may be a <u>StringXX</u> variable or a literal string in quotes.

See also

RenameFile



Assign String24, "D:\WINDOWS\TELIX.TTF" EraseFile String24

EraseFile "C:\TELIX\QWIK\TELIX.QWK"

ExitScript Function Example

ExitScript

The ExitScript function halts execution of the script. Telix will not prompt for confirmation, but instead will stop the script immediately.

ExitScript Example

```
Assign String64, "D:\TELIX\DOWN\WORK\TELIX.QWK"

Send "D;Z" Enter

Send Enter

Download String64 with Zmodem

if TransferStatus=0 then

begin

Alarm 1

Message "Download failed! Exiting script."

ExitScript

end

// If download succeeded then upload replies.

Send "U;Z" ENTER

Send Enter

Upload "D:\TELIX\UP\TELIX.REP"
```

ExitTelix Function

Example

ExitTelix

The ExitTelix function halts execution of the script, and exits Telix completely. It is exactly like pressing Alt-X in terminal mode.



ExitTelix

Hangup Function

Hangup

The Hangup function disconnects you from any system you might be connected to at the time. It is exactly like pressing Alt-H from terminal mode.



if OnLine then HangUp

Dial "1"

Input Function Example

Input string, integer

The Input function gets up to integer characters from the keyboard and places them into string. Integer is any integer value from 1 to 80, and string must be a StringXX-type variable.

No prompting is made by SIMPLE. If you wish to prompt the user for the data, you will want to display the prompt yourself using the **Show** command.



```
// An example of the Input routine below allows the user to input up to 40 \, // characters, placing them into String22.
```

Input String22, 40

Message Function Example

Message string

The Message command places string into a centered box on the screen for exactly three seconds. It is very much like the prompts that Telix uses to request confirmations from you, but it allows you to specify the message in the box. String may be a <u>StringXX</u> variable or a literal string in quotes.

Message Example

```
Assign String64, "D:\TELIX\DOWN\WORK\TELIX.QWK"

Send "D;Z" Enter
Send Enter
Download String64 with Zmodem

if transferstatus = 0 then
begin
message "Failed to download mail packet!"
exitscript
end
else then
message "Downloaded mail packet successfully."
```

MUL Function

Example

MUL integer1, integer2

The MUL function multiplies the value of *integer1* by *integer2*. Neither integer need be defined prior to an MUL. The first value must be an IntegerXX variable, and the second value may be either a variable or literal value. The comma between the two values is required.

See also

Assign, ADD, DIV, SUB



Assigne Integer30, 25 MUL Integer30, 2

Printer Function

Example

Printer ON | OFF

The Printer command toggles the printer log on and off, just as Ctrl-@ does in terminal mode. You must specify the state you wish the log to be in, **ON** or **OFF**.



Printer On

Printer Off

RenameFile Function

Example

RenameFile string1, string2

The RenameFile function renames the file specified in *string1* to the filename specified in *string2*. Either string may be a <u>StringXX</u> variable or a literal string in quotes.

See also

EraseFile



Assign String24, "C:\TELIX\QWK\TELIX.QWK"
RenameFile String24, "C:\TELIX\QWK\TELIX.BAK"

RunScript Function

Example

RunScript string

The RunScript function loads the script specified in *string* and executes it. When this new script terminates, your script will continue from this position. *String* may be either a <u>StringXX</u> variable or a literal string in quotes.

RunScript places the value returned by the called script into the system variable <u>ReturnCode</u>. All SIMPLE scripts will return a zero. SALT scripts may return varying values.

RunScript Example

```
dial "1"

if ReturnCode > 0 then
  begin
    RunScript "LOGON.SLC"
    RunScript "GETMAIL.SLC"
  end
else then
  begin
    Alarm 1
  Message "Could not connect to system!"
end
```

Send Function

Example

Send string [ENTER]

The Send function sends the data contained in *string* out the connect device, and also to the screen. If the keyword **ENTER** follows *string* then a carriage return will be sent as well. *String* may be a <u>StringXX</u> variable or a string constant in quotes.



Assign String14, "Telix Support"

Send String14 Send "Chatting with Sysop" Enter

Shell Function

Example

Shell

The Shell function opens a DOS window and allows you to execute other programs manually. To return to your script from DOS, simply type "exit" at the DOS prompt.

You must return to the directory you started in if things are expected to function properly. If you "exit" back to Telix while in another directory, the current directory that scripts use will be incorrect and files might not be found where they should be. Be careful using the Shell command.



