# Installer

Jens Tröger

Copyright © 1999/2000 by Jens Tröger

**COLLABORATORS**

| | *TITLE* :<br><br>Installer | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | Jens Tröger | August 24, 2022 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# Installer

## 1.1 The Installer Language Programming Guide

```
                The Installer Language Programming Guide

© 1999/2000 by Jens Tröger
for copyright information and licence rules, please read the InstallerNG
guide and the .LICENCE file (both included in this distribution)

Preliminaries

               Introduction
                What's this thing about

               Author
                The author of this guide and the InstallerNG

               C= Installer vs. InstallerNG
                Old contra new

               Versions of Installer
                Keep this in mind!

               Installation
                Installing the Installer

First Steps

               Hello World
                Our first working programm

               The language
                Introduction to the language

               Starting the Installer
                How to start this tool

Programming secrets

               Symbols
                The bricks of a script
```

## 1.2 Introduction

Today, the installation of software products can be a very complex procedure. This is caused by the sometimes large set of different files or a very lowlevel intervention into the systems resources. Especially for novice users this process can be difficult and the system may be destroyed (worst case...).

This was the motivation for Commodore to build a tool, which covers the installation process and offers an easy to use and graphical interface. This thing was called the Installer. The user just tells where and what to install and the Installer cares for the installation process itself, i.e. the Installer checks for the versions, copies the files to the correct destinations, sets up a correct environment for the installed tool and so on. The user can choose, whether he wants an easy installation (means, as less as possible queries) or if he wants to get notified for every action.

If you are a programmer and you want to provide your user such an easy installation, you must write an Installer-Script. This script is simply a textfile and contains a programm, written in a special language. This language is simple and offers very much functions for querying the user, for setting and getting system properties, for file handling, string manipulation and, last but not least, a lot of mathematical functions. Furthermore, put an icon into your archive, which contains special tooltypes (for tooltypes please refer to your workbench guide) for the Installer. When the user double-clicks this icon, the Installer will get started and then looks for your script to execute it.

This guide will introduce you the Installer usage itself and the script language. You should have programmed and, additionally, should have some knowledge about the AmigaDOS. If not, use this course for starting your programming career....

## 1.3 That`s me ;)

Snail Mail
Jens Tröger
Hochschulstraße 48, 11-4
01069 Dresden
Germany

Phone
(+49) 351/4701609

E-Mail
jt18@irz.inf.tu-dresden.de

WWW
http://www.inf.tu-dresden.de/~jt18
http://www.savage.light-speed.de

IRC
Nick: _savage
Channel: #amigager

## 1.4 C= Installer vs. InstallerNG

If you know the C= Installer, you should have noticed its ugly interface, the amount of failures and the bad useability. If you have already programmed the C= Installer, you know that the language isn't up to date anymore. Thats why I started to implement a new Installer: the InstallerNG. This new Installer looks really nice, has nearly no bugs and runs very stable. For the programmer, it offers new and also enhanced functions. For a list of all the new things look here.

Since the InstallerNG is fully compatible to the C= Installer (at least to the
new Installer of the AmigaOS 3.5) this guide remains valid for both Installers!
The language of the C= Installer can be seen as a subset of the InstallerNG's
language and I will note, whether the C= Installer understands a specific ↩
    function
or not by a {NG}. So it does not matter, if I talk about the InstallerNG or the
C= Installer - both are meant, when I say Installer.

Some may be upset about the fact, that scripts run into errors, even if these
scripts did work fine with the C= Installer. This is not the fault of the ↩
    InstallerNG.
The C= Installer is very lazy - it does no checks, accepts very much errors and
did never notify the programmer/user about errors. The InstallerNG is definitely
not lazy and reports errors!

## 1.5 Versions if the Installer

I think, there are three important version of the Installer

older than 42.9/42.12
  You can be sure, that every Installer has at least version 42.9/42.12.
  These versions extended the old language by some new features and functions.
  I guess, when you programm a script, you can be sure that this script runs
  at least version 42.12, since this was the last official release. But a
  version check would be safe.

44.10 (AmigaOS 3.5)
  This version comes with the new AmigaOS 3.5 and was done by Jochen Becher
  (Haage & Partner). He just added some new functions to the Installer language
  which support multimedia files and a simple backtrace mechanism.

