

# Installer

**COLLABORATORS**

	<i>TITLE :</i> Installer		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		February 12, 2023	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Installer</b>	<b>1</b>
1.1	Installer.guide	1
1.2	Installer.guide/Copyrights	1
1.3	Installer.guide/Background	2
1.4	Installer.guide/Overview	3
1.5	Installer.guide/Standard Invocation	3
1.6	Installer.guide/Initial Actions	4
1.7	Installer.guide/Startup Screens	5
1.8	Installer.guide/Installation Actions	5
1.9	Installer.guide/Style Guide	5
1.10	Installer.guide/Scripting Language Tutorial	6
1.11	Installer.guide/Basic Elements	7
1.12	Installer.guide/Escape Characters	8
1.13	Installer.guide/Symbols (Variables)	8
1.14	Installer.guide/Types of Symbols	9
1.15	Installer.guide/Statements	9
1.16	Installer.guide/Data Types	10
1.17	Installer.guide/Special Features	12
1.18	Installer.guide/Miscellaneous	13
1.19	Installer.guide/Installer Language Reference	14
1.20	Installer.guide/VERY IMPORTANT NOTES	14
1.21	Installer.guide/Statements (Ref)	15
1.22	Installer.guide/s_set	16
1.23	Installer.guide/s_symbolset	17
1.24	Installer.guide/s_makedir	17
1.25	Installer.guide/s_copyfiles	18
1.26	Installer.guide/s_copylib	19
1.27	Installer.guide/s_startup	21
1.28	Installer.guide/s_tooltype	22
1.29	Installer.guide/s_textfile	23

---

1.30	Installer.guide/s_execute . . . . .	23
1.31	Installer.guide/s_run . . . . .	24
1.32	Installer.guide/s_rexx . . . . .	24
1.33	Installer.guide/s_makeassign . . . . .	25
1.34	Installer.guide/s_rename . . . . .	25
1.35	Installer.guide/s_delete . . . . .	26
1.36	Installer.guide/s_protect . . . . .	27
1.37	Installer.guide/s_complete . . . . .	28
1.38	Installer.guide/s_message . . . . .	28
1.39	Installer.guide/s_working . . . . .	28
1.40	Installer.guide/s_welcome . . . . .	28
1.41	Installer.guide/Control Statements . . . . .	29
1.42	Installer.guide/cs_if . . . . .	30
1.43	Installer.guide/cs_select . . . . .	30
1.44	Installer.guide/cs_while . . . . .	30
1.45	Installer.guide/cs_until . . . . .	30
1.46	Installer.guide/cs_foreach . . . . .	31
1.47	Installer.guide/cs_sequence . . . . .	31
1.48	Installer.guide/cs_abort . . . . .	31
1.49	Installer.guide/cs_exit . . . . .	31
1.50	Installer.guide/cs_trap . . . . .	32
1.51	Installer.guide/cs_onerror . . . . .	32
1.52	Installer.guide/Debugging Statements . . . . .	32
1.53	Installer.guide/ds_user . . . . .	33
1.54	Installer.guide/ds_debug . . . . .	33
1.55	Installer.guide/User-Defined Procedures . . . . .	33
1.56	Installer.guide/Functions . . . . .	34
1.57	Installer.guide/f_string . . . . .	36
1.58	Installer.guide/f_cat . . . . .	36
1.59	Installer.guide/f_substr . . . . .	36
1.60	Installer.guide/f_strlen . . . . .	37
1.61	Installer.guide/f_transcript . . . . .	37
1.62	Installer.guide/f_tackon . . . . .	37
1.63	Installer.guide/f_fileonly . . . . .	37
1.64	Installer.guide/f_pathonly . . . . .	38
1.65	Installer.guide/f_expandpath . . . . .	38
1.66	Installer.guide/f_askdir . . . . .	38
1.67	Installer.guide/f_askfile . . . . .	39
1.68	Installer.guide/f_askstring . . . . .	39

---

---

1.69	Installer.guide/f_asknumber . . . . .	40
1.70	Installer.guide/f_askchoice . . . . .	40
1.71	Installer.guide/f_askoptions . . . . .	41
1.72	Installer.guide/f_askbool . . . . .	42
1.73	Installer.guide/f_askdisk . . . . .	42
1.74	Installer.guide/f_exists . . . . .	43
1.75	Installer.guide/f_earlier . . . . .	43
1.76	Installer.guide/f_getsize . . . . .	43
1.77	Installer.guide/f_getdevice . . . . .	44
1.78	Installer.guide/f_getdiskspace . . . . .	44
1.79	Installer.guide/f_getsum . . . . .	44
1.80	Installer.guide/f_getversion . . . . .	44
1.81	Installer.guide/f_getenv . . . . .	45
1.82	Installer.guide/f_getassign . . . . .	45
1.83	Installer.guide/f_iconinfo . . . . .	46
1.84	Installer.guide/f_database . . . . .	47
1.85	Installer.guide/f_select . . . . .	48
1.86	Installer.guide/f_patmatch . . . . .	48
1.87	Installer.guide/f_symbolval . . . . .	48
1.88	Installer.guide/f_comparison . . . . .	49
1.89	Installer.guide/f_basicmath . . . . .	49
1.90	Installer.guide/f_logical . . . . .	49
1.91	Installer.guide/f_bit . . . . .	50
1.92	Installer.guide/f_bitshift . . . . .	50
1.93	Installer.guide/f_bittest . . . . .	50
1.94	Installer.guide/Summary of Parameters . . . . .	50
1.95	Installer.guide/Pre-Defined Variables . . . . .	53
1.96	Installer.guide/Installer Language Quick Reference . . . . .	55
1.97	Installer.guide/Overview (Quick) . . . . .	55
1.98	Installer.guide/Quick Language Overview . . . . .	56
1.99	Installer.guide/Pre-Defined Variables (Quick) . . . . .	57
1.100	Installer.guide/Default Help String Variables . . . . .	58
1.101	Installer.guide/Statements (Quick) . . . . .	59
1.102	Installer.guide/Functions (Quick) . . . . .	62

---

# Chapter 1

## Installer

### 1.1 Installer.guide

```

                Installer
*****

    This document contains all the information needed to use the Amiga
    Installer tool.

                Copyrights

                Background

                Overview

                Style Guide
                Read this!

                Scripting Language Tutorial

                Installer Language Reference

                Installer Language Quick Reference
                NOTE: On pre V39 (< Workbench 3.0) AmigaGuide you will see some
    funny marks on the screen. They are V39 formatting commands which make
    the text look nicer under >=V39. They do not reduce the amount of
    information shown in this document.
```

### 1.2 Installer.guide/Copyrights

```
Copyrights & Sentiments
*****
```

The Installer and its documentation is ©1995-96 by ESCOM AG.

---

Based on previous work by Loren Wilton, the V43 Installer and its documentation have been created by Heinz Wrobel. This work is dedicated to Joan Thuesen.

## 1.3 Installer.guide/Background

### Background

\*\*\*\*\*

Installation of applications from floppy disks onto a hard disk has proven to be a very inconsistent and often frustrating endeavor for most end-users. This has been caused by many factors, some of which are:

- a. Many products do not come with any utility or script to install an application on a hard disk.
- b. Many products assume a great deal of familiarity with the startup process of the Amiga and applications, including assigns, device names (as opposed to volume names), etc.
- c. The installation scripts or utilities included with some products vary widely in their ability to deal with different environments and systems.

In 1991, Commodore set out to remedy this situation, by developing a standard tool that developers can include with their products, which provides the user with a standard way to install applications. The Installer's features were based on a number of assumptions:

- a. Installation requirements vary widely--some need assigns, some need new drawers created, some install pieces in system drawers such as a fonts drawer, a 'product' might be just an upgrade and the installation must check to see which version (if any) they currently have installed, etc.
- b. Different users have different levels of comfort and expertise when attempting to install software, and the Installer should be able to accommodate a range of users. Many installation scripts assume a great deal of knowledge, which is very intimidating for a novice.
- c. The Installer tool must be very flexible internally, but present a consistent pleasant graphical user interface to the user that only shows the user information or prompts that they need to see. The Installer should be resolution, color and font sensitive.
- d. Writing scripts to install an application will require some effort, but certainly no more than writing an AmigaDOS shell script equivalent, and the resulting installation procedure will be more friendly, flexible, and much better looking than the latter.

Amiga Technologies improves the Installer to allow even better and

---

more versatile Installation procedures while keeping it fairly general. Many new features have been added during V43 Installer development.

## 1.4 Installer.guide/Overview

### Overview

\*\*\*\*\*

The Installer is a script driven program, that presents a consistent installation environment to the end user. The user never sees the script. Instead they are presented with simple yes/no choices, and may be asked to specify locations to put things on their system.

To accommodate different user levels, they can choose to run the tool in novice, average or expert modes. Scripts can include help text to explain any choices that the user must make. At each step the user is given the option of aborting the installation.

Standard Invocation

Initial Actions

Startup Screens

Installation Actions

## 1.5 Installer.guide/Standard Invocation

### Standard Invocation

=====

The Installer is normally started from a Workbench Project icon which has the same name as the script to interpret and has a default tool of Installer. A number of tooltypes are available to modify the operation of the Installer:

``SCRIPT'`

Path to a script file to be used with Installer.

``APPNAME'`

Name of the application being installed (appears in the startup screen). This MUST be given.

``MINUSER'`

The minimum possible operation mode of the installation for a script. This will be either NOVICE (all decisions made by Installer), AVERAGE (only important decisions made by user) or EXPERT (user confirms almost all actions). The Default is NOVICE.

---



**`DEFUSER`**

Indicates which operation mode button should be initially selected. Same values as MINUSER, with the value of the MINUSER tooltype being the default (which will be NOVICE if MINUSER not defined).

**`NOPRINT`**

If set to FALSE, then the printer option in the log file settings will be ghosted.

**`PRETEND`**

If set to FALSE, indicates that PRETEND mode not available for this script.

**`LANGUAGE`**

Used to set the variable '@language'. The default value for '@language' is the name of the current locale or "english" if there isn't any available. The use of this variable is left up to the install script and should be used to adapt the language for the script's messages to the chosen information. Note that with Installer V42 and better, the current locale as set by the OS will automatically be used if no language has been specified. This tooltype should probably only be used to force a certain language while testing a script or when a system with OS 2.04 is expected to be the target. For the latter, it is recommended that you use the 'getversion' command to check the OS version.

**`LOGFILE`**

The name of the log file that the Installer should use. This must be a full path. The default is 'install\_log\_file'.

**`LOG`**

In NOVICE mode the default is to create a log file (to disk). If this tooltype is set to FALSE, the creation of a log file in NOVICE mode is disabled.

Although the Installer can be started from the Shell, that is not the recommended mode. Shell invocation is provided mainly for script debugging purposes. The command template is:

```
`SCRIPT,APPNAME,MINUSER,DEFUSER,LOGFILE,LANGUAGE,NOPRETEND/S,  
NOLOG/S,NOPRINT/S`
```

## 1.6 Installer.guide/Initial Actions

### Initial Actions

=====

The first thing the Installer does is to compile the installation script into an internal format that can be easily interpreted. If there are syntax errors in the script, they will be caught during this phase.

## 1.7 Installer.guide/Startup Screens

### Startup Screens

=====

Next, the Installer asks the user what Installation Mode to run in, either NOVICE, AVERAGE or EXPERT. If the user chooses NOVICE, they will not be asked any more questions (although they may be requested to do things). In the other user levels, a second display appears asking the user if he wants to install "for real" or "do a dry run", and if he wants a transcription of the installation process written, and if so, whether to a file or to the printer. In EXPERT mode, the user should be asked for every configuration option like installation drawers, while only the most important questions will be asked in AVERAGE mode.

## 1.8 Installer.guide/Installation Actions

### Installation Actions

=====

Now the Installer interprets its internal version of the script. Any commands that call for a user interface will cause the Installer to algorithmically generate a display, always including buttons to allow for context sensitive help and aborting the installation.

## 1.9 Installer.guide/Style Guide

### Style Guide

\*\*\*\*\*

Making an installation script for any typical application is fairly easy with Installer 43.2. But to make a really great installation, please adhere to the rules outlined below:

- Installer V43 provides some new features over older versions like better return values for certain statements or optional proportional rendering for choices. The minimum version necessary is marked in the respective descriptions. Please check the variable '@installer-version' before using these new features. This variable will default to 0 with old versions of Installer like V1.24, so it is safe and easy to check. Please do not do an exact equality check. Check for a minimum revision, just like with libraries.
- Starting with V42 there are some reasonable guidelines for naming things. Please adhere to them or upgrading Installer will be very

hard. Don't use the '@' sign for your variables. This sign marks installer system variables. Please do use a prefix for your variable names like '#' and name your procedures carefully by using a prefix like 'P\_'. Names without a prefix may collide with future enhancements to Installer.

- Always, always, always set up meaningful help texts within your install script.
- Don't install parts of your application into the standard OS drawers like 'DEVS:', 'SYS:', or 'FONTS:' unless absolutely necessary. Many people try to keep their OS installation and applications separate for easier updates. Never forget that 'PROGDIR:' might be what you really want.
- If you can't keep needed files together, at least leave a way to change the destination to something other than e.g. 'LIBS:'.
- Always, always, always allow for a deinstallation in your script. This is another reason to avoid spreading files around too much.
- If you are installing or deinstalling libraries, take care that you don't simply overwrite an existing library or delete it when some other program might have a need for it. If in doubt ask the USER, if she is not a NOVICE.
- Don't keep the user waiting if it is not necessary. If your application fits on one disk, ask all necessary questions first, and install the files afterwards.
- If you have to ask the user for an application directory, make very clear if your script will create the application directory there or if the user has to create and specify the directory himself. This has been the source of much confusion in many scripts.
- Don't ask any questions in NOVICE mode. If you absolutely have to ask questions, don't support NOVICE mode. Any NOVICE mode installation should be capable of running successfully without user interaction.
- Make a difference between AVERAGE and EXPERT mode. In EXPERT mode the user should be able to confirm just about everything. In AVERAGE mode, only very important questions should be asked. Do not let AVERAGE and EXPERT be equal. Give the expert user something to work with and don't confuse the average user.

## 1.10 Installer.guide/Scripting Language Tutorial

### Scripting Language Tutorial

\*\*\*\*\*

The script language of the Installer is based on LISP. It is not difficult to learn, but requires a lot of parentheses. An Installer

---

script can easily be made to look very readable.

- Basic Elements
- Escape Characters
- Symbols (Variables)
- Types of Symbols
- Statements
- Data Types
- Special Features
- Miscellaneous

## 1.11 ~~Installer.guide/~~Basic Elements

Basic Elements  
=====

There are only a few basic elements of the Installer language. Here is a list of these basic elements with a few examples each.

decimal integers

```
'5', '32769', '-3'
```

hexadecimal integers

```
'$a000', '$FB'
```

binary integers

```
'%0010010', '%11'
```

strings

```
"Hello", 'Hello'
```

symbols

```
'#x', '#loopvar', 'P_funcfoo'
```

comments

```
`; this is a comment ( )'
```

```
`( )'
```

for statement definition

space (or any white space)

delimits symbols

---

## 1.12 Installer.guide/Escape Characters

Escape Characters

=====

Escape characters are supported as in the C++ language:

`'\n'`  
newline character

`'\r'`  
return character

`'\t'`  
tab character

`'\h'`  
horizontal tab character (V42.6)

`'\v'`  
vertical tab character (V42.6)

`'\b'`  
backspace character (V42.6)

`'\f'`  
formfeed character (V42.6)

`'\"'`  
double quote

`'\''`  
single quote

`'\\'`  
backslash

`'\ooo'`  
some octal number 'ooo'. You can use '\0' to get a NUL character.  
(V42.6)

`'\xXX'`  
some hex number 'XX'. (V42.6)

NOTE: Depending on the AmigaGuide or MultiView version you are using, you might see an additional backslash character where there should only be one. This should show you one backslash: `'\'`. And this should show two of them: `'\\'`.

## 1.13 Installer.guide/Symbols (Variables)

Symbols (Variables)

=====

---

A symbol is any sequence of characters surrounded by spaces that is not a quoted string, an integer or a control character. This means that symbols can have punctuation marks and other special characters in them. The following are all valid symbols:

- `'x'`
- `'total'`
- `'this-is-a-symbol'`
- `'**name**'`
- `'@#_#@'`

When naming variables, you should not use '@' as prefix for your names, but you should select a prefix like '#' to avoid collisions with future Installer enhancements.

## 1.14 Installer.guide/Types of Symbols

Types of Symbols

=====

There are three types of symbols:

- a. user-defined symbols. These are created using the 'set' function.
- b. built-in function names. These include things like '+' and '\*' as well as textual names such as 'delete' or 'rename'.
- c. special symbols. These are variables which are created by the Installer before the script actually starts to run, and are used to tell the script certain things about the environment. These symbols always begin with an '@' sign. An example is '@default-dest' which tells you the default directory that was selected by the Installer.

For naming conventions, please check the  
Style Guide

.

## 1.15 Installer.guide/Statements

Statements

=====

The format of a statement is:

```
(operator <operand1> <operand2> ...)
```

A statement to assign the value '5' to the variable '#x' would be:

```
(set #x 5)
```

You can read this as "set '#x' to 5". Note that the variable '#x' does not have to be declared - it is created by this statement.

Note that there is no difference between operators and functions - the function

```
    'set'
```

, and the arithmetic operator '+' are both used exactly the same way.

Combining statements: A statement can be used as the operand to another statement as follows:

```
(set #var (+ 3 5))
```

In this case, the statement '(+ 3 5)' is evaluated first, and the result is 8. You can think of this as having the '(+ 3 5)' part being replaced by an 8. So now we are left with:

```
(set #var 8)
```

which is the same form as the first example.

Note that the '(+ 3 5)' part actually produced a value: 8. This is called the result of the statement. Many statements return results, even some that might surprise you, such as 'set' and 'if'.

## 1.16 Installer.guide/Data Types

Data Types

=====

All data types in the Installer are dynamic, that is to say the type of a variable is determined by the data it contains. So if you assign the string "Hello, World" to the variable '#x', then '#x' will be of type STRING. Later you can assign an integer to '#x' and '#x' will be of type INTEGER. When using variables in expressions, the interpreter will attempt to convert to the proper type if possible.

Special forms: There are two exceptions to the form of a statement. The first type is used for string substitution: If the first item in parentheses is a text string rather than a function name, the result of that clause is another string that is created by taking the original string and performing a C-style 'sprintf'-like formatting operation on it, using the other arguments of the statement as parameters to the formatting operation.

---

Thus the statement:

```
("My name is %s and I am %ld years old" "Mary" 5)
```

Becomes:

```
"My name is Mary and I am 5 years old"
```

NOTE: since the formatting operation uses the ROM `'RawDoFmt()'` routine, decimal values must always be specified with `'"%ld"'` rather than `'"%d"'` (The interpreter always passes numeric quantities as 32 bit longwords). Note that a variable containing a string may be used rather than the string itself.

The second type of exception occurs if the elements in parentheses are themselves statements in parentheses. In this case, the interpreter assumes that all the elements are statements to be executed sequentially.

For example, this statement sets the value of three different variables: `'#var1'`, `'#var2'`, and `'#var3'`.

```
((set #var1 5) (set #var2 6) (set #var3 7))
```

What this feature does is allow the language to have a block structure, where an `'if'` statement can have multiple statements in its `'then'` or `'else'` clause. Note that the result of this statement will be the result of the last statement in the sequence.

Complex statements: Here is an example of how statements in the script language can be combined into complex expressions. We will start with an `'if'` statement. The basic format of an `'if'` statement is:

```
(if <condition> <then-statement> [<else-statement>])
```

The condition should be a statement which returns a value. The `'then'` and optional `'else'` parts should be statements. Note that if the `'then'` or `'else'` statements produce a result, then the `'if'` statement will also have this result.

Our first example is a rather strange one: Using an `'if'` statement to simulate a boolean `'not'` operator. Note that there are actually easier ways in the script language to do this.

```
(set #flag 0) ; set a flag to FALSE
```

```
(set #flag (if #flag 0 1)) ; a Boolean NOT
```

Basically, the `'if'` statement tests the variable `'flag'`. If flag is non-zero, it produces the value `'0'`. Otherwise, the result is `'1'`. In either case, `'flag'` is set to the result of the `'if'` statement.

Now, let's plug some real statements into our `'if'` statement.

```
(if #flag ; conditional test
  (message "'flag' was non-zero\n") ; "then" clause.
```



```

    (message "'flag' was zero\n")          ; "else" clause.
) ; if                                     ; closing parenthesis

```

Note the style of the indenting. This makes for an easier to read program. The preferred indent is four spaces.

Now, we'll add a real condition. '=' tests for equality of the two items.

```

(if (= #a 2)                               ; conditional test
    (message "a is 2\n")                   ; "then" clause
    (message "a is not 2\n")              ; "else" clause
) ; if                                     ; closing parenthesis

```

Finally, just to make things interesting, we'll make the 'else' clause a compound statement. Note the extended use of parentheses to generate exactly the two needed clauses. Installer won't check things if you mess up here!

```

(if (= #a 2)                               ; conditional test
    (                                       ; "then" clause
        (message "a is 2\n")
    )
    (                                       ; "else" compound statement
        (message "a is not 2\n")
        (set #a 2)
        (message "but it is now!\n")
    )
) ; if                                     ; end of compound statement
; end of if

```

## 1.17 Installer.guide/Special Features

### Special Features

=====

When the Installer starts running, it attempts to determine the best place to install the application. Any volume named 'WORK:' is given preference, as this is the standard way that an Amiga comes configured from Amiga Technologies. Starting with V42 the largest writable partition available will also be under consideration as default destination. This covers the typical cases.

There are two keyboard shortcuts. Whenever there is a 'Help' button active, pressing the HELP key will also bring up the help display. Whenever there is an 'Abort' button active, pressing ESC brings up the abort requester. Also, whenever the Installer is busy, pressing ESC brings up the abort requester - there is text in the title bar to that effect.

If an application must have assigns or other actions performed during system boot, the Installer will add these to a file named 'S/User-Startup' on the boot volume. If this file isn't available, 'S:User-Startup' will be tried. The Installer will then add the lines

```

if exists S:user-startup
execute S:user-startup
endif

```

to the user's 'Startup-Sequence'. The Installer will attempt to determine the true boot volume of the system when looking for the 'Startup-Sequence' and 'User-Startup'. It can handle any AmigaDOS scripts executed from 'Startup-Sequence' up to 10 levels of nesting.

The Installer can create an assign to just a device, volume or logical assignment. This comes in handy when you want to update an application which comes on a volume named 'MyApp:', but the installed version is in a directory with the logical assign 'MyApp:'!

The Installer always copies files in the AmigaDOS 'CLONE' mode, meaning all the protection bits, filenotes and file dates are preserved. When copying files the Installer gives a "fuelgauge" readout of the progress of the copy.

The Installer can find the version number of any executable file that has either a RomTag with an ID string (such as libraries and devices) or has a version string conforming to that given in the Amiga User Interface Style Guide. The Installer can also checksum files.

## 1.18 Installer.guide/Miscellaneous

Miscellaneous  
=====

To perform a set of actions on all the contents of a directory matching a pattern you can use the 'foreach' operator. To perform a set of actions on an explicit set of files, the following Installer statements can be used as a template:

```

(set #n 0)
(while (set thisfile (select #n "file1" "file2" "file3" ""))
(
  (set #n (+ #n 1))
  (... your stuff involving this file ...)
)
) ; while

```

Note that an empty string is considered a FALSE value to any condition operator.

To run an external Shell command which normally requires user input, redirect the input from a file with the needed responses. For example, to format a disk one could combine the statement shown below with a file which contains only a newline character.

```
(run "format <nl_file drive DF0: name ToBeEmpty")
```

## 1.19 Installer.guide/Installer Language Reference

### Installer Language Reference

\*\*\*\*\*

VERY IMPORTANT NOTES

Statements

Control Statements

Debugging Statements

User-Defined Procedures

Functions

Summary of Parameters

Pre-Defined Variables

## 1.20 Installer.guide/VERY IMPORTANT NOTES

### Very Important Notes

=====

- a. 'diskimage' is a reserved name (V42). Don't try to use this name. Please read on!
- b. Starting with V42 there are some reasonable guidelines for naming things. Please adhere to them or upgrading Installer will be very hard. Don't use the '@' sign for your variables. This sign marks installer system variables. Please do use a prefix for your variable names like '#' and name your procedures carefully by using a prefix like 'P\_'. Names without a prefix may collide with future enhancements to Installer.
- c. Remember that Installer doesn't support local variables.
- d. When the script exits either by coming to the end or via the 'exit' statement, a message will be displayed saying where the application was installed and where the logfile (if any) was written. Note that you must store in '@default-dest' where you actually installed the application (see '@default-dest' below).
- e. A newline character (\n, '0x0a') will cause a line break when the Installer performs word-wrapping. A hard-space (ALT-space, '0xa0') will prevent a word break when the Installer performs word-wrapping. Also, quoted sections will be considered one word for word-wrapping purposes. For example, if the following help text was used:

"The disk name \"FrameZapper 2.0\" is needed to complete installation."

then the text "FrameZapper 2.0" will not have a word break before the "2".

- f. The maximum size of a string in a script is 512 bytes. The maximum size of any string variable is 10000 bytes. If you need to create long help text for example, break it into 512 byte chunks and then use the automatic string concatenation ability of the Installer to create the final, larger string. Also, don't overlook the use of line continuation of strings in scripts to make your scripts more manageable. If you ever find that the Installer reports a stack overflow error, look to see if it was caused by too many small strings being concatenated and merge them into larger blocks.
- g. File name and directory name wildcard patterns specified in a script must not be longer than 64 characters.

## 1.21 Installer.guide/Statements (Ref)

Statements

=====

Variable management

`'set'`

`'symbolset'`  
(V42.9)

Help with files

`'mkdir'`

`'copyfiles'`

`'copylib'`

`'startup'`

`'tooltype'`

`'textfile'`

Run external parts of the installation

`'execute'`

`'run'`

---

```
`rexx'  
More help with files
```

```
`makeassign'
```

```
`rename'
```

```
`delete'
```

```
`protect'  
Inform the user
```

```
`complete'
```

```
`message'
```

```
`working'
```

```
`welcome'
```

## 1.22 Installer.guide/s\_set

The `'set'` statement

-----

```
(set <varname> <value> [<varname2> <value2> ...])
```

Set the variable `'<varname>'` to the indicated value. If `'<varname>'` does not exist it will be created. Set returns the value of the last assignment.

NOTE: All variables are typeless, and any variable may be used wherever a string could be used. All variables are global. For naming conventions, please check the  
Style Guide

.

The `'set'` statement can be used to convert a string to an integer value:

```
(set <integer-var> (+ <string-var>))
```

To do the reverse, use

```
`cat'
```

.

## 1.23 Installer.guide/s\_symbolset

The `'symbolset'` statement (V42.9)

```
-----
(symbolset <symbolname> <value> [<symbolname2> <value2> ...])
```

Set the variable that is named by the contents of the string variable or expression `'<symbolname>'` to the indicated value. If the variable that `'<symbolname>'` names does not exist, it will be created. The statement `'symbolset'` returns the value of the last assignment.

Together with the function `'symbolval'`, this function is intended to allow somewhat dynamic handling of variable names, e.g. to create variables with an arbitrary index in the name.

NOTE: You should read the description of `'set'`, as the rest of the semantics are the same.

## 1.24 Installer.guide/s\_mkdir

The `'mkdir'` statement

```
-----
(mkdir <name> <parameters>)
```

Creates a new directory. Starting with Installer V42.9, this statement will try to create the complete path with all specified intermediate directories. Parameters:

`'prompt'`  
tell the user what's going to happen.

`'help'`  
text of help message

`'infos'`  
create an icon for directory

`'confirm'`  
if this option is present, the user will be prompted, else the directory will be created silently.

`'safe'`  
make directory even if in PRETEND mode

---

## 1.25 Installer.guide/s\_copyfiles

The `'copyfiles'` statement

-----

```
(copyfiles <parameters>)
```

Copies one or more files from the install disk to a target directory. Each file will be displayed with a checkmark next to the name indicating if the file should be copied or not. Note that a write protected file is considered "delete protected" as well. Parameters:

`'prompt'`

`'help'`

tell the user what's going to happen.

`'source'`

name of source directory or file.

`'dest'`

name of destination directory, which is created if it doesn't exist. Note that both source and dest may be relative pathnames.

`'newname'`

if copying one file only, and file is to be renamed, this is the new name.

`'choices'`

a list of files/directories to be copied (optional)

`'all'`

all files/directories in the source directory should be copied.

`'pattern'`

indicates that files/directories from the source dir matching a pattern should be copied. The pattern should be no more than 64 characters long. Note that only one of `'choices'`, `'all'` or `'pattern'` should be used at any one time.

`'files'`

only copy files. By default the Installer will match and copy subdirectories.

`'infos'`

switch to copy icons along with other files/directories.

`'noposition'`

reset the position of every icon copied.

`'fonts'`

switch to not display `'font'` files, yet still copy any that match a directory that is being copied.

`'nogauge'`

don't display the status indicator.

`'(optional <option> <option> ...).'`  
dictates what will be considered a failure on copying. The first three options are mutually exclusive (they may not be specified together).

`'fail'`  
Installer aborts if could not copy (the default).

`'nofail'`  
Installer continues if could not copy.

`'oknodelete'`  
aborts if can't copy, unless reason was "delete protected".

The next two options may be used with any other `'optional'` options.

`'force'`  
unprotect destination

`'askuser'`  
ask user if the file should be unprotected (but not in novice) In the case of `'askuser'`, the default for novice mode is an answer of "no". Therefore, you may want to use `'force'` to make the novice mode default answer appear to be "yes".

`'(delopts <option> <option> ...).'`  
removes options set by `'optional'`

`'confirm'`  
if this option is present, user will be prompted to indicate which files are to be copied, else the files will be copied silently.

`'safe'`  
copy files even if in PRETEND mode.

`'compression'`  
if this flag is set, all the source files must have been compressed with the Un\*x `'compress'` LZW algorithm, version 3.0 or better. The files will automatically be uncompressed while copying them to the destination. Note that compression with more than 13 bits takes up a really significant amount of memory when uncompressing a file. The maximum number of bits supported is 16. This needs close to 500KB of memory while uncompressing! Make a wise choice for the maximum amount of bits when compressing a file or the uncompression might fail due to lack of memory!

WARNING: The compression implementation is still highly experimental and in alpha stadium. You may play with it and submit bug reports. DO NOT USE THIS FOR PRODUCTION CODE!

## 1.26 `Installer.guide/s_copylib`

The `'copylib'` statement

-----



(copylib <parameters>)

Copies one file using version checking; i.e., it only overwrites an existing file if the new file has a higher version/revision number. The 'copylib' statement will create the destination directory as long as there is only one level missing. For example, copying to a non-existent 'DEVS:midi' would create the directory 'midi', but copying to 'DEVS:midi/extra' where neither 'midi' nor 'extra' exists would fail. Note that a write protected library file is considered "delete protected" as well. Parameters:

'prompt'

'help'

tell the user what's going to happen.

'confirm'

if this option is present, user will be prompted to confirm the copy operation, else the files will be copied silently. Note that an EXPERT user will be able to overwrite a newer file with an older one.

'safe'

copy the file even if in PRETEND mode.

'source'

name of source file.

'dest'

name of destination directory, which is created if it doesn't exist. Note that both source and dest may be relative pathnames.

'newname'

if the file is to be renamed, this is the new name.

'infos'

switch to copy the icon along.

'noposition'

reset the position of every icon copied.

'nogauge'

don't display the status indicator.

'(optional <option> <option> ...)'

dictates what will be considered a failure on copying. The first three options are mutually exclusive (they may not be specified together).

'fail'

Installer aborts if could not copy (the default).

'nofail'

Installer continues if could not copy.