Shell

Show Function

Example

Show string [ENTER]

The Show function places the data contained in *string* on the screen. If the keyword **ENTER** follows *string*, then a carriage return will be displayed as well. *String* may be a <u>StringXX</u> variable or a string constant in quotes.

Show is very similar to $\underline{\text{Send}}$, but the data is **not** sent over the connect device. Be careful not to confuse Show and Send.



Assign String14, "Telix Support"

Show String14
Show "Chatting with Sysop" Enter

Sound Function

Example

Sound integer1, integer2

The Sound function causes a tone of frequency (pitch) *integer1* to be played on the PC speaker for *integer2* hundredths of a second. You may want to experiment with values for *integer1* to determine acceptable frequencies. Script execution will not continue until the time has elapsed.



Sound 200, 30

Sound 500, 10

SUB Function

Example

SUB integer1, integer2

The SUB function subtracts the value of *integer2* from *integer1*. Neither integer need be defined prior to an SUB. The first value must be an Integer XX variable, and the second value may be either a variable or literal value. The comma between the two values is required.

See also

Assign, ADD, DIV, MUL



<your text>

Time Function Example

Time string

The Time function places the current time into *string*. *String* must be a <u>StringXX</u>-type variable.



Time String16

show "The time is " show String16 ENTER

Upload Function

Example

Upload string WITH protocol

The Upload function acts just as if you had pressed Alt-PgUp and entered a protocol and filename. It will upload the file (or files if a batch protocol is used) indicated by *string*. *String* may be a <u>StringXX</u> variable or a string constant in quotes.

The Upload command uses the protocol specified on the command line by the **WITH** operator. If a protocol is not specified, Telix will prompt for the protocol. Protocols allowed are:

XModem 1K-XModem G-1K-XModem YModem YModem-G ZModem

Remember that uploads usually need to be triggered on the remote site before you can send the file. You will usually need to <u>Send</u> a start command prior to using the Upload command.

The success or failure of the transfer is reported in the special system variable TransferStatus that is described in the <u>If...Then...Else</u> section.



Assign String64, "D:\TELIX\DOWN\WORK\TELIX.REP"

Send "U;Z" Enter
Send Enter
Upload String64 with Zmodem
If TransferStatus = 0 Then
 Message "Download failed!"

Send "U;G" Enter
Send Enter
Upload "C:\TELIX\DOWN\TLX320-1.ZIP" With Ymodem-G
If TransferStatus > 0 Then
 Message "Download succeeded."

UsageLog Function Example

UsageLog ON | OFF

The UsageLog command toggles the Telix usage log on and off. You must specify the state you wish the log to be in, $\bf ON$ or $\bf OFF$.



UsageLog On

UsageLog Off

Wait Function

Wait integer

The Wait function forces the script to pause for *integer* seconds.



Wait 60

SIMPLE Program Control

SIMPLE scripts would be less than useful if there wasn't a way to cause the lines to execute out of order or in repetitive blocks. There are three commands that can cause a SIMPLE script to take on a much higher degree of functionality, and thus complexity. Starting with the easiest one, they are the WaitFor.command, If...Then...Else command, and the WhenEver loop.

See also

WaitFor, If...Then...Else, WhenEver

WaitFor Command

Example

WaitFor string1 [MAXOF integer] THEN string2

Often when automating logons to online systems, you must enter information in response to a certain prompt, but it is not known exactly when that prompt will be ready for your input. Waitfor simulates exactly what you would do when logging onto a system. It waits for the prompt to appear, and then does what you tell it.

The prompt to wait for is given in *string1*, and it may be a literal string in quotes, or a <u>StringXX</u> variable. Case is not significant, and the string must be no more than 40 characters.

The optional **MAXOF** directive tells the WaitFor command how long to wait before giving up. *integer* is the maximum number of seconds to wait. If **MAXOF** is defined, and *integer* seconds elapse without *string1* being received, SIMPLE skips the command given in *string2* and continues with the next command in the script.

string2 is any valid <u>SIMPLE function</u>, including blocks surrounded by <u>Begin</u> and <u>End</u>.



Clark Development's PCBoard BBS prompts the user for various inputs, always in the same order. It will prompt you for your color preference, your first name, your last name, and your password. You can use the WaitFor command in a short SIMPLE script to automate this process as follows:

```
Assign String1, "Jeff"
Assign String2, "Woods"
Assign String101, "first name"
Assign String102, "last name"
Waitfor "you want graphics" MaxOf 30 Then Send "Y Q" Enter
Waitfor String101 MaxOf 10 Then
Send String1 Enter
Waitfor String102 MaxOf 10 Then Send String2 Enter
Waitfor "ssword" MaxOf 10 Then Send BBSPassword Enter
```

If...Then...Else Command

Example

If [NOT] condition THEN command [ELSE [IF] THEN command2] [ELSE [IF] THEN...]