44.10 (InstallerNG 1.4+)
  My InstallerNG is compatible the the latest version of the original Installer.
  The scripts run without problems, but the programmer can make use of the
  new functions of the InstallerNG.

## 1.6 The installation of the Installer

The Installer is a system tool and, thus, can be found in your C:
drawer. For those who want to use my InstallerNG (recommended!) - just
double-click at the Install icon and follow the steps.

In fact, it does not matter where the Installer can be found as long as
it resides in the systems path.

## 1.7 What's new for the InstallerNG

```
                Here you find all the things I added to the InstallerNG and, ↩
                   thus, which features
are not supported by the old Installer. So if you want to use these new features
and run the script on the original Installer you may run into errors. Thats why
a version check is very important!

No restrictions
  The original Installer cannot handle larger strings (currently, I don't know ↩
     what
  is meant by "larger" strings...). With the InstallerNG a string (and the value ↩
      of
  a string variable too) can be as long as it fits into your memory.

Nice GUI
  The builtin-gui is based on a BOOPSI class-collection, which was also written
  by me; these classes allow easy font-adaption, resizing and support the ↩
     MagicWB
  pens. Additionally, you may "plug-in" other gui-systems (like MUI, BGui, ...)
  via a shared library named "installergui.library". This archive also contains ↩
     the
  interface definition, such that anyone could program a custom gui for my ↩
     InstallerNG.

  Furthermore, the help window can stay open, while you install your packages;
  this is a builtin feature and should be provided by every GUI.

Comfortable WB-Start
  If you run the InstallerNG from WB and give it no script via tooltypes
  a requester pops up which asks you whether you want to load a script by a
  file-requester or if you want to app-iconify the installer. If you drop a
  script-file on the application icon the InstallerNG gets started.

Returncode
  The InstallerNG returns RETURN_OK (0) if everything of the installation
  went fine, or, in case of an error, it returns RETURN_FAIL (20). This could
  be useful, if you call the InstallerNG from a script and the script wants
  to check whether the InstallerNG was successfull or not.

Flexible interpretation
  If an error raises while the interpretation process, the InstallerNG provides
  to continue at the very next function. Please be careful with this option,
  because going on may lead to some other errors, but often it's really useful ↩
     to
  finish the (uncomplete) installation.

New builtin variables
  @installer-ng-version -- the version of the InstallerNG
  @proceed-button -- holds the text for the "Proceed with install"-button

Constants
  - TRUE/DOSTRUE and FALSE/DOSFALSE are now constants and cannot be modified
  - NOVICE, AVERAGE and EXPERT are builtin constants, so you can use them ↩
     instead
    of 0, 1 and 2 (usefull for
               CONFIRM
                and
```

```
              USER
               functions)

New Tooltypes/CLI-Arguments
  LAZYCOMPILE: if set, then the InstallerNG is as lazy as the C= installer
               is. that means, InstallerNG skips its semantic check procedures
               to be more compatible
  DEBUGMODE: if set, then InstallerNG will switch on it's debugmode
  CREATEUNINSTALL=CUI: if set, then InstallerNG creates an uninstall skript
  COPYFILECOMMENT=CFC: if set, every copied file will be commented with the ↩
     package name
  ALWAYSCONFIRM: if set, every action has to be confirmed in every user-level!
  NOSYSDELETE: if set, calls to DELETE from system drawers will be ignored

Interuptable Interpretation
  The InstallerNG can be interupted everytime by sending the CTRL-F signal to ↩
     its
  process. This option allows to break out of infinite loops.

Local environments
  Everytime you want to, you are allowed to create a new environment (i.e. to ↩
     declare
  several new variables). Inside this environment you can run some code, which ↩
     uses
  the local variables prior the global ones. See the function
              LET
               for more details.

SOOP - Simple Object Oriented Programing
  With help of the new functions PUT-PROPERTY, GET-PROPERTY and REMOVE-PROPERTY ↩
     the
  InstallerNG implements LISP-like property-lists for symbols. Imagine of a ↩
     symbol
  as an object and the properties as the objects attributes. Furthermore, if you ↩
      write
  PROCEDURE's, which are able to operate on an object's attributes, you just can ↩
      produce
  simple OO code :)    ...without a class hierarchy, but object oriented!

UNDO-REDO environments
  Using the function
              SWING
               you are able to build an environment, in which you can
  "swing" from one (topmost) function to the next. When reaching the last one, ↩
     the
  installation may proceed. This looks/works much like the MS-Setup program :)

  With v44 of the C= installer, you are able to simulate such an environment by
  special
              TRACE
               and
              RETRACE
               functions and the
              BACK
               parameter

Full installation control
```

```
  If you want to, the InstallerNG asks for confirmation of every action, no  ↩
     mattter
  what the script-programmer codes in his installer script

AppWindow
  InstallerNG can now act as an socalled "AppWindow", i.e. you may drop files  ↩
     into
  the window and InstallerNG uses them. This only works, when the InstallerNG  ↩
     asks
  for a file or directory (see
              ASKFILE
              ,
              ASKDIR
              )

Enhanced Functions


               DATABASE

               EXISTS
                New Functions


               BEEP

               COMPARE

               DELAY

               FINDBOARD

               FLUSHLIBS

               LET

               NOP

               RANDOM

               REBOOT

               SETENV

               SIMULATE-ERROR

               SWING

               GET-PROPERTY

               PUT-PROPERTY

               REMOVE-PROPERTY
```