'oknodelete'

aborts if can't copy, unless reason was "delete protected"

The next two options may be used with any other 'optional' options.

'force'

unprotect destination

'askuser'

ask user if the file should be unprotected (but not in novice) In the case of 'askuser', the default for novice mode is an answer of "no". Therefore, you may want to use 'force' to make the novice mode default answer appear to be "yes".

'(delopts <option> <option> ...)'

removes options set by 'optional'

## 1.27 Installer.guide/s\_startup

The 'startup' statement

-----

(startup <appname> <parameters>)

This command edits the 'S:User-Startup' file, which is executed by the user's 'Startup-Sequence' (Installer will modify the user's 'Startup-Sequence' if needed, although in a friendly way). The 'command' parameter is used to declare AmigaDOS command lines which will be executed. The command lines are grouped by application, using the supplied argument 'appname'. If there is already an entry in 'S:User-Startup' for that application, the new command lines will completely replace the old. The command lines for other applications will not be affected. Note: The prompt and help parameters for the 'startup' statement are only used by the confirmation display to edit 'User-Startup'. This only happens in NORMAL or EXPERT mode, or in NOVICE mode if the 'confirm' parameter is included. Parameters:

'prompt'

'help'

tell the user what's going to happen.

'confirm'

if this option is present, user will be prompted to confirm the operation, otherwise the modification proceeds silently.

'command'

used to declare an AmigaDOS command line to be executed at system startup.

Note that Installer will try to locate the User-Startup file on the boot volume in the script directory. Only if this is not available, the assign 'S:' will be used.

## 1.28 Installer.guide/s\_tooltype

The 'tooltype' statement

-----

```
(tooltype <parameters>)
```

Modify an icon's tool type and more. Normally the new tool type values will be set up in advance by various statements in the install language (i.e. the user does not actually have to type in the tooltype values). For example, you could use an 'askchoice' to ask the user what type of screen resolution they want and then format the tooltype string based on their choice. The 'tooltype' operation merely asks for a confirmation before actually writing.

Note that the 'settooltype' parameter has either one or two arguments, in the general form '(settooltype "tooltype")' or '(settooltype "tooltype" "toolvalue)'. The first form is used to delete a tooltype from an icon. The second form will set a new value for a new or existing tooltype. The tooltype name will always be forced to uppercase. Parameters:

'prompt'

'help'

tell the user what's going to happen.

'dest'

the name of the icon to be modified. There is no need to specify a '.info' extension.

'settooltype'

the tooltype name and value string

'setdefaulttool'

default tool name for a project

'setstack'

set size of stack

'noposition'

reset icon position to NOICONPOSITION. Be careful, don't arbitrarily mess with the user's icon positions on updates!

'setposition (V42.12)'

Two integer values giving the new icon position in X and Y direction. Do not use this lightly. It is intended to keep icon positions on updates with help of the new function

'iconinfo'

. Arbitrarily changing icon positions will lead to annoyed users due to different Workbench and font setups.

'swapcolors'

OBSOLETE! DO NOT USE THIS!

'confirm'

---

if this option is present, the user will be asked for confirmation, otherwise the modification proceeds silently.

``safe'`  
make changes even if in PRETEND mode

## 1.29 Installer.guide/s\_textfile

The ``textfile'` statement  
-----

(textfile <parameters>)

Creates a text file from other textfiles or computed text strings. This can be used to create configuration files, AREXX programs or execute scripts. Parameters:

``prompt'`  
``help'`  
tell the user what's going to happen.

``dest'`  
the name of the text file to be created.

``append'`  
a string to be appended to the new text file.

``include'`  
a text file to be appended to the new text file. If the specified file does not exist, it will be treated as an empty file and ignored.

``confirm'`  
if this option is present, the user will be asked for confirmation, otherwise the writing proceeds silently.

``safe'`  
create file even if in PRETEND mode

## 1.30 Installer.guide/s\_execute

The ``execute'` statement  
-----

(execute <argument> ...)

Executes an AmigaDOS script with the arguments given. Parameters:

``prompt'`

---

```

`help'
    tell the user what's going to happen.

`confirm'
    if this option is present, the user will be asked for confirmation,
    otherwise the execute proceeds silently.

`safe'
    execute script even if in PRETEND mode

    This statement returns the primary result of the script directly,
    and, starting with Installer V42, the secondary result in the variable
    `@ioerr'.

```

### 1.31 Installer.guide/s\_run

The `run' statement

-----

```
(run <argument> ...)
```

Executes a compiled program with the arguments given. Parameters:

```

`prompt'
`help'
    tell the user what's going to happen.

`confirm'
    if this option is present, the user will be asked for confirmation,
    otherwise the run proceeds silently.

`safe'
    execute script even if in PRETEND mode

    This statement returns the primary result of the program directly,
    and, starting with Installer V42, the secondary result in the variable
    `@ioerr'.

```

### 1.32 Installer.guide/s\_rexx

The `rexx' statement

-----

```
(rexx <argument> ...)
```

Executes an ARexx script with the arguments given. If the ARexx server is not active, an error will be generated. Parameters:

```

`prompt'
`help'

```

---

tell the user what's going to happen.

``confirm'`

if this option is present, the user will be asked for confirmation, otherwise the rexx script proceeds silently.

``safe'`

execute script even if in PRETEND mode

This statement returns the primary result of the script directly, and, starting with Installer V42, the secondary result in the variable `@ioerr`.

### 1.33 Installer.guide/s\_makeassign

The ``makeassign'` statement

```
(makeassign <assign> [<path>] (parameters))
```

Assigns `<assign>` to `<path>`. If `<path>` is not specified, the assignment is cleared. Parameters:

`safe`

assign even if in PRETEND mode

NOTE: `<assign>` must be supplied without a colon; i.e. `"ENV"` not `"ENV:"`.

### 1.34 Installer.guide/s\_rename

The ``rename'` statement

```
(rename <oldname> <newname> <parameters>)
```

Renames a file or directory. If the `'disk'` parameter is given, then this command relabels the disk named `<oldname>` to `<newname>`. When relabeling a disk, only include a colon in the old name.

Returns 1 if the rename was successful, 0 if it failed. Parameters:

``prompt'`

``help'`

tell the user what's going to happen.

``confirm'`

if this option is present, the user will be asked for confirmation, otherwise the rename proceeds silently.

`'disk'`

switch to get rename to relabel a disk.

```
`safe'
  rename even if in PRETEND mode
```

## 1.35 Installer.guide/s\_delete

The 'delete' statement

```
-----
```

```
(delete <file> <parameters>)
```

Delete a file. Note that a write protected file is considered "delete protected" as well. Starting with Installer V42.9, you can specify an AmigaDOS pattern and the 'all' option. Parameters:

```
`prompt'
`help'
  tell the user what's going to happen.
```

```
`confirm'
  if this option is present, the user will be asked for confirmation,
  otherwise the delete proceeds silently.
```

```
`(optional <option> <option> ...)'
  should deletions be forced. options:
```

```
  ``force''
    unprotect destination
```

```
  ``askuser''
    ask user if the file should be unprotected (but not in novice
    mode)
```

In the case of 'askuser', the default for novice mode is an answer of "no". Therefore, you may want to use 'force' to make the novice mode default answer appear to be "yes".

```
`(delopts <option> <option> ...)'
  removes options set by "optional"
```

```
`safe'
  delete even if in PRETEND mode
```

```
`infos'
  also delete corresponding info file. Do not use this option
  together with 'all'.
```

```
`all'
  check all matching subdirectories, too. (V42.9)
```

NOTE: Please use the pattern matching feature and subdirectory scanning with care. The user will not be able to abort while this command is executing. It might be unwise to delete a huge

amount of data in one step.

## 1.36 Installer.guide/s\_protect

The 'protect' statement

---

```
(protect <file> [<string of flags to change>] [<decimal mask>] <parameters>)
```

Either gets the protection status of a file (if a second argument is not given), or sets it. Two methods exist for setting the status: string (e.g. `""+rgr -wgw +e -dgd""`) or numeric (e.g. 5). The string method allows the changing of any of the flags individually, while numeric writes to all flags at once (possibly changing bits unintendedly). The bits in the binary representation of the decimal mask correspond to the flags in the following manner:

```
8 7 6 5 4 3 2 1  <- Bit number

h s p a r w e d  <- corresponding protection flag

^ ^ ^ ^ ^ ^ ^ ^
| | | | | | | |
| | | | | | | +- \
| | | | | | +---- | 0 = flag set
| | | | | +----- | 1 = flag clear
| | | | +----- /
| | | |
| | | |
| | | +----- \
| | +----- | 0 = flag clear
| +----- | 1 = flag set
+----- /
```

Note that the meaning of the bits in the numeric value follows the DOS convention that a 1 in the high four bits (flags "hspa") indicates that the flag is set, while a 1 in the lower four bits (flags "rwed") indicates that the flag is cleared. You can use the `""g""` and `""o""` modifiers to set group and other flags. They are valid for all the immediately following `""rwed""` flags.

Please avoid using the `""h""` bit. Its meaning is undefined and it may be reused by Amiga Technologies.

When setting bits, 'protect' returns 1 if the attempt succeeded, else it returns a 0. Getting the bits returns either the numeric value of the protection status (see interpretation, above) or -1 upon failure. Parameters:

'safe'

change protection even if in PRETEND mode

---



### 1.37 Installer.guide/s\_complete

The 'complete' statement  
-----

```
(complete <number>)
```

This statement is used to inform the user how complete the installation is. The number (which must be between 0 and 100) will be printed in the title bar of the Installer window with a '%' sign.

### 1.38 Installer.guide/s\_message

The 'message' statement  
-----

```
(message <string> <string> ... <parameters>)
```

This statement displays a message to the user in a window, along with Proceed, Abort and optional Help buttons. Note that messages are not printed when running at user level 0 (novice) if 'all' is not specified as option. Parameters:

'all'

If you specify this optional parameter, the message will be displayed for all user levels (V42.4). Use this wisely. Don't confuse a NOVICE user with messages.

### 1.39 Installer.guide/s\_working

The 'working' statement  
-----

```
(working <string> <string> ...)
```

The strings will be concatenated to form a message which will appear below a standard line that reads "Working on Installation". Useful if you are doing a long operation other than file copying (which has its own status display).

### 1.40 Installer.guide/s\_welcome

The 'welcome' statement  
-----

```
(welcome <string> <string> ...)
```

---

Installer looks for the occurrence of this statement in a script file during compilation. If it does not exist (as is the case for older scripts) the 'Welcome to the <APPNAME> App installation utility' display is presented to the user as soon as compilation has finished. If this statement is present, Installer will not put up the 'Welcome...' display until the 'welcome' statement is reached. This allows for the execution of code before the first displays come up. Note that the state of the '@user-level' and '@pretend' variables will be based on the initial defaults including any modification by tooltypes. The string arguments are prepended to the standard help text for whichever of the two initial displays appears first.

## 1.41 Installer.guide/Control Statements

### Control Statements

=====

NOTE: Strings can be used as the result of a test expression. An empty string is considered a FALSE value, all others are considered TRUE.

Conditional flow control change

'if'

'select'

Loop constructions

'while'

'until'

'foreach'

Linear flow of statements

'Statement sequence'

Ending a script

'abort'

'exit'

Error handling

'trap'

'onerror'

## 1.42 Installer.guide/cs\_if

The 'if' statement  
-----

```
(if <expression> <>true-statement> <>false-statement>)
```

Operates as a standard 'if'-'then' statement.

## 1.43 Installer.guide/cs\_select

The 'select' statement  
-----

```
(select <n> <item1> <item2> ...)
```

Only the selected element will be evaluated. In this manner, 'select' can be used as a case select construct.

## 1.44 Installer.guide/cs\_while

The 'while' statement  
-----

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

Operates as a standard "do-while" statement.

## 1.45 Installer.guide/cs\_until

The 'until' statement  
-----

```
(until <expression> <statement> ... )
```

Operates as a standard "do-until" statement.

---

## 1.46 Installer.guide/cs\_foreach

The 'foreach' statement

-----

```
(foreach <drawer name> <pattern> <statement>)
```

For each file or directory matching the pattern located in the given drawer, <statement> will be executed. The special variables '@each-name' and '@each-type' will contain the filename and the DOS object type, respectively. (By DOS object type we mean the same value as found in 'fib\_DirEntryType' if one were to 'Examine()' the object.) Negative values for the DOS object type are for files, positive values represent directories. Patterns specified in a script may not be longer than 64 characters.

## 1.47 Installer.guide/cs\_sequence

Statement sequence

-----

```
((...) (...) (...))
```

Execute a sequence of statements. The statements in the parentheses will be executed in order. This is not needed at topmost level. If the sequence of statements is an argument to a statement or function, the value of the sequence will be the value of the last statement executed in the sequence.

## 1.48 Installer.guide/cs\_abort

The 'abort' statement

-----

```
(abort <message> <message> ...)
```

Exits the installation procedure with the given messages and then processes the onerror statements (if any).

## 1.49 Installer.guide/cs\_exit

The 'exit' statement

-----

```
(exit <string> <string> ... (quiet))
```

This causes normal termination of a script. If strings are provided, they are displayed. The "done with installation" message is then displayed. The "onerror" statements are not executed. If (quiet) is specified, the final report display is skipped.

## 1.50 Installer.guide/cs\_trap

The 'trap' statement

```
(trap <trapflags> <statements>)
```

Used for catching errors. Works much like C 'longjmp', i.e. when an error occurs, control is passed to the statement after 'trap'. 'trapflags' determine which errors are trapped. The trap statement itself returns the error type or zero if no error occurred. The current error type values are:

```
1 == user aborted
2 == ran out of memory
3 == error in script
4 == DOS error (see '@ioerr' below)
5 == bad parameter data
```

## 1.51 Installer.guide/cs\_onerror

The 'onerror' statement

```
(onerror <statements>)
```

When a fatal error occurs that was not trapped, a set of statements can be called to clean-up after the script. These statements are logged in by using the onerror construct. Note that onerror can be used multiple times to allow context sensitive termination.

## 1.52 Installer.guide/Debugging Statements

Debugging Statements

```
=====
```

---

```

`user`

`debug`

```

### 1.53 Installer.guide/ds\_user

The `user` statement

-----

```
(user <user-level>)
```

Used to change the user level of the current installation. This statement should only be used when debugging scripts. Remove such statements from any script before distribution of your product. Returns the current user level.

### 1.54 Installer.guide/ds\_debug

The `debug` statement

-----

```
(debug <anything> <anything> ...)
```

When the Installer is run from a Shell, `debug` will print the values of the parameters with a space between each parameter. For example, the statements

```
(set #myvar 2)
(debug "The value of 'myvar' is" #myvar)
```

will print `The value of myvar is 2`. If the parameter is an uninitialized variable, then debug will print `` as its value.

### 1.55 Installer.guide/User-Defined Procedures

User-Defined Procedures

=====

The Installer has user-defined procedures (subroutines). This functionality is currently very primitive. There are no local variables. Starting with Installer V42.7, there is a convenient way to pass arguments though, which is described below. To define a new procedure, use the `procedure` command:

```
(procedure <procedure-name> [<args>] <statements>)
```

You can then call the procedure like this:

---

(<procedure-name>)

Note that '<procedure-name>' is not a string, just a symbolic name. You should use a prefix like 'P\_' for your names. Otherwise they might collide with future enhancements to Installer.

The return value of a procedure is the value of the last statement executed in the sequence of statements within the procedure.

Easier argument passing is available starting with Installer V42.7. '<args>' is a list of variable names. If you call the procedure with any arguments, the variables will be set with these arguments. If you leave out any arguments at the end, the corresponding variables will retain their previous value. An example:

```
(procedure P_ADDMUL arg1 arg2 arg3
  (* (+ arg1 arg2) arg3)
)

(message (P_ADDMUL 1 2 3))      ; shows 9
(message (P_ADDMUL 4 5))      ; shows 27 as arg3 stays the same
```

## 1.56 Installer.guide/Functions

Functions

=====

String management

String substitution

'cat'

'substr'

'strlen'

'transcript'

Handling of file and path names

'tackon'

'fileonly'

'pathonly'

'expandpath'

Query the user

`askdir`

`askfile`

`askstring`

`asknumber`

`askchoice`

`askoptions`

`askbool`

`askdisk`

`exists`

`earlier`

Obtain DOS information

`getsize`

`getdevice`

`getdiskspace`

`getsum`

`getversion`

`getenv`

`getassign`

Information about icons

`iconinfo`

(V42.12)

Obtain configuration information

`database`

Selective value retrieval

`select`

`patmatch`

`symbolval`

(V42.9)

All sorts of math

---



Value comparison  
Math 101  
Logical functions  
Bitwise logical functions  
Bitshifting functions  
Bittesting

## 1.57 Installer.guide/f\_string

The String Substitution Function  
-----

```
(<string> <arguments> ...)
```

The "string substitution function". Whenever a text string is the first item in a parenthesized group, the arguments will be substituted into the string using the Exec function 'RawDoFmt()'.

NOTE: This function does no argument type checking.

## 1.58 Installer.guide/f\_cat

The 'cat' function  
-----

```
(cat <string> <string> ...)
```

Concatenates the strings and returns the resulting string.

To convert an integer to a string, use the 'cat' function. All integer arguments to 'cat' are converted to strings during concatenation. Use

```
'set'
```

```
, to convert a string to an integer. You might
```

want to look at

```
'symbolset'
```

```
, too.
```

## 1.59 Installer.guide/f\_substr

---

The `'substr'` function

-----

```
(substr <string> <start> [<count>])
```

Returns a substring of `'string'`, beginning with the character at offset `'start'` (offset begins with 0 for the first character) and including `'count'` characters. If `'count'` is omitted, the rest of the string (to its end) is returned.

## 1.60 Installer.guide/f\_strlen

The `'strlen'` function

-----

```
(strlen <string>)
```

Returns the length of the given string.

## 1.61 Installer.guide/f\_transcript

The `'transcript'` function

-----

```
(transcript <string> <string> ...)
```

Concatenates the strings, appends a newline and then prints the resulting string to the transcript file (if any).

## 1.62 Installer.guide/f\_tackon

The `'tackon'` function

-----

```
(tackon <path> <file>)
```

Concatenates the filename to the pathname and returns resulting string. It correctly deals with a leading `'/'` or any `':'` in the `'file'` parameter.

## 1.63 Installer.guide/f\_fileonly

The 'fileonly' function

-----  
(fileonly <path>)

Returns only the file part of a pathname.

## 1.64 Installer.guide/f\_pathonly

The 'pathonly' function

-----  
(pathonly <path>)

Returns only the non-file part of a pathname.

## 1.65 Installer.guide/f\_expandpath

The 'expandpath' function

-----  
(expandpath <path>)

Returns the full path, given a shortened path. For example, it might expand 'SYS:c' to 'System2.x:c'.

## 1.66 Installer.guide/f\_askdir

The 'askdir' function

-----  
(askdir <parameters>)

Asks the user for a directory name, with a scrolling list requester. The user can either create a new directory or specify an existing one. If the user cancels, the routine will cause an abort.

NOTE: It is always best to first insure that the volume you want is mounted by using

'askdisk'

.

Parameters:

'prompt'

'help'

---

tell the user what's going to happen.

``default'`

default name of directory to be selected. Note that this may be a relative pathname.

``newpath'`

allows non-existent paths to be supplied as the default drawer.

``disk'`

show drive list first.

``assigns'`

indicates that logical assigns should satisfy requests as well.

## 1.67 Installer.guide/f\_askfile

The ``askfile'` function

-----

`(askfile <parameters>)`

Asks the user for a file name, with a scrolling list requester. The default path can be either reference a file or a drawer. If a file, the filename gadget is filled in. Parameters:

``prompt'`

``help'`

tell the user what's going to happen.

``newpath'`

allows non-existent paths to be supplied as the default drawer.

``disk'`

show drive list first.

``default'`

default name of file to be selected. Note that this may be a relative pathname.

## 1.68 Installer.guide/f\_askstring

The ``askstring'` function

-----

`(askstring <parameters>)`

Prompts the user to enter a text string. Parameters:

```
`prompt`
`help`
    tell the user what's going to happen.

`default`
    the default text string.
```

## 1.69 Installer.guide/f\_asknumber

The `'asknumber'` function

```
-----
      (asknumber <parameters>)
```

Prompts the user to enter an integer quantity. Prints the allowed range below the integer gadget if the `'range'` parameter is given, and prevents the user from proceeding without entering a valid number. If no range is given, the allowed range is the non-negative numbers. However, if a default value is given which is negative, and no range is given, the allowed number range will be extended downward to include the given default value. ←

Parameters:

```
`prompt`
`help`
    tell the user what's going to happen.

`range`
    valid input range of numbers, the first must be less or equal to
    the second for proper operation.

`default`
    default value
```

## 1.70 Installer.guide/f\_askchoice

The `'askchoice'` function

```
-----
      (askchoice <parameters>)
```

Ask the user to select one out of N choices, using radio buttons. A bit mask is returned as a result, with the first bit indicating the state of the first choice, etc. Since this is returned in a bitmask, you can specify a maximum of 32 possible choices. Parameters:

Parameters:

```
`prompt`
`help`
```

---

tell the user what's going to happen.

``choices'`

a list of choice strings, such as ``"Apples"', `"Cherries"',` etc.

``default'`

the number of the default choice (defaults to 0)

NOTE: Starting with Installer V42.6, you can do magic things with the ``choices'`:

1. If you use an empty string as choice descriptor, the choice will be invisible to the user, i.e. it will not be displayed on screen. By using variables you can easily set up a programable number of choices then while retaining the bit numbering.
2. Previous versions of Installer did not support proportional fonts well and some people depended on the non proportional layout of the display for table like choices. So Installer will continue to render choices non proportional unless you start one of the choices with a special escape sequence ``"<ESC>[2p"'`. This escape sequence allows proportional rendering. It is wise to specify this only in the first choice of the list. Note this well. (V42)

## 1.71 Installer.guide/f\_askoptions

The ``askoptions'` function

-----

`(askoptions <parameters>)`

Ask the user to select any number of N choices, using checkbox buttons. A bit mask is returned as a result, with the first bit indicating the state of the first choice, etc. Since this is returned in a bitmask, you can specify a maximum of 32 possible choices.

Parameters:

``prompt'`

``help'`

tell the user what's going to happen.

``choices'`

a list of choice strings, such as ``"Apples"', `"Cherries"',` etc.

``default'`

a bit mask of the buttons to be checked (defaults to -1)

NOTE: Starting with Installer V42.6, you can do magic things with the ``choices'`:

1. If you use an empty string as choice descriptor, the choice will be invisible to the user, i.e. it will not be displayed on screen. By using variables you can easily set up a programable number of

choices then while retaining the bit numbering.

2. Previous versions of Installer did not support proportional fonts well and some people depended on the non proportional layout of the display for table like choices. So Installer will continue to render choices non proportional unless you start one of the choices with a special escape sequence `"<ESC>[2p"`. This escape sequence allows proportional rendering. It is wise to specify this only in the first choice of the list. Note this well. (V42)

## 1.72 Installer.guide/f\_askbool

The `'askbool'` function

-----

```
(askbool <parameters>)
```

Ask the user to select yes or no. Parameters:

```
'prompt'
```

```
'help'
```

tell the user what's going to happen.

```
'default'
```

0 = no, 1 = yes

```
'choices'
```

change the positive and negative text. The defaults are `"Yes"` and `"No"`. So to change the text to `"Proceed"` and `"Cancel"` you would use `'(choices "Proceed" "Cancel")'`

## 1.73 Installer.guide/f\_askdisk

The `'askdisk'` function

-----

```
(askdisk <parameters>)
```

Ask the user to insert a disk in a user friendly manner. For instance, the prompt can describe the disk by its label; e.g. `"FooBar Program Disk"`. This function will not exit until the correct disk is inserted, or the user aborts.

```
'prompt'
```

```
'help'
```

tell the user what's going to happen.

```
'dest'
```

the volume name of the disk to be inserted

``newname'`  
a name to assign to the disk for future reference. This assignment is done even in Dry Run mode - it is considered "safe".

``disk'`  
switch to get a drive list to be shown initially.

``assigns'`  
this indicates that logical assigns should satisfy the request as well.

Note: The volume name must be supplied without a colon; i.e. "ENV" not "ENV:".

## 1.74 Installer.guide/f\_exists

The ``exists'` function

-----  
`(exists <filename> (noreq))`

Returns 0 if `<filename>` does not exist, 1 if a file, and 2 if a directory. If ``noreq'` is specified, no requester is displayed if the path given is not on a mounted volume. In this case the result is 0.

## 1.75 Installer.guide/f\_earlier

The ``earlier'` function

-----  
`(earlier <file-1> <file-2>)`

Returns TRUE if `file-1` is earlier than `file-2`.

## 1.76 Installer.guide/f\_getsize

The ``getsize'` function

-----  
`(getsize <filename>)`

Returns the size of a file in bytes.

---



## 1.77 Installer.guide/f\_getdevice

The 'getdevice' function

```
-----  
  
(getdevice <path>)
```

returns the name of the device upon which <path> resides. For example, 'c:mount' as a path might return 'WB\_2.x'.

## 1.78 Installer.guide/f\_getdiskspace

The 'getdiskspace' function

```
-----  
  
(getdiskspace <pathname>)
```

Returns the available space in bytes on the disk given by pathname. Returns -1 if the pathname is bad or information could not be obtained from the filesystem even though pathname was valid.

## 1.79 Installer.guide/f\_getsum

The 'getsum' function

```
-----  
  
(getsum <filename>)
```

Returns the checksum of a file, for comparing versions. You can use this function to obtain the checksum when preparing the script and to check it while the script is running.

## 1.80 Installer.guide/f\_getversion

The 'getversion' function

```
-----  
  
(getversion <filename> (resident))
```

If the named file has a RomTag with an ID string or a OS 2.x version string, this will return the version number. If filename is not provided, then the version of the OS is returned instead. Note that this function does NOT assume files ending with '.library' or '.device' reside in a particular place - the path must be included. If 'resident' is specified, the function attempts to return version of library or device in memory. For example:

---

```
(getversion "intuition.library" (resident))
```

would return the version/revision of intuition. Note that using the 'resident' parameter causes first the library and then the device list to be checked.

The version number is returned as a 32 bit value, where the high order 16 bit word is the version and the low order word is the revision. Here are some sample statements to parse a version number:

```
(set vernum (getversion "c:iconx"))
(set ver (/ vernum 65536))
(set rev (- vernum (* ver 65536) ) )

(message
  ("You have version %ld.%ld" ver rev)
)
```

If the file in question cannot be located, a version of 0 will be returned.

## 1.81 Installer.guide/f\_getenv

The 'getenv' function  
-----

```
(getenv <name>)
```

Returns the contents of the given ENV: variable.

## 1.82 Installer.guide/f\_getassign

The 'getassign' function  
-----

```
(getassign <name> <opts>)
```

Returns the pathname of the object '<name>'. The default is for logical assignments only, but can be changed using an options string where the characters are:

```
`v'
  only match volumes
```

```
`a'
  only match logical assignments
```

```
`d'
  only match devices
```

Therefore 'a' would be equivalent to having no options. Returns an empty string on failure.

Notes:

- '<name>' must be supplied without a colon; i.e. '"ENV"' not '"ENV:"'. A variable previously set to name may be used in place of name.
- If a device name is used as the name and the search is limited to devices, then 'getassign' will return the device or volume name if the device exists, otherwise it will return an empty string. An example usage would be '(getassign "df1" "d")'.
- 'getassign' cannot handle non-binding assigns due to an AmigaDOS limitation. It will return an empty string for them, starting with Installer V42. The same thing will happen for deferred assigns that cannot be resolved by a simple access for some reason.

## 1.83 Installer.guide/f\_iconinfo

The 'iconinfo' function (V42.12)

-----  
(iconinfo <parameters>)

Obtain information about an icon's tool type and more. Except for the result, this function differs from other functions. The arguments for most parameters are not values but names of symbols that will be set to those values by the function. Be careful!

Parameters:

'prompt'

'help'

tell the user what's going to happen.

'dest'

the name of the icon to be modified. There is no need to specify a '.info' extension.

'gettooltype'

the tooltype name and result symbol name string.

'getdefaulttool'

symbol name for the default tool name of a project.

'getstack'

symbol name for the current stack size of the icon.

'getposition'

Two symbol names for the saved icon position in X and Y direction. Do not use this lightly. It is intended to keep icon

---

positions on updates with help of the new parameter `'setposition'` of the statement

```
'tooltype'
```

```
. Arbitrarily changing icon positions
```

will lead to annoyed users due to different Workbench and font setups. If the icon doesn't have a position set, -1 is returned for the respective position value. This may be passed to

```
'tooltype'
```

```
.
```

```
'confirm'
```

if this option is present, the user will be asked for confirmation, otherwise the modification proceeds silently.

```
'safe'
```

make changes even if in PRETEND mode

## 1.84 Installer.guide/f\_database

The `'database'` function

```
-----
```

```
(database <feature> [<checkvalue>])
```

Returns information about the Amiga that the Installer is running on. The argument `'<feature>'` must be a string. This function always returns a string result, even if the result looks like a number. If the feature requested is not recognized, the function returns `"unknown"`. The currently understood features and their possible values for `'<feature>'` are:

```
"vblank"
```

```
"50", "60"
```

```
"cpu"
```

```
"68000", "68010", "68020", "68030", "68040", "68060"
```

```
"fpu"
```

```
"NOFPU", "68881", "68882", "FPU40"
```

```
"graphics-mem"
```

returns a string representing the amount of free graphics memory in bytes

```
"total-mem"
```

returns a string representing the total amount of free memory in bytes

```
"chiprev"
```

```
'AA', 'ECS', 'AGNUS'.
```

If you specify the optional string `'<checkvalue>'`, a boolean result

will be returned instead of a string result, showing if the string result would match. This is an exact match provided as convenience operator. To check for a range of CPUs, you should do something like this:

```
(not (patmatch "68000|68010" (database "cpu")))
```

Always try to think defensive when using the 'database' function. Different configurations might return unexpected result combinations. Be prepared to handle them.

## 1.85 Installer.guide/f\_select

The 'select' function

```
(select <n> <item1> <item2> ...)
```

Returns the value of the Nth item.

## 1.86 Installer.guide/f\_patmatch

The 'patmatch' function

```
(patmatch <pattern> <string>)
```

Determines if a string matches an AmigaDOS pattern. Returns either TRUE or FALSE.

## 1.87 Installer.guide/f\_symbolval

The 'symbolval' function (V42.9)

```
(symbolval <symbolname>)
```

Evaluates the string expression '<symbolname>' and retrieves the value of the symbol named by the result. Together with the statement

```
'symbolset'
```

, this function can be used to set up a somewhat dynamic management of variable names.

NOTE: Please read the descriptions of

```
'set'
```

, too!

---

## 1.88 Installer.guide/f\_comparison

The Comparison Functions

---

```
(= <expression-1> <expression-2>)
(> <expression-1> <expression-2>)
(>= <expression-1> <expression-2>)
(< <expression-1> <expression-2>)
(<= <expression-1> <expression-2>)
(<> <expression-1> <expression-2>)
```

These are the standard relational expressions.

## 1.89 Installer.guide/f\_basicmath

The basic Math Functions

---

```
(+ <expression> ...)
```

Returns the sum of all the arguments.

```
(- <expression-1> <expression-2>)
```

Returns the first argument minus the second argument.

```
(* <expression> ...)
```

Returns the product of all the arguments.

```
(/ <expression-1> <expression-2>)
```

Returns the first argument divided by the second argument.

## 1.90 Installer.guide/f\_logical

The logical Functions

---

```
(AND <expression-1> <expression-2>)
(OR <expression-1> <expression-2>)
(XOR <expression-1> <expression-2>)
(NOT <expression>)
```

---

Standard logical functions.

## 1.91 Installer.guide/f\_bit

The bitwise logical Functions

-----

```
(BITAND <expression-1> <expression-2>)
(BITOR <expression-1> <expression-2>)
(BITXOR <expression-1> <expression-2>)
(BITNOT <expression>)
```

Bitwise versions of the standard logical functions.

## 1.92 Installer.guide/f\_bitshift

The bitshifting Functions

-----

```
(shiftright <number> <amount to shift>)
(shiftleft <number> <amount to shift>)
```

These functions perform a bit-oriented shift by the amount specified. Zeros are shifted in on the opposite side.

## 1.93 Installer.guide/f\_bittest

The 'IN' Function

-----

```
(IN <expression> <bit number-1> ...)
```

Returns 0 if none of the given bit numbers (starting at 0 for the LSB) is set in the result of expression, else returns a mask of the bits that were set.

## 1.94 Installer.guide/Summary of Parameters

Summary of Parameters

=====

'(all)'

In the 'copyfiles' statement, specifies that all files are to be

---

copied.

`'(append <string>).'`

Within a `'textfile'` statement, will append the string to the textfile.

`'(assigns).'`

An option used in the `'askdisk'` statement to indicate that logical assigns will match the askdisk request as well.

`'(choices <string-1> <string-2> ...).'`

Used to display a series of checkmarks or radio buttons depending on the command used. This is used in the `'askchoice'` and `'askoptions'` functions to indicate what choices the user has. It can also be used in the `'copyfiles'` statement to specify that only certain files can be copied. If absent, some other criterion will be used to determine which files to copy.

NOTE: Starting with Installer V42.6, you can do magic things with the `'choices'` when used with `'askchoice'` or `'askoptions'`:

1. If you use an empty string as choice descriptor, the choice will be invisible to the user, i.e. it will not be displayed on screen. By using variables you can easily set up a programable number of choices then while retaining the bit numbering.
2. Previous versions of Installer did not support proportional fonts well and some people depended on the non proportional layout of the display for table like choices. So Installer will continue to render choices non proportional unless you start one of the choices with a special escape sequence `'"<ESC>[2p"'`. This escape sequence allows proportional rendering.

`'(command <text> ...).'`

Specifies the text of a command to be inserted into the `'S:User-Startup'` file. Argument strings are merged.

`'(confirm <user-level>).'`

On some statements, the user will only be informed of the action and allowed to cancel it if the `'confirm'` option is specified. The user level can be `'expert'` or `'average'` `'expert'` is the default.

`'(default <value>).'`

Specifies the default value of an askchoice, askstring, or asknumber action.

`'(delopts <option> <option> ...).'`

Indicates to the `'copyfiles'`, `'copylib'` and `'delete'` statements that the listed options should be removed from the global internal list of options for this statement. The default global option is `"fail"`.

---



- ``(dest <filename>)``  
Specifies the file or directory to be modified as part of the command.
- ``(disk)``  
When used with the ``rename`` statement, specifies that a disk relabel operation is really desired. When used with the ``askdir`` or ``askfile`` statement, specifies that a drive list should be shown initially instead of a file list.
- ``(fonts)``  
Indicates to the ``copyfiles`` statement that accompanying ``.font`` files are to be copied as well.
- ``(help <string-1> <string-2> ...)``  
This is used to specify the help text for each action.
- ``(infos)``  
Indicates to the ``copyfiles`` statement that accompanying ``.info`` files are to be copied as well. If the destination drawer does not exist, a default icon will be made for the drawer the Installer creates. It is also valid for the ``delete`` statement.
- ``(include <filename>)``  
Within a ``textfile`` statement, will append the listed file to the textfile. If the file does not exist, the include will be ignored.
- ``(newname <name>)``  
Used in ``copyfiles`` to specify that a file will have a new name after being copied. Used in ``askdisk`` to assign the new name to the inserted disk. Used in ``copylib`` to specify that the library will have a new name after being copied.
- ``(newpath)``  
Used by ``askdir`` and ``askfile`` to allows non-existent paths to be supplied as the default drawer.
- ``(nogauge)``  
When used with the ``copyfiles`` and ``copylib`` statements this disables the copy status indicator.
- ``(noposition)``  
Used to modify the positioning of an icon to `'NO_ICON_POSITION'`.
- ``(pattern <string>)``  
Used in the ``copyfiles`` statement to specify a wildcard pattern.
- ``(prompt <string-1> <string-2> ...)``  
This is used to provide the "title" of the screen which explains to the user what this step does.
- ``(range <min> <max>)``  
Specifies the range of allowable numbers for an asknum statement.
- ``(safe)``  
This tells the Installer that an action not normally performed in Pretend mode should be performed.
-

`'(setdefaulttool <value>)'`  
Used to modify the default tool of an icon.

`'(setstack <value>)'`  
Used to modify the stack size included in an icon.

`'(settooltype <tooltype> <value>)'`  
Used to modify a tooltype to a certain value. If the tooltype does not exist it will be created; if the '`<value>`' parameter is omitted, the tooltype will be deleted. A tooltype without a value may be added in the following manner:

```
(settooltype <tooltype-string> "")
```

Remember that `'(tooltype <tooltype-string>)'` deletes the tooltype given.

`'(source <filename>)'`  
Specifies the file or directory to be read as part of this command.

`'(swapcolors)'`  
This is obsolete and should no longer be used. It is here for backwards compatibility and serves no function. DON'T USE IT!

`'(optional <option> <option> ...)'`  
Indicates to the `'copyfiles'` and `'copylib'` statements that it is not a fatal error to have a copy fail. Used for `'delete'` to indicate if deletion should be "forced".

`'(resident)'`  
Used for `'getversion'` to specify that the library and device list in memory should be searched.

## 1.95 Installer.guide/Pre-Defined Variables

### Pre-Defined Variables

=====

Pre-defined variables are available for use by the install script. They may be modified on-the-fly, but their type may not be changed (e.g. from strings to numeric) unless it never had a value to begin with.

`'@abort-button'`  
Replacement text for the 'Abort Install' button.

`'@app-name'`  
The 'APPNAME' value given at startup.

`'@icon'`  
The full pathname of the icon used to start the Installer not including any `'info'` suffix on a WB start. This is not

---

necessarily the script's icon! Starting with Installer V42.12, this variable will contain the full path to the script on a Shell start.

`'@execute-dir'`

If this variable is set to a valid path, then the Installer will change directory to it whenever a `'run'` or `'execute'` statement is performed.

`'@default-dest'`

The Installer's suggested location for installing an application. If you installed the application somewhere else (as the result of asking the user) then you should modify this value - this will allow the `"final"` statement to work properly. Note that creating a drawer and putting the application in that drawer is considered installing the application somewhere else. Set it to `''''` if there really is no definite place that the `"application"` was installed. The log file will be copied to the drawer indicated by `'@default-dest'` unless it was set to `''''`.

`'@language'`

Used to set the variable `'@language'`. The default value for `'@language'` is the name of the current locale or `"english"`. The use of this variable is left up to the install script.

`'@pretend'`

The state of the Pretend flag (1 if Pretend mode).

`'@user-level'`

The user-level the script is being run at: 0 for novice, 1 for average, 2 for expert.

`'@installer-version'`

An integer representing the current version of Installer. The value matches the one that `'getversion'` would return for Installer. This variable is available starting with Installer V42.

`'@error-msg'`

The text that would have been printed for a fatal error, but was overridden by a trap statement.

`'@special-msg'`

If a script wants to supply its own text for any fatal error at various points in the script, this variable should be set to that text. The original error text will be appended to the `special-msg` within parenthesis. Set this variable to `''''` to clear the `special-msg` handling.

`'@ioerr'`

The value of the last DOS error. Can be used in conjunction with the `'trap'` statement to learn more about what an error occurred. Starting with Installer V42, this variable is also set by the `'run'`, `'execute'`, and `'rexx'` statements.

`'@each-name'`

`'@each-type'`

Used in a `'foreach'` loop (see above).

---

```
`@askoptions-help'
`@askchoice-help'
`@asknumber-help'
`@askstring-help'
`@askdisk-help'
`@askfile-help'
`@askdir-help'
`@copylib-help'
`@copyfiles-help'
`@mkdir-help'
`@startup-help'
```

Default help text for various functions. These can be appended to the explanation provided for a particular action or used as is.

## 1.96 Installer.guide/Installer Language Quick Reference

Installer Language Quick Reference

\*\*\*\*\*

Overview

Quick Language Overview

Pre-Defined Variables

Default Help String Variables

Statements

Functions

## 1.97 Installer.guide/Overview (Quick)

Overview

=====

- Attempts to install in 'Work:' by default if it exists.
- HELP key brings up context-sensitive help. Esc key brings up the abort requester.
- Can add assigns to 'S:User-Startup', and adds lines to 'S:Startup-Sequence' if necessary to make sure 'S:User-Startup' is executed upon boot-up. Uses the boot volume for 'S:'.
- Can check versions of files and libraries.

- Install can run in "Real" (do it) or "Pretend" (dry run) modes.

## 1.98 Installer.guide/Quick Language Overview

### Quick Language Overview

=====

- Language is LISP-like (lots of parentheses ()) (-:).
- Variables are typeless (a la ARexx), i.e. strings and numbers are treated interchangeably.
- Strings are delimited with `'''` or `''`.
- Certain embedded sequences are available for strings:
  - `\n`  
newline character
  - `\r`  
return character
  - `\t`  
tab character
  - `\h`  
horizontal tab character (V42.6)
  - `\v`  
vertical tab character (V42.6)
  - `\b`  
backspace character (V42.6)
  - `\f`  
formfeed character (V42.6)
  - `\"`  
double quote
  - `'`  
single quote
  - `\\`  
backslash
  - `\ooo`  
some octal number `'ooo'` (V42.6)
  - `\xxx`  
some hex number `'XX'` (V42.6)
- Statements go in parentheses ( ). The general format is:  
`'(operator <operand1> <operand2> ...)'`

- E.g., to assign the value '5' to the variable 'x', use '(set x 5)'
- To produce the sum of two numbers, use '(+ 5 9)'
- Note that there is no difference between operators and functions—the function 'set' and the arithmetic operator '+' are both used exactly the same way.
- Combining statements: A statement can be used as the operand to another statement. E.g.:

```
(set x (+ 3 5))
```

In this case, the statement '(+ 3 5)' is evaluated first, and the result is 8. You can think of this as having the '(+ 3 5)' part being replaced by an 8, leaving:

```
(set v 8)
```

- Note that the '(+ 3 5)' part actually produced a value: '8'. This is called the "result" of the statement. Many statements return results, even some that might surprise you (such as 'set' and 'if').
- Comments are preceded with a semi-colon ';'.
- Hex numbers are preceded with a '\$' (e.g. '\$23').
- Binary numbers are preceded with a '%' (e.g. '%0101').
- Many statements return a value which can be used in assignments, tests, etc.
- Data can be formatted using a string literal with argument placemarkers, for example:

```
("I am %ld foot %ld inches tall." 6 3)
;Produces a string with %ld's replaced with 6 and 3.
;Remember that decimal values must be specified as longwords.
```

## 1.99 Installer.guide/Pre-Defined Variables (Quick)

Pre-Defined Variables

=====

```
'@default-dest'
  directory where install wants to put things by default.

'@each-name'
'@each-type'
  used in
      'foreach'
      .
```

``@error-msg'`  
message that would be displayed if error not trapped. See  
``trap'`  
.

``@execute-dir'`  
Installer will change to this directory before performing a  
statement  
``run'`  
, or  
``execute'`  
.

``@icon'`  
pathname of install start icon on WB start or (V42.12) install  
script on Shell start.

``@installer-version'`  
Installer program version in `'version'` format.

``@language'`  
language specified in tooltypes/Shell (locale name if available or  
`'"english"'`).

``@pretend'`  
state of `'pretend'` (dry run mode) flag: 0-Real, 1-Pretend.

``@special-msg'`  
custom fatal error message.

``@user-level'`  
0-Novice, 1-Average, 2-Expert.

## 1.100 Installer.guide/Default Help String Variables

### Default Help String Variables

=====

You can set these variables to override the default help texts for the appropriate commands.

- `@askoptions-help`
  - `@askchoice-help`
  - `@asknumber-help`
  - `@askstring-help`
  - `@askdisk-help`
  - `@askfile-help`
-

- @askdir-help
- @copylib-help
- @copyfiles-help
- @mkdir-help
- @startup-help

## 1.101 Installer.guide/Statements (Quick)

Statements

=====

Many commands have standard parameters (some optional):

- ``(all)'`  
specifies all files are to be handled
  - ``(append <string>)'`  
add string to text file (for textfile)
  - ``(choices <string1> <string2> ...)'`  
checkmark or radio button options
  - ``(command <string1> <string2>...)'`  
add to `'S:User-Startup'`
  - ``(confirm <user-level>)'`  
confirmation
  - ``(default <value>)'`  
default value, choice, string, etc.
  - ``(dest <file>)'`  
output to <file>
  - ``(help <string1> <string2> ...)'`  
define current help info
  - ``(include <file>)'`  
insert file in textfile statement
  - ``(infos)'`  
copy/delete ``.info'` files also
  - ``(newname <name>)'`  
specify new file or disk name
  - ``(noposition)'`  
make icon "floating"
  - ``(pattern <string>)'`
-



used w/ "files" for patterns

```
'(prompt <string1> <string2> ...)'
  text to show user
```

```
'(range <min> <max>)'
  numeric input ('asknum') range
```

```
'(safe)'
  force Installer to perform action even if in Pretend mode.
```

```
'(settooltype <tooltype> <value>)'
  set icon tool type
```

```
'(setdefaulttool <value>)'
  set icon's default tool
```

```
'(setstack <value>)'
  set icon's stack value
```

```
'(source <file>)'
  read from <file>
```

```
'(swapcolors)'
  swap first two planes of icon's image if OS rev less than v36
```

```
'(welcome <string> <string> ...)'
  invokes "welcome" display
```

Note: Custom parameters are shown below in < >, and standard parameters are shown as '(param..)' where 'param' is one of 'help', 'prompt', 'safe', etc. See above for details on standard parameters.

```
'(abort <string1> <string2> ...)'
  abandon installation
```

```
'(complete <num>)'
  display percentage through install in titlebar
```

```
'(copyfiles (prompt..) (help..) (source..) (dest..) (newname..) (choices..)
  '(all) (pattern..) (files) (infos) (confirm..) (safe) (optional
  <option> <option> ...) (delopts <option> <option> ...) (nogauge))'
```

copy files (and subdir's by default). files option say NO subdirectories

```
'(copylib (prompt..) (help..) (source..) (dest..) (newname..) (infos) (confirm)'
  '(safe) (optional <option> <option> ...) (delopts <option>
  <option> ...) (nogauge))'
```

install a library if newer version

```
'(delete file (help..) (prompt..) (confirm..) (infos) (optional <option> <option> ←
  ...) (all))'
  '(delopts <option> <option> ...) (safe))'
```

delete file

---

```
`(execute <arg> (help..) (prompt..) (confirm) (safe))'
    execute script file

`(exit <string> <string> ... (quiet))'
    end installation after displaying strings (if provided)

`(foreach <dir> <pattern> <statements>)'
    do for entries in directory

`(if expr truestatements falsestatements)'
    conditional

`(makeassign <assign> <path> (safe)) ; note: <assign> doesn't need `:'
    create an assignment

`(mkdir <name> (prompt..) (help..) (infos) (confirm..) (safe))'
    make a directory

`(message <string1> <string2>... (all))'
    display message with Proceed, Abort buttons

`(onerror (<statements>))'
    general error trap

`(protect <file> [<string of flags to change>] [<decimal mask>] <parameters>)'
    get/set file protection flags

`(rename <old> <new> (help..) (prompt..) (confirm..) (safe))'
    rename files

`(rexx <arg> (help..) (prompt..) (confirm..) (safe))'
    execute ARexx script

`(run <arg> (help..) (prompt..) (confirm..) (safe))'
    execute program

`(set <varname> <expression>)'
    assign a value to a variable

`(startup (prompt..) (command..))'
    add a command to the boot scripts (startup-sequence, user-startup)

`(symbolset <symbolname> <expression>)'
    assign a value to a variable named by the string result of
    `<symbolname>' (V42.9)

`(textfile (prompt..) (help..) (dest..) (append) (include..) (confirm..) (safe))'
    create text file from other text files and strings

`(tooltype (prompt..) (help..) (dest..) (settooltype..) (setstack..))'
    `(setdefaulttool..) (nposition) (confirm..) (safe))'

    modify an icon

`(trap <flags> <statements>)'
    trap errors. flags: 1-abort, 2-nomem, 3-error, 4-dos, 5-badargs
```

---

```
`(until <expr> <statements>)`  
do-until conditional structure (test end of loop)  
  
`(welcome <string> <string> ...)`  
allow Installation to commence.  
  
`(while <expr> <statements>)`  
do-while conditional structure (test top of loop)  
  
`(working)`  
indicate to user that Installer is busy doing things
```

## 1.102 Installer.guide/Functions (Quick)

### Functions

=====

```
`(= <expr1> <expr2>)`  
equality test (returns 0 or 1)  
  
`(> <expr1> <expr2>)`  
greater than test (returns 0 or 1)  
  
`(>= <expr1> <expr2>)`  
greater than or equal test (returns 0 or 1)  
  
`(< <expr1> <expr2>)`  
less than test (returns 0 or 1)  
  
`(<= <expr1> <expr2>)`  
less than or equal test  
  
`(+ <expr1> <expr2> ...)`  
returns sum of expressions  
  
`(- <expr1> <expr2>)`  
returns '<expr1>' minus '<expr2>'  
  
`(* <expr1> <expr2> ...)`  
returns product of expressions  
  
`(/ <expr1> <expr2>)`  
returns '<expr1>' divided by '<expr2>'  
  
`(AND <expr1> <expr2>)`  
returns logical 'AND' of '<expr1>' and '<expr2>'  
  
`(OR <expr1> <expr2>)`  
returns logical 'OR' of '<expr1>' and '<expr2>'  
  
`(XOR <expr1> <expr2>)`  
returns logical 'XOR' of '<expr1>' and '<expr2>'
```

---

---

```
`(NOT <expr>)`  
    returns logical 'NOT' of '<expr>'  
  
`(BITAND <expr1> <expr2>)`  
    returns bitwise 'AND' of '<expr1>' and '<expr2>'  
  
`(BITOR <expr1> <expr2>)`  
    returns bitwise 'OR' of '<expr1>' and '<expr2>'  
  
`(BITXOR <expr1> <expr2>)`  
    returns bitwise 'XOR' of '<expr1>' and '<expr2>'  
  
`(BITNOT <expr>)`  
    returns bitwise 'NOT' of '<expr>'  
  
`(shiftright <number> <amount to shift>)`  
    logical shift right  
  
`(shiftleft <number> <amount to shift>)`  
    logical shift left  
  
`(IN <expr> <bit-number> <bitnumber>...)`  
    returns '<expr>' 'AND' bits  
  
`(<format string> <arg1> <arg2> ...)`  
    printf clone  
  
`(askdir (prompt..) (help..) (default..) (newpath) (disk))`  
    ask for directory name  
  
`(askfile (prompt..) (help..) (default..) (newpath) (disk))`  
    ask for file name  
  
`(askstring (prompt..) (help..) (default..))`  
    ask for a string  
  
`(asknumber (prompt..) (help..) (range..) (default..))`  
    ask for a number  
  
`(askchoice (prompt..) (choices..) (default..))`  
    choose 1 options  
  
`(askoptions (prompt (help..) (choices..) default..))`  
    choose n options  
  
`(askbool (prompt..) (help..) (default..) (choices..))`  
    0=no, 1=yes  
  
`(askdisk (prompt..) (help..) (dest..) (newname..) (assigns))`  
`(cat <string1> <string2>...)`  
    returns concatenation of strings  
  
`(exists <filename> (noreq))`  
    0 if no, 1 if file, 2 if dir  
  
`(expandpath <path>)`  
    Expands a short path to its full path equivalent
```

---

---

```
`(earlier <file1> <file2>)`  
    true if file1 earlier than file2  
  
`(fileonly <path>)`  
    return file part of path (see pathonly)  
  
`(getassign <name> <opts>)`  
    return value of logical name (no `:`) `  
    <opts>`: `v`-volumes,  
    `a`-assigns, `d`-devices  
  
`(getdevice <path>)`  
    returns name of device upon which <path> resides  
  
`(getdiskspace <path>)`  
    return available space  
  
`(getenv <name>)`  
    return value of environment variable  
  
`(getsize <file>)`  
    return size  
  
`(getsum <file>)`  
    return checksum of file for comparison purposes  
  
`(getversion <file> (resident))`  
    return version/revision of file, library, etc. as 32 bit num  
  
`(iconinfo <parameters>)`  
    return information about an icon (V42.12)  
  
`(pathonly <path>)`  
    return dir part of path (see fileonly)  
  
`(patmatch <pattern> <string>)`  
    Does <pattern> match <string> ? TRUE : FALSE  
  
`(select <n> <item1> <item2> ...)`  
    return n'th item  
  
`(strlen <string>)`  
    string length  
  
`(substr <string> <start> [<count>])`  
    returns a substring of <string>  
  
`(symbolval <symbolname>)`  
    returns the value of the symbol named by the string expression  
    `  
    <smbolval>` (V42.9)  
  
`(transcript <string1> <string2>)`  
    puts concatenated strings in log file  
  
`(tackon <path> <file>)`  
    return properly concatenated file to path
```

---