The If conditional is one of the most powerful, and thus complex functions of SIMPLE. It has several options and is relatively freeform, but it must follow certain conventions. For this reason, following the explanation of the If statement, our examples will become more complex, as we build on what has been learned so far.

The general purpose of an If statement is to test to see if a certain condition is true, and to execute certain commands if so, or optionally, certain commands if not.

Condition is the quality you wish to test for being true or false. Conditions may be comparing strings for equality, or inequality, to each other, checking for the existence of a certain file on the disk, or for checking to see if a certain condition exists, such as if Telix is connected to a system. You can check for the opposite of any condition by preceding *condition* with the optional directive **NOT**.

The **THEN** keyword is required for all If statements, and must follow the *condition*.

Command is the action that could be performed based on the result of the *condition*. The *command* may be any <u>SIMPLE function</u>, and can even be a group of instructions marked by <u>Begin</u> and <u>End</u>.

The optional **ELSE THEN** directive tells SIMPLE to perform the actions specified in *command2* if the *condition* was not true. The **ELSE THEN** directive will be discussed further below.

There are five predefined conditions that can be used:

ONLINE You may check the state of the carrier detect signal to determine if you are connected to another system or not. The format of the Online conditional is:

If [NOT] ONLINE [=YES] [=NO] THEN command

YES and **NO** are optional and are only included for clarity. See the <u>Online Examples</u> for more information.

EXIST You may check for the existence of a file on the disk prior to attempting an action on that file. The format of the Exist conditional is:

If [NOT] EXIST string THEN command

String may be a $\underline{StringXX}$ variable or a string constant in quotes. See the \underline{Exist} $\underline{Examples}$ for more information.

BBSNUMBER You may check the value of this system variable to determine which dialing directory entry number you last connected to. The format of the BBSNumber conditional is:

If [NOT] BBSNUMBER = integer THEN command

Integer is any integer number. For example, if you know that the Telix Support BBS is entry # 1 in the current dialing directory, and you wish to find out if we are currently online with this particular system, you could check the BBSNumber variable for the value of 1. See the BBSNumber Example for more information.

RETURNCODE You may check the results of any of three other SIMPLE commands with the ReturnCode system variable. The three commands you can check for success are <u>Dos</u>, <u>Dial</u>, and <u>RunScript</u>. The format of the ReturnCode conditional is as follows:

If [NOT] RETURNCODE = integer THEN command

Integer is any integer number. See the ReturnCode Example for more information.

TRANSFERSTATUS You may check the result of the <u>Upload</u> or <u>Download</u> commands with the TransferStatus system variable. The format of a TransferStatus conditional is as follows:

If [NOT] TRANSFERSTATUS = integer THEN command Integer is any integer number. See the <u>TransferStatus Example</u> for more information.

An optional **ELSE THEN** directive may be included with any **IF** statement to further control the flow of the program. **ELSE THEN** statements may be nested within one another, and infinitely deep. See the <u>Example</u> for more information.



If...Then...Else Examples

```
If Online Then
Begin
    Send "U; Z" Enter
Upload "Telix.REP" with Zmodem
If TransferStatus = 0 Then
    Download "Telix.QWK" with Xmodem
Else If
    Upload "Telix.REP" With Zmodem
Else
    Hangup
End
Else Then
Dial "11"
```

Online Examples

If Online Then
Begin
Download "TELIX.QWK" With ZModem
EraseFile "TELIX.REP"
End

If Not Online Then
 Dial "m1-919-481-9399" MaxOf 50

If Online=No Then Dial "5"

Exist Examples

Assign String1, "D:\TELIX\QWK\TELIX.REP"

If Exist String1 Then
Upload String1 With Zmodem

If Not Exist String1 Then
If Online Then
Begin
Download String1 With Zmodem
EraseFile "ANYFILE"
End



BBSNumber Example

```
If Online Then
  If BBSNumber = 1 Then
Begin
    EraseFile "TELIX.QWK"
    Download "TELIX.QWK" With Zmodem
End
```

Note that the command executed for "If Online" encompasses all of the rest of the above example through the End statement. This is the reason for the indentation as above. It reminds us which lines comprise the command to execute if the condition is true.



ReturnCode Example

In the follow example, assume that you know that the program Foo.EXE returns a DOS Errorlevel of 1 if today is a Saturday, and an Errorlevel of 0 for every other day of the week. Given these conditions, you could call the a BBS to download a mail packet on Saturdays only. Note the RUNSCRIPT directive in the Dial command; we will assume that entry 1 has a linked script which will log us on to the board and finish leaving us at the main prompt.

```
Dos "Foo.EXE"
If ReturnCode = 1 Then
Begin
  Dial "1" Maxof 50 RunScript
  Send "QMAIL4 D;Y"
  Download "TELIX.QWK" With Zmodem
  Hangup
End
```



TransferStatus Example

The following example will try to send a mail packet to the Telix BBS and, if not successful, will try again. The example assumes that we are already logged on and ready to upload.

Send "TELIX.REP" With Zmodem
If TransferStatus = 0 Then
 Send "TELIX.REP" With Zmodem

WhenEver Loop

Example

```
WHENLOOP
```

```
WHENEVER string THEN command
WHENEVER string THEN command
WHENEVER string THEN command
```

ENDWHEN

The Whenever loop is the most powerful, and thus most complex function of SIMPLE. It has several options and is relatively freeform, but it must follow certain conventions.

The purpose of a Whenever loop is best explained in terms of <u>WaitFor</u>. Please be sure you understand Waitfor before continuing here.

One of the inherent problems of WaitFor is that the order the prompts come in must be fixed. One particular bulletin board can randomly prompt your for your birthday as verification of who you really are. Such a prompt renders the WaitFor command fairly useless in such a situation.

Whenever is the answer to this problem. You may set up a group of up to sixteen different strings and can define the proper actions SIMPLE should take whenever that particular prompt comes in. The order of the prompts will not matter. If a prompt is received, the defined action will occur.

A Whenever Loop always begins with the keyword **WhenLoop**.

Immediately following **WhenLoop** is a series of up to sixteen **Whenever** directives. A **whenever** directive defines the string to watch for and the actions to execute when it is received. The keyword **THEN** must appear between the two, exactly as above.

String may be a <u>StringXX</u> variable or a string constant in quotes. Case is not significant, and *string* may be no longer than 40 characters.

Whenever *string* is received, *command* will be executed. The power of the Whenever loop lies in the fact that absolutely any SIMPLE construct may be used here as the command, with the single exception of another Whenever loop. Waitfor, If, or any SIMPLE function may be used.

The keyword **EndWhen** must follow the last **Whenever** directive.

The keyword **QuitWhen** is used to exit from a given **WhenLoop** at any time.



WhenEver Loop Example

Using the example of a BBS which can randomly prompt for your date of birth during the login, the following script will log on to that system, enter the mail door, download a mail packet, and log off.

```
WhenLoop
  Whenever "language t"
                          Then Send "2"
                                                 Enter
  Whenever "first name"
                          Then Send "Jeff"
                                                 Enter
  Whenever "last name"
                          Then Send "Woods"
                                                 Enter
  Whenever "ssword"
                          Then Send BBSPassword Enter
 Whenever "birthday"
Whenever "new mail"
  Whenever "birthday"
                          Then Send "03/25/66" Enter
                          Then Send "N Q"
                                                 Enter
  Whenever "rd Command"
                          Then Send "MAILDOOR" Enter
  Whenever "mail Command" Then
    Begin
      Send "D;Y" Enter
      Waitfor "ready to Send Telix.QWK" MaxOf 300 Then
        Download "TELIX.QWK" With Zmodem
      If TransferStatus = 0 Then
        Begin
          Message "Download Not Successful!"
          Sound 200, 2
        End
      Else
        Begin
          Wait(20)
          Send "G;Y" Enter
          Hangup
          QuitWhen
        End
    End
EndWhen
```

The last Whenever directive specifies to the script that it is to do everything within the outermost Begin/End. Note that the final whenever is in response to a prompt that indicates we are in the mail door. If we are in the mail door, we send a command to start the download of a mail packet. When the packet is ready, we actually attempt download. Upon checking the result of the transfer, we either log off and exit the WhenLoop (we did what we wanted), or we make a beep and go back to the whenloop, which will trigger again on the mail Command prompt, and try to download again.

Study the example above carefully, as it is very typical of a complete SIMPLE script. Be careful when using WhenLoops as the above script can try over and over to download a packet. If there is a problem downloading due to the BBS, you could run up quite a bill. Advanced automation scripts should be in SALT, which allows greater control.