Installer

COLLABORATORS			
	TITLE : Installer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Installer

1.1 Installer.guide

```
                Documentation for 1.24 Installer    (Last Revised: January 12th ↩
                   , 1993)

     (C) Copyright 1991-93 Commodore-Amiga,Inc.  All Rights Reserved


               Adaptation to Guide format by Gérard Cornu (4-Oct-94)
                                               Summary:

            1
              Background
                            2
              Overview
                            3
              Scripting Language Tutorial
                            4
              Installer Language Reference
                            5
              Installer Language Quick Reference

                             INDEX
```

1.2 The version of this Guide is...

```
Please note that this is the Guide version string, not the doc version upon
which this Guide is based (Installer.doc 1.24):

"$VER: Installer.guide 1.22 (16.10.94) Guide version of Installer.doc 1.24
       by Gérard Cornu, original doc (C) Copyright 1991-93 Commodore-Amiga,Inc.
       All Rights Resrved."
```

1.3 Section 1: Background

Section 1: 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.

About a year ago ([from January 12th, 1993]), 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.

e. Not everyone will be running 2.0 by the time the tool becomes available, so it must run under 1.3 and 2.0.

1.4 Section 2: Overview

Section 2: 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.

2.1
Standard Invocation
2.2
Initial Actions
2.3
Startup Screens
2.4
Installation Actions

1.5 2.1 Standard Invocation

2.1 Standard Invocation

The Installer program requires a 10000 byte stack. Project icons for
Installer script should indicate a stack size of 10000. If starting
Installer from a CLI, first do a "Stack 10000".

The Installer is normally started up 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:

APPNAME

LOG

NOPRINT

DEFUSER

LOGFILE

PRETEND

LANGUAGE

MINUSER

SCRIPT
Although the installer can be started up from the CLI, that is not ↩
the
recommended mode. CLI invocation is provided mainly for script debugging
purposes. The command template is:

```
 SCRIPT
/K,
 APPNAME
/K,
 MINUSER
/K,
 DEFUSER
/K,
 LOGFILE
/K,

 NOLOG
/S,
 NOPRETEND
/S,
 NOPRINT
/S,
 LANGUAGE
/K
```

1.6 SCRIPT

```
SCRIPT   - Path to a script file to be used with Installer.
```

1.7 APPNAME

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

1.8 MINUSER

```
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.
```

1.9 DEFUSER

```
     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 is not defined).
```

1.10 NOPRINT

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

1.11 PRETEND

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

1.12 NOPRETEND

```
     NOPRETEND  - indicates that
   PRETEND mode
   is not available for this script.
```

1.13 LANGUAGE

```
     LANGUAGE - Used to set the variable
   @language
   (default for @language is
"english".
```

1.14 LOGFILE

```
LOGFILE  - The name of the log file that the Installer should use.  This
           must be a full path.  The default is "install_log_file".
```

1.15 LOG

```
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.
```

1.16 NOLOG

```
NOLOG    - indicates that the creation of a log file in NOVICE mode
           is disabled.
```

1.17 2.2 Initial Actions

```
   2.2  Initial Actions
```

The first thing the installer does is 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.

1.18 2.3 Startup Screens

```
   2.3  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 to either a file or printer.

1.19 2.4 Installation Actions

```
   2.4  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.

1.20 Section 3: Scripting Language Tutorial

```
                  Section 3: 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.

```
            3.1
                 Basic Elements
                           3.2
                 Escape Characters
                           3.3
                 Symbols (Variables)
                           3.4
                 Types of Symbols
                           3.5
                 Statements
                           3.6
                 Data Types
```

1.21 3.1 Basic Elements

3.1 Basic Elements

The basic elements of the installer language are:

Type	Example
decimal integers	5
hexadecimal integers	$a000
binary integers	%0010010
strings	"Hello" or 'Hello'
symbols	x
comments	; this is a comment
( )	for statement definition
space (or any white space)	delimits symbols

1.22 3.2 Escape Characters

3.2 Escape Characters

Escape characters are supported as in the C language:

Escape sequence	Produces
'\n'	newline character
'\r'	return character
'\t'	tab character
'\0'	a NUL character
'\"'	a double-quote
'\'	a backslash

1.23 3.3 Symbols (Variables)

3.3 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**
@#__#@
```

1.24 3.4 Types of Symbols

```
                  3.4  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.
```

1.25 3.5 Statements

```
                  3.5
                 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
               ").
```

1.26 3.6 Data Types

```
                  3.6  Data Types

All data types in the installer are dynamic, that is to say the type of a
variable is determined by the data that is in it.  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 "printf"-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 longwords). Note that a variable containing the 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 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.
   )                                         ; closing parenthesis

Note the style of the indenting. This makes for an easier to read program.

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
   )                                ; closing parenthesis

Finally, just to make things interesting, we'll make the "else" clause a
compound statement.

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

1.27 3.7 Special Features

3.7 Special Features

When the Installer first starts up, 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 Commodore.

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". 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 boot volume of the system when looking for the "startup-sequence" and 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 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 1990 DevCon notes. The Installer can also
checksum
files. A separate utility named "instsum" is provided to determine a file's checksum for use with this feature.

1.28 3.8 Miscellaneous

```
                  3.8  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 ...)
       )
   )

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

To run an external CLI 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")
```

1.29 Section 4: Installer Language Reference

```
                  Section 4: Installer Language Reference

         4.1
                 Notes
                         4.2
                 Statements
                         4.3
                 Control Statements
                         4.4
                 Debugging Statements
                         4.5
                 User-Defined Procedures
                         4.6
                 Functions
                         4.7
```

Summary of Parameters
4.8
Pre-Defined Variables

1.30 4.1 Notes

4.1 NOTES

a. 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
").

b. 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".

c. 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 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 caused by too many small strings being concatenated and merge them into larger blocks.

d. The "
run
" and "
execute
" statements only return the result of the command run or executed under 2.0; they always return 0 under 1.3. If you must have some result under both 1.3 and 2.0, try this combo:

```
# in the DOS script to execute:
failat 31
command
if error
    setenv installer-result 10
```

```
   else
       if warn
           setenv installer-result 5
       else
           setenv installer-result 0
       endif
   endif

   # in the installer script
   (
                execute
                DOS-Script)
   (
                set
                theResult (
                getenv
                "installer-result"))
```

e. Filename and directoryname wildcard patterns specified in a script must be no longer than 64 characters.

1.31 4.2 Statements

4.2 Statements

abort

protect

complete

rename

copyfiles

rexx

copylib

run

delete

set

execute

startup

exit

textfile

```
                makeassign

                tooltype

                makedir

                welcome

                message

                working
```

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

```
               (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.

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

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

Use the "
                cat
               " statement to do the reverse.
```

1.33 (makedir <name> <parameters>)

```
               (makedir <name> <parameters>)

Creates a new directory.  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, user will be prompted, else the
directory will be created silently.


safe
 - make directory even if in
PRETEND mode
.
```

1.34 (copyfiles <parameters>)

```
(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
 - tell the user what's going to happen.


help
 - text of help message.


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.


infos
 - switch to copy icons along with other files/directories.


fonts
 - switch to not display ".font" files, yet still copy any that
match a directory that is being copied.

(
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
.
```

1.35 (copylib <parameters>)

```
                (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.  "Copylib" 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
                 - tell the user what's going to happen.


                help
                 - text of help message.


                source
                 - name of source directory or file


                dest
                 - name of destination directory
                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


                infos
                 - switch to copy icons along with other files

   (
                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
                 - user will be asked to confirm.  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
               .
```

1.36 (startup <appname> <parameters>)

```
               (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 EXPERT mode. Parameters:

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


                help
                 - text of help message.


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

1.37 (tooltype <parameters>)

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

Modify an icon's tool type. 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.  Parameters:


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


                help
                 - text of help message.


                dest
                 - the icon to be modified.


                settooltype
                 - the tooltype name and value string.


                setdefaulttool
                 - default tool for a project.


                setstack
                 - set size of stack.


                noposition
                 - reset to NOICONPOSITION.


                swapcolors
                 - swap first two planes of icon's image if OS rev less
                       than v36.


                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
               .
```

1.38 (textfile <parameters>)

```
                (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
                 - tell the user what's going to happen.


                help
                 - text of help message.


                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.


                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
               .
```

1.39 (execute <argument> ...)

```
                (execute <argument> ...)

Executes an AmigaDOS script with the arguments given.  NOTE: Does not ask
user for confirmation, however this can be added by using "
                 askchoice
                " or
"
                 askbool
                ". Parameters:


                 prompt
```

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


                help
                 - text of help message.


                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
               .

Returns a result if executed under 2.0.  Returns 0 under 1.3.  See
                NOTES
                for
workarounds under 1.3.
```

1.40 (run <argument> ...)

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

Executes a compiled program with the arguments given.  NOTE: Does not ask
user for confirmation, however this can be added by using "
                askchoice
               " or
"
                askbool
               ". Parameters:


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


                help
                 - text of help message.


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


                safe
                 - run program even if in
                PRETEND mode
               .

Returns a result if executed under 2.0.  Returns 0 under 1.3.  See
```

```
                NOTES
                for
workarounds under 1.3.
```

1.41 (rexx <argument> ...)

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

Executes an ARexx script with the arguments given.  NOTE: Does not ask user
for confirmation, however this can be added by using "
                askchoice
               " or
"
                askbool
               ". If the ARexx server is not active, an error will be generated.
Parameters:


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


                help
                 - text of help message.


                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
               .
```

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

```
               (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:".
```

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

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


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


                 help
                  - text of help message.


                 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
                .
```

1.44 (delete <file> <parameters>)

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

Delete a file.  Note that a write protected file is considered "delete
protected" as well.  Parameters:


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


                 help
                  - text of help message.
```

```
                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)
              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
               .
```

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

```
               (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. "+r -w +e -d") 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.

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
                .
```

1.46 (abort <message> <message> ...)

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

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

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

```
                (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.
```

1.48 (complete <number>)

```
(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.
```

1.49 (message <string> <string> ...)

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

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). Parameters:

```
help
 - optional help text
```

1.50 (working <string> <string> ...)

```
(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
).
```

1.51 (welcome <string> <string> ...)

```
(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.

1.52 Control Statements

4.3 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.

foreach

if

onerror

select

trap

until

while

((...))

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

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

Operates as a standard "if-then" statement.

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

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

Operates as a standard "do-while" statement.

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

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

Operates as a standard "do-until" statement.

1.56 (foreach <drawer name> <pattern> <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.) Patterns specified in a script must be no
longer than 64 characters.
```

1.57 ((...) (...) (...))

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

Execute a sequence of statements.  The statements in the parentheses will be
executed in order -- not needed at topmost level.
```

1.58 (trap <trapflags> <statements>)

```
                (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
                )
    5 - bad parameter data
```

1.59 (onerror <statements>)

```
(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.
```

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

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

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

1.61 4.4 Debugging Statements

```
   4.4  Debugging Statements


debug

user
```

1.62 (user <user-level>)

```
(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
.

1.63 (debug <anything> <anything> ...)

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

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

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

will print "This value of myvar is 2". If the parameter is an uninitialized variable, then debug will print "<NIL>" as its value.

1.64 4.5 User-Defined Procedures

```
   4.5  User-Defined Procedures
```

The Installer has user-defined procedures (subroutines). This functionality is currently very primitive. There are no local variables. To define a new

```
procedure, use the "procedure" command:

   (procedure <procedure-name> <statement>)

You can then call the procedure like so:

   (<procedure-name>)

Note that <procedure-name> is not a string, just a symbolic name.
```

1.65 4.6 Functions

```
                  4.6  Functions

                 +

                 BITNOT

                 NOT

                 -

                 BITXOR

                 OR

                 *

                 cat

                 pathonly

                 /

                 database

                 patmatch

                 AND

                 earlier

                 select

                 askbool

                 exists

                 shiftleft

                 askchoice

                 fileonly
```

shiftrght

askdir

getassign

string

askdisk

getdevice

strlen

askfile

getdiskspace

substr

asknumber

getenv

tackon

askoptions

getsize

transcript

askstring

getsum

XOR

BITAND

getversion

= > >= < <= <>

BITOR

IN

1.66 (<string> <arguments> ...)

(<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 RawDoFmt. Note: This function does no argument type checking.

1.67 (cat <string> <string> ...)

```
(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 the "

```
 set
```

" statement to convert a string to an integer.

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

```
(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.

1.69 (strlen <string>)

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

Returns the length of the given string.

1.70 (transcript <string> <string> ...)

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

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

1.71 (tackon <path> <file>)

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

Concatenates the filename to the pathname and returns resulting string. Currently, "tackon" cannot deal with a leading '/' in the <file> parameter. This may be fixed in a future version.

1.72 (fileonly <path>)

```
(fileonly <path>)

Returns only the file part of a pathname.
```

1.73 (pathonly <path>)

```
(pathonly <path>)

Returns only the non-file part of a pathname.
```

1.74 (expandpath <path>)

```
(expandpath <path>)

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

1.75 (askdir <parameters>)

```
                (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 the
"
                 askdisk
                " command. Parameters:


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


                 help
                  - text of help message.


                 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
 - debugging parameter; indicates that logical assigns should
satisfy requests as well.  Remove this parameter before
distributing disk.
```

1.76 (askfile <parameters>)

```
(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
 - tell the user what's going to happen.


help
 - text of help message.


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.
```

1.77 (askstring <parameters>)

```
(askstring <parameters>)
```

Prompts the user to enter a text string. Parameters:

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

```
help
 - text of help message.


default
 - the default text string.
```

1.78 (asknumber <parameters>)

```
(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.  Parameters:


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


help
 - text of help message.


range
 - valid input range of numbers.


default
 - default value
```

1.79 (askchoice <parameters>)

```
(askchoice <parameters>)

Ask the user to select one out of N choices, using radio buttons.
Parameters:


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


help
 - text of help message.
```

```
choices
 - a list of choice strings, such as "ok" "cancel", etc.


default
 - the number of the default choice (defaults to 0)
```

1.80 (askoptions <parameters>)

```
(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.  Parameters:


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


help
 - text of help message.


choices
 - a list of choice strings, such as "ok" "cancel", etc.


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

1.81 (askbool <parameters>)

```
(askbool <parameters>)

Ask the user to select yes or no.  Parameters:


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


help
 - text of help message.


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")
```

1.82 (askdisk <parameters>)

```
(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
 - tell the user what's going to happen.

help
 - text of help message.

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
 - Debugging option; this indicates that logical assigns should
satisfy the request as well.  Remove this parameter before
distributing disk.
```

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

1.83 (exists <filename> (noreq))

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

Returns 0 if does not exists, 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.

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

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

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

1.85 (getsize <filename>)

(getsize <filename>)

Returns the size of a file.

1.86 (getdevice <path>)

(getdevice <path>)

returns the name of the device upon which <path> resides. For example,
"c:mount" as a path might return "WB_2.x".

1.87 (getdiskspace <pathname>)

(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).

1.88 (getsum

(getsum <filename>)

Returns the checksum of a file, for comparing versions.

1.89 (getversion <filename> (resident))

(getversion <filename> (resident))

If the named file has a RomTag with an ID string or a 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, 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 is 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 "
                1.0
               " ver rev)
)
```

1.90 (getenv <name>)

```
(getenv <name>)

Returns the contents of the given ENV: variable.
```

1.91 (getassign <name> <opts>)

```
(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").
```

1.92 (database <feature>)

```
(database <feature>)

Returns information about the Amiga that the Installer is running on.
"Feature" is 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 are:

    "vblank"         : "50", "60"

    "cpu"            : "68000", "68010", "68020", "68030", "68040"

    "graphics-mem"   : [returns a string representing the amount of free
                       graphics memory]

    "total-mem"      : [returns a string representing the total amount of
                       free memory]
```

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

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

Returns the value of the Nth item.
```

1.94 (patmatch <pattern> <string>)

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

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

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

```
(=  <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.
```

1.96 (+ <expression> ...)

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

Returns the sum of all the arguments.

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

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

Returns the first argument minus the second argument.

1.98 (* <expression> ...)

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

Returns the product of all the arguments.

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

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

Returns the first argument divided by the second argument.

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

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

Standard logical functions.

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

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

Bitwise versions of the standard logical functions.

1.102 (shiftleft, shiftrght <number> <amount to shift>)

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

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

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

```
(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.
```

1.104 4.7 Summary of Parameters

```
                  4.7  Summary of Parameters


                all

                newname

                append

                newpath

                assigns

                nogauge

                choices

                noposition

                command

                optional

                confirm

                pattern

                default

                prompt

                delopts

                range
```

```
                dest

                safe

                disk

                setdefaulttool

                files

                setstack

                fonts

                settooltype

                help

                source

                include

                swapcolors

                infos
```

1.105 (assigns)

```
               (assigns)

A debug option used in the "
                askdisk
               " statement to indicate that logical
assigns will match the askdisk request as well. This parameter should not
be used for final disks, only for debugging.
```

1.106 (help <string-1> <string-2> ...)

```
(help <string-1> <string-2> ...)

This is used to specify the help text for each action.
```

1.107 (prompt <string-1> <string-2> ...)

```
(prompt <string-1> <string-2> ...)

This is used to provide the "title" of the screen which explains to the user
what this step does.
```

1.108 (safe)

```
                (safe)

This tells the installer that an action not normally performed
in
                 PRETEND mode
                 should be performed.
```

1.109 (choices <string-1> <string-2> ...)

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

Used to display a series of checkmarks.  This is used in the "
                 askchoice
                "
function 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).
```

1.110 (pattern <string>)

```
                (pattern <string>)

Used in the "
                 copyfiles
                " statement to specify a wildcard pattern.
```

1.111 (all)

```
                (all)

In the "
                 copyfiles
                " statement, specifies that all files are to be copied.
```

1.112 (source <filename>)

```
(source <filename>)

Specifies the file or directory to be read as part of this command.
```

1.113 (dest <filename>)

```
(dest <filename>)

Specifies the file or directory to be modified as part of the command.
```

1.114 (newname <name>)

```
                (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.
```

1.115 (newpath)

```
                (newpath)

Used by "
                 askdir
                " and "
                 askfile
                " to allows non-existent paths to be supplied
as the default drawer.
```

1.116 (confirm <user-level>)

```
                (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).
```

1.117 files

```
(files)

Only copy files. By default the installer will match and copy subdirectories.
```

1.118 (infos)

```
                (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.
```

1.119 (fonts)

```
                (fonts)

Indicates to the "
                 copyfiles
                " statement that accompanying ".font" files are
to be copied as well.
```

1.120 (optional <option> <option> ...)

```
                (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".
```

1.121 (delopts <option> <option> ...)

```
                (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".
```

1.122 (nogauge)

```
                (nogauge)

When used with the "
                 copyfiles
                " and "
                 copylib
                " statements this disables the
copy status indicator.
```

1.123 (settooltype <tooltype> <value>)

```
(settooltype <tooltype> <value>)

Used to modify a tooltype to a certain value. If the tooltype does not exist
it will be created; if the <values> 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.
```

1.124 (setdefaulttool <value>)

```
(setdefaulttool <value>)

Used to modify the default tool of an icon.
```

1.125 (setstack <value>)

```
(setstack <value>)

Used to modify the stack size included in an icon.
```

1.126 (noposition)

```
(noposition)

Used to modify the positioning of an icon to NO_ICON_POSITION.
```

1.127 (swapcolors)

```
(swapcolors)

Used to swap the first two planes of the image of the icon being modified if
the version of the OS if less than 36 (i.e., prior to version 2.0).  This
does mean that your icons need to have the 2.0 color scheme on your
distribution disks.
```

1.128 (disk)

```
                (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).
```

1.129 (append <string>)

```
                (append <string>)

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

1.130 (include <filename>)

```
                (include <filename>)

Within a "
                 textfile
                " statement, will append the listed file to the textfile.
```

1.131 (default <value>)

```
                (default <value>)

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

1.132 (range <min> <max>)

```
                (range <min> <max>)

Specifies the range of allowable numbers for an
                 asknumber
                 statement.
```

1.133 (command <text> ...)

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

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

1.134 4.9 Pre-Defined Variables

```
                4.9  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

                 @app-name

                 @default-dest

                 @each-name

                 @each-type

                 @error-msg
```

@execute-dir

@icon

@ioerr

@language

@pretend

@special-msg

@user-level

@*-help

1.135 @abort-button

@abort-button

Replacement text for the "Abort Install" button.

1.136 @app-name

@app-name

The
APPNAME
value given as tooltype or CLI option at startup.

1.137 @icon

@icon

The pathname of the icon used to start the installer.

1.138 @execute-dir

@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.
```

1.139 @default-dest

```
@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 "".
```

1.140 @language

```
                @language

   Language specified in
               LANGUAGE
               tooltype or CLI option, default for
   @language is "english". The use of this variable is left up to the
   install script.
```

1.141 @pretend

```
                @pretend

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

1.142 Pretend Mode

```
                Installer can run in

      "Real" mode, (do it),
               @pretend
               is set to 0
   or
      "Pretend" mode, (dry run),
               @pretend
               is then set to 1.
```

1.143 @user-level

```
@user-level

The user-level the script is being run at:

      0 for novice,
      1 for average,
      2 for expert.
```

1.144 @error-msg

```
                @error-msg

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

1.145 @special-msg

```
@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.
```

1.146 @ioerr

```
                @ioerr

The value of the last DOS error.  Can be used in conjunction with the
"
              trap
             " statement to learn more about why an error occurred.
```

1.147 @each-name, @each-type

```
                @each-name

@each-type

Used in a "
              foreach
```

```
" loop.
```

1.148 Default help text for various functions:

```
                @askchoice-help
 @askdir-help
 @askdisk-help
 @askfile-help
 @asknumber-help
 @askoptions-help
 @askstring-help
 @copyfiles-help
 @copylib-help
 @makedir-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.
```

1.149 Section 5: Installer Language Quick Reference

```
               Section 5: Installer Language Quick Reference

          5.1
             Overview
                          5.2
             Quick Language Overview
                          5.3
             Pre-Defined Variables
                          5.4
             Default Help String Variables
                          5.5
             Statements
                          5.6
             Functions
```

1.150 5.1 Overview

```
               5.1  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.

Can
              check versions
              of files/libraries.

Installer can run in
              'Real' (do it) or 'Pretend' (dry run) modes
             .
```

1.151 5.2 Quick Language Overview

```
                5.2  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        '\r'    return
     '\t'    tab            '\0'    NULL
     '\"'    double-quote   '\'    backslash


              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 1 foot 0 inches tall." 6 3)
   ;Produces a string with 1's replaced with 6 and 3.
   ;Remember that decimal values must be specified as longwords.
```

1.152 5.3 Pre-Defined Variables

```
                5.3  Pre-Defined Variables

 @abort-button  -  Replacement text for the "Abort Install" button.
 @app-name      -  The
             APPNAME
             being given at startup.

              @default-dest
              -  Directory where install wants to put things by default.
 @each-name     -  used in "
              foreach
             " loop.
 @each-type     -  used in "
              foreach
             " loop.

              @error-msg
              -  Message that would be displayed if error not trapped (see
              trap
             ).

              @execute-dir
              -  If set to a valid path, installer will change directory to
```

```
                    it whenever a "
                 run
                " or "
                 execute
                " statement is performed.
    @icon          -  The pathname of the install script icon.

                 @ioerr
                 -  The value of the last DOS error.

                 @language
                 -  Language specified in tooltypes/CLI (default "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.

                 @*-help
                 -  Default help text for various
                 functions
                .
```

1.153 5.5 Statements

```
                  5.5  Statements

Many commands have standard parameters (some optional):

   (
                 all
                )                              ; specifies all files are to copied

   (
                 append
                 <string>)                    ; add string to text file
                                             (for
                 textfile
                )

   (
                 choices
                 <string1> <string2> ...)  ; 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 .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 (
                asknumber
               ) 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 show 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 " link "safe"}) (
                 optional
                 <option> <option> ...)
   (
                 delopts
                 <option> <option> ...) (
                 nogauge
                ))
                                ; install a library if newer version

(
                 delete
                 file (
                 help
                ..) (
                 prompt
                ..) (
                 confirm
                ..)
```

```
   (
                optional
                <option> <option> ...) (
                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

(
                makedir
                <name> (
                prompt
               ..) (
                help
               ..) (
                infos
               ) (
```

```
                confirm
               ..) (
                safe
               ))
                                    ; make a directory

(
                message
                <string1> <string2>...)          ; 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)

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

(
                tooltype
                (
                prompt
               ..) (
                help
               ..) (
                dest
               ..) (
                settooltype
               ..)
   (
                setstack
               ..) (
                setdefaulttool
               ..) (
                noposition
               ) (
                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
```

1.154 5.6 Functions

```
                  5.6  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>

(
                shiftleft
```

```
                <number> <amount to shift>) ; logical shift left

(
                shiftrght
                <number> <amount to shift>) ; logical shift right

(
                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
               ))
                                    ; Ask the user to insert a disk

(
                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' = logical,
                                          '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

(
                 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>

(
                 transcript
                 <string1> <string2>)    ; puts concatenated strings in log file

(
                 tackon
                 <path> <file>)              ; return properly concatenated
                                           file to path
```

1.155 Adaptation

```
                  Note from Gérard Cornu, the 'Guide adapter' ;-) :
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   The original 'Installer.doc' upon which
                this Guide
                is based, is
   (C) Copyright 1991-93 Commodore-Amiga,Inc.  All Rights Reserved.

   I have 'only' adapted it to the AmigaGuide format (as supplied with WB 2.1).

   I have used 'BadLinks' 1.17 by Roger Nedel, to check for bad links
   (he didn't find any ;-) and 'AGIndex' 1.04 by Camiel Rouweler, to generate
   the index, which I have 'touched up' by hand.

   If you find any bugs like bad links, or any other  mistakes pertaining
   only to the adaptation to the Guide format, don't hesitate to send them
   to
                me
               . Thanks.
```

Gérard Cornu

1.156 The doc to guide adapter's address... ;-)

Send a postcard with your appreciation and ideas to:

Gérard Cornu
11 Avenue Edouard Aynard
69130 Écully (France)

Or an email to:

gerard@ariane.univ-lyon2.fr

Important

1.157 Shameless plug...

I can translate commercial software's documentation or hardware manuals, in fact anything technical, from English (or American) to French and vice versa.

I have a very good knowledge and experience of the Amiga and computing in general.

Please click
here
for contact address.

1.158 Index

Index:
~~~~~

((...))

*

+

_
~~~~~

```
/
<
<=
=
>
>=
@*-help
@abort-button
@app-name
@default-dest
@each-name
@each-type
@error-msg
@execute-dir
@icon
@ioerr
@language
@pretend
@special-msg
@user-level
abort
Adaptation
all
AND
append
APPNAME
askbool
askchoice
askdir
```

askdisk

askfile

asknumber

askoptions

askstring

assigns

Background

Basic Elements

BITAND

BITNOT

BITOR

BITXOR

cat

check versions

checksum

choices

command

complete

confirm

Control
Statements

copy files

copyfiles

copylib

Data Types

database

debug

Debugging
Statements

default

Default Help
String Variables

delete

delopts

dest

disk

earlier

embedded
sequences

Escape
Characters

execute

exists

exit

expandpath

fileonly

files

fonts

foreach

format string

fuelgauge

Functions

getassign

getdevice

getdiskspace

getenv

getsize

getsum

getversion

pathonly

patmatch

pattern

Pre-Defined
Variables

PRETEND mode

prompt

protect

Quick
Language Overview

range

rename

resident

rexx

run

safe

Scripting
Language Tutorial

select

set

setdefaulttool

setstack

settooltype

shiftleft

shiftrght

source

Special
Features

Standard
Invocation

startup