## 1.8 'Hello World' - the first working program

Now let's write our first program for the Installer. Run an editor (something like GoldED, even the ED command of your shell is good enough) and just write the following line:

```
(message "Hello world")
```

Save this as "t:helloworld" and open a shell window (see your Workbench manual for help). Type "installer t:helloworld" and press enter. The Installer should open a window, which asks you, if you are "Novice", "Average" or "Expert". Select "Expert" and press the "Proceed" button. Currently, just ignore the next panel and press the "Proceed" again. Now you should see our "Hello world" text. Quit the Installer by pressing the "Proceed" button again.

Thats all... you did it!

## 1.9 The language - an overview

Have a look at the
Hello World
program to see a very small but legal
script. The language has a very simple structure. Some may say, it is LISP but they are wrong (LISP is an old dirty-functional language). It may look so, but has
nothing in common.

A script consists of a collection of functions. A function just starts with an opening bracket, followed by the functions name and several (or maybe zero) arguments for this function and ends with a closing bracket. Examples for such functions are

```
(+ 5 2)  ; just add 5 and 2
(message "really simple, isn't it?!")  ; show a message
```

You see, comments start with a semi-colon and end with the end of the line. The InstallerNG also supports multi-lined comments, which are enclosed in "/*" and "*/" (like comments in C). Furthermore, the Installer does not care for upper or lower letters!

It may look strange to you, that we write the function symbol at first, followed by its arguments. In mathematics this style is called "prefix notation". Everyone
knows "infix notation" - the school-like addition is infix, because we write the functional symbol between its arguments. Of course there is also an "postfix notation", guess, how this looks like!

Every function deliveres a result and this result has a type. This makes the it possible to use every function as argument to another function, if the types are valid. This means, you cannot use a function, which evaluates to a string

result, as argument to a function which expects a number argument. A runtime error will be produced in such cases. Try to calculate the result of this expression:

```
(+ 1
   (+ 2 3)
   4
   (- 5
      (* 2 3)
      (/ 9 3)
   )
)
```

Of course we need variables! A veriable can be declared by using the SET function:

```
(set #number 5)
(set #string "hello")
```

The first SET defines a variable "number" of type NUMBER and gives it the value 5. The second function defines a variable "string" of type STRING and the value of this variable will be the string "hello". You see, the Installer distinguishes exactly two types: the numbers and the strings. Later, we will add a new type, but for now, this is quiet enough. Additionaly, the Installer offers several builtin variables, which hold information about the current Installer environment. The script can use these variables as they were defined by the user. It is a convention, that builtin variables start with "@" and the user defined variables with "#", but this is definitely no must!

A script can become very large. In such cases it would be useful to have custom functions for maybe version checks, copying end so on. You can define your own functions by using the PROCEDURE function. A user defined function should start with a "P_" to avoid collisions with later extension to the builtin function set .

```
(procedure P_Error #errobject #errcode
  (
    (beep) {NG}
    (message "Error #" #errcode " with " #errobject)
    (exit (quiet))
  )
)

..

(P_Error "my_file" 5)
```

This defines a function "P_Error" which expects two arguments: #errobject and #errcode. You can invoke such functions, just as they were builtin - simply call them.

## 1.10 How can I start the Installer

Just like nearly every other Amiga tool, you can start the Installer either through a shell or from Workbench.

## 1.11 Running from Shell/CLI

The Installer can be started by simply typing its name and the script file. The script file argument is the only one argument, which you must specify! Every other argument is optional. This is the argument template of the InstallerNG:

```
SCRIPT/A,APPNAME/K,MINUSER/K,DEFUSER/K,LOGFILE/K,LANGUAGE/K,
NOPRETEND/S,NOLOG/S,NOPRINT/S,LAZYCOMPILE/S,DEBUGMODE/S,
CREATEUNINSTALL=CUI/S,COPYFILECOMMENT=CFC/S,ALWAYSCONFIRM/S,
NOSYSDELETE=NSD/S
```

APPNAME specifies the name of the application you want to install. Usually this is the name of your tool. MINUSER sets the minimal operation mode for the Installer and DEFUSER presets the operation mode. The user may change the operation mode by selecting a mode in the first welcome panel (refer to the WELCOME function). Use LOGFILE to set the file, which will be handled as an installation protocol or set NOLOG, if you want to forbid any logging actions. NOPRINT disables the logging to the standard printer. If you set NOPRETEND then the user cannot turn on the pretend mode. In pretend mode, the Installer just simulates an installation process. The LANGUAGE specifies the language, which should be used in the script.

The rest of the arguments is valid only for the InstallerNG and they set the "Advanced options". LAZYCOMPILE turns off any check procedure at startup and the InstallerNG does not look for errors during compilation. DEBUGMODE turns on the debug console and prints useful warnings. Use CREATEUNINSTALL to produce an uninstall script from the current installation session. If you run this produced script again with the Installer, it will de-install the package. COPYFILESCOMMENT just comments every copied file with the name of the current application name (see APPNAME argument) by appending the old file comment to the application name. For full installation control, you should set the ALWAYSCONFIRM argument, which forces the Installer to ask for confirmation everytime. NOSYSDELETE avoids the deletion from system drawers

like C: or LIBS: or whatever...

## 1.12 Running from WB

If you run the Installer from Workbench, you can set up a working environment for
it by specifying tooltypes.

```
SCRIPT=<scriptfile>
APPNAME=<name>
MINUSER=<novice|average|expert>
DEFUSER=<novice|average|expert>
LOGFILE=<logfile>
LANGUAGE=<language>
PRETEND
LOG
NOPRINT
ICONIFY {NG}
LAZYCOMPILE {NG}
DEBUGMODE {NG}
CREATEUNINSTALL {NG}
COPYFILECOMMENT {NG}
ALWAYSCONFIRM {NG}
NOSYSDELETE {NG}
```

Except the ICONIFY tooltype, these tooltypes are equal to the shell arguments and,
thus, I do not write the meaning here again.

ICONIFY holds, if you give no SCRIPT argument. Usually, the Installer would ask,
whether the user wants to load a script or just iconify the Installer. Using this
tooltype forces the Installer to immediately iconify.

## 1.13 The types of the Installer

The Installer distinguishes betweem two main types: STRING
and NUMBER. Additionally, the parameter functions do not return
any of these main types, but a PARAM type just to notify, that
such a parameter function was executed.

Now forget about the PARAM type, it is internally. Only work
with the STRING and NUMBER types!

## 1.14 The Errors

```
To understand these errors think of the syntactical structure of any program:

A program consists of one or more functions or function lists. An expression can
be either a number, a string, a variable or a new function. Functions are  ↩
    enclosed
in paranthesis, the first symbol can be anything but a number and a function-
specific number of arguments. An argument can be again any expression.

Syntax Errors

  ( expected
    The Installer needs a new function

  ( or function expected
    The Installer needs the beginning of a new function or the name of
    a function. (you may have wrote a number)

  Function not allowed here
    A function-name (like ASKFILE...) is used as a parameter to any other
    function. Remove this or enclose it with parenthesis.

  Unexpected EOS
    The end of the source was reached to early. Maybe a missing close- ↩
       parenthesis
    leads this error.

  Expression expected
    Any expression is needed here.

  Functional expression needed
    The first expression behind opening parenthesis must be an identifier or a
    string. What you wrote is maybe a number.

  ) expected
    You forgot a ")" ???
```

## 1.15 The Installer Language

```
                The language used by the Installer is a simple, imperative  ↩
                   language. Since I
like functional languages, I tried to give this language a "functional"
touch, i.e. every expression can be evaluated and returns a typed result.
Furthermore I started to make the language a bit type stronger, because types
are very needful for preventing errors. But don't panic, this language is
definitly not functional (it has side-effects!) and very easy to use.

Imagine of the Installer as the Interpreter of a given script. Interpreter means ↩
    ,
the Installer first looks at the whole program (i.e. the script) and then  ↩
    fetches
the first function, evaluates it and maybe uses the result as an argument for  ↩
    the
next function, then it gets the next function, evaluates it... and so on. For  ↩
    more
```

```
detailed information see section Technical You may have noticed the syntax:
it may look strange to some, but it is a simple prefix notation. "Prefix" means,
that the functional symbol is at first position, followed by its parameters.  ↩
    Every
function must be enclosed by parenthesis. For example to simply add two numbers,
you must write: (+ 2 3)

A complete list of all functions you will find here. Of course you find  ↩
    everything
of a good imperative language: conditionals, variables, a big set of built-in
functions, the ability to define custom functions and much more.

Since  the original Installer does not offer all the things I wanted to use, I
added some more functions and features. See the What's New section for more
information.

NOTE: everytime I talk about a string or a number value, you are allowed to use
      an identifier of type string or number or an expression (function,
      function list) which deliveres a result of type string or number.


               Symbols

               Syntax
                 Builtin functions

               Builtin variables
                 Advanced features
```

## 1.16 The symbols of the language

```
                Symbols are the bricks of every programming language. A variable ↩
                   , a number
or even the keywords are the symbols (also called: tokens) of a language. By
writing a meaningful sequence of symbols, you just write your program. Here
you will find the symbols for the Installer programming langage. This is not
a formal definition, but I think it is useful.

Spaces
  Spaces are the characters between other symbols and are skipped, when the
  InstallerNG scanns the script-file. Every character with an ASCII less or
  equal 32 gets handled as a space.

Parenthesis
  Parenthesis are used to enclose functions and function lists. Only "(" and ")"
  are legal for that.

Strings
  A string is enclosed in either "..." or '...' and must not contain
  linefeeds. Special characters start with a backslash, followed by
  the character, which should appear in the string itself:

     \0     for a NULL character (ASCII-0)
     \b     beep (ASCII-8)
```

```
      \t \h  tabulator (ASCII-9)
      \n     linefeed (ASCII-10)
      \v     ? (ASCII-11)
      \f     ? (ASCII-12)
      \r     carriage return (ASCII-13)

      \\     for a backslash itself

      \o     octal encoded number
      \x     hex encoded number

      \"     to use a " inside of a "..." string
      \'     to use a ' inside of a '...' string

   Example: "string"
            "first line\nsecond line"
            "numbers are: 123 \o70 \xffff"
            "string 'cite'"
            "string \"cite\""
            'string "cite"'
            'string \'cite\''


Numbers
  There are three types of numbers:
    binary: starting with "%" and followed by a sequence of "0" and "1"
    decimal: starting with a number or a "+" or a "-" and followed
      by a sequence of "0"..."9"
    hex: starting with "$" and followed by a sequence of "0"..."9"
      and "a"..."f" (lower or upper case allowed)

    Example: -4 +53 23 %101011 $A35B

Identifiers
  Functions
    Functions are character sequences (like variables), but the Installer  ↩
       identifies
    them as function sybols. See the
              builtin functions
               section for which symbols
    are reserved. Case insensitive.

    Note: You must not use reserved names for your identifiers! Reserved names
    are e.g. the builtin function names or the builtin variable names.

    Example: < >= / AND ASKFILE

  Variables
    Are character sequences, which are not builtin functions. Note, that only  ↩
       the
    first 32 characters count! Case insensitive.

    Example: #bla ___*A^ popopop

Comments
  Single line comments start with a semicolon ";" and end with a return (ASCII ↩
     -10)
```

```
  Multi lined comments can be enclosed in "/*" and "*/" and may contain anything
  but a EOF (ASCII-0). Note, that multi lined comments are new with the  ↩
     InstallerNG
  and NOT supported by the C= Installer!

  Example: ; single lined comment

           /*
               multi lined comment
           */
```

## 1.17 The layout of the language

```
                 Defining the syntax for a programming language means to say,  ↩
                    which sequences
of symbols build a correct source code. It does not make sense to write some
numbers and variables -- the Installer has specific rules for which symbol must
follow another symbol. You know, that a function must be enclosed in brackets
and can have some arguments. This is a syntactical rule for the Installer
programming language. This syntax definition does whether define the types of  ↩
   the
arguments nor the legal count of arguments for the functions! This so called
"context sensitive" check can be done after the syntax check, or can be skipped
by specifying the LAZYCOMPILE option at
              startup
              .

Below you find both, an informal and a formal syntax definition.

Informal
  A legal script contains at least one function. A function opens with
  a "(" followed by the functional symbol (this is called "prefix notation")
  followed by zero or more argument expressions; a function ends with a ")".
  A valid expression can be either a number, a string, an identifier or a
  function again. In addition, you can group a collection of functions by  ↩
     enclosing
  them again with brackets.

Formal
  Below you find the EBNF description:

     <prog>         ::=   [ <func> ]+

     <func>         ::=   "("  "IDENT"  [ <expr> ]* ")"
                        | "("  "STRING"  [ <expr> ]* ")"
                        | "("  [ <func> ]+ ")"

     <expr>         ::=   "NUMBER"
                        | "STRING"
                        | "IDENT"
                        | <func>
```

```
  For a definition of the symbols NUMBER, STRING and IDENT see the

              Symbols
               section.
```

## 1.18 Builtin variables

```
                The builtin variables are declared an initialized by the  ↩
                   Installer itself at
startup. They hold useful information about the environment, in which the script
will execute or you can modify the environment by setting these variables. A
script can use these variables just like custom ones an may, for instance,  ↩
   localize
the texts. If you set a new value for some variable, you must care for its type,
otherwise the script may run into runtime errors.

@abort-button
  The text, which should be used for the "Abort installation" button

  Default: "Abort installation"
  Type: STRING

@app-name
  Name of the application to install. This will be used for the "Comment every  ↩
     File
  with Packagename" option too.

  Default: "user-application"
  Type: STRING

@icon
  The path and name of the script, i.e. the icon, where the Installer was  ↩
     started
  from (WB start) or the full path to the script when started from shell.

  Default: the script, even
  Type: STRING

@execute-dir
  The working directory for the commands started with
              RUN
               or
              EXECUTE
                  Default: "" (should be the scripts dir)
  Type: STRING

@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.

  Default: "Work:"
```

```
  Type: STRING

@language
  The language, which is currently used by the Installer. This depends on the  ↩
     preferred
  system language and the available catalog file

  Default: "english"
  Type: STRING

@pretend
  The state of the "pretend" flag (1 for pretend)

  Default: 0 or set by startup-args
  Type: NUMBER

@proceed-button {NG}
  This holds the text for the "Proceed with Install"-button. Use this to  ↩
     customize
  the button text. Useful if you run uninstall-scripts

  Default: "Proceed with Install"
  Type: STRING

@user-level
  The user level, which is the Installer running on. (0 for "Novice", 1 for " ↩
     Average",
  2 for "Expert"). Note: this can be set by the
              USER
               function, do not use
              SET
                  for this case!
  Note: the InstallerNG affers the builtin constants NOVICE, AVERAGE and
  EXPERT for a easier usage.

  Default: 0 or set by startup-args
  Type: NUMBER

@installer-version
  Current version of the Installer. Note: this does not equal the version of the
  InstallerNG!

  Default: 0x002c000a (which is a 44 in the upper word and a 6 in the lower one)
  Type: NUMBER

@installer-ng-version {NG}
  This holds the current InstallerNG version. By testing this value against zero ↩
     ,
  you can determine whether you run on the old Installer (zero) or the  ↩
     InstallerNG
  (not zero)

  Default: 0x00010004 (which is 1 in the upper 16 bits and 4 in the lower)
  Type: NUMBER

@error-msg
```

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

  Default: ""
  Type: STRING

@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.

  Default: ""
  Type: STRING

@ioerr
  In case of a DOS-error, this variable holds the error-code.

  Default: 0, set by every DOS error
  Type: NUMBER

@each-name
@each-type
  Name and type (file or directory) of the currently examined file of the
              FOREACH
               function

  Default: depends
  Type: STRING/NUMBER

@askoptions-help
@askchoice-help
@asknumber-help
@askstring-help
@askdisk-help
@askfile-help
@askdir-help
@copylib-help
@copyfiles-help
@makedir-help
@startup-help
  The bultin help texts.

  Default: depends
  Type: STRING
```

## 1.19 Builtin functions

```
                The Installer provides a large amount of functions for nearly  ↩
                   everything
```

you want. You can query the user, examine the system, manipulate files,
run scripts and programs, you can show effects (needs the datatypes) and
last but not least you have many functions for calculating stuff.

The InstallerNG offers some more functions which (I guess) are hardly
needed today. These functions are marked with {NG}. Furthermore, the InstallerNG
enhanced some functions without losing compatibility. These enhancements are
noted by {+}.

With the new AmigaOS 3.5 the Installer offers some more functions. These
functions are also supported by the InstallerNG and are marked with a {44.6}
(which is the minimum version of the new Installer of the AmigaOS 3.5).

Note: the specification of the arguments (if any) uses a special notation --
i.e. [arg]+ means several arguments, but at least one has to be given; [arg]*
means that this function can have zero or more arguments and [arg]{n-m} (or
even [arg]{n}) means n 'til m (even only n) arguments. For simplification,
I write just [arg] to denote only one argument.

Conditional
  These functions control the working flow of your script. Using conditions
  you can decide where to continue the script execution. Note, that for ↩
     conditions,
  an empty string will be interpreted like the number zero: as FALSE

IF

SELECT

UNTIL

WHILE
 Multimedia and visual support
  For an entertaining installation, the new Installer provides functions for
  handling pictures, sounds, animations and so on via datatypes. In addition,
  you can run the Installer on a custom screen with simple background features
  by using the EFFECT function.
  NOTE: the InstallerNG implements asynchronous viewers!

CLOSEMEDIA
{44.6}

SETMEDIA
{44.6}

SHOWMEDIA
{44.6}

EFFECT
{44.6}

Mathematical stuff

  Comparison

```
                =

                <>

                >

                >=

                <

                <=

                COMPARE
                {NG}

    Traditional math

                +

                -

                *

                /
                   Logical operations

                AND

                OR

                XOR

                NOT
                   Bit testing & manipulation

                BITAND

                BITOR

                BITXOR

                BITNOT

                IN

                SHIFTLEFT

                SHIFTRIGHT
                 Querying the user
In most cases the script needs information from the user, e.g. where to
install the package or by asking what to install. This can be realized by
these following functions.


                ASKDIR
```

ASKFILE

ASKSTRING

ASKNUMBER

ASKCHOICE

ASKOPTIONS

ASKBOOL

ASKDISK
Notifying the user

BEEP
{NG}

COMPLETE

MESSAGE

WELCOME

WORKING
Examining the system

In most cases the script needs to know about the system environment. Several functions can be used to find out different system's properties and you should use these functions rather than runing external commands.

DATABASE
{+}

FINDBOARD
{NG}

GETASSIGN

GETDEVICE

GETDISKSPACE

GETENV

GETSIZE

GETSUM

GETVERSION

QUERYDISPLAY
{44.6}

String manipulation

It is often needed to modify, append or extract strings.

```
              CAT

              PATMATCH

              STRLEN

              SUBSTR
               File handling and DOS
Everything you can think of for file manipulation, copying and
handling icons and related stuff.


              COPYFILES

              COPYLIB

              DELETE

              EARLIER

              EXECUTE

              EXISTS
              {+}

              EXPANDPATH

              FILEONLY

              FOREACH

              ICONINFO

              MAKEASSIGN

              MAKEDIR

              PATHONLY

              PROTECT

              RENAME

              REXX

              RUN

              STARTUP

              TACKON

              TEXTFILE

              TOOLTYPE
               Debugging and additional execution control
```

In case of an error, a script could clean up its environment or
undo some steps and so on by using some of these functions. Furthermore,
this is important and very helpfulf when programming scripts.

For being more userfriendly, you should use the new functions
SWING or TRACE/RETRACE just to give the user a chance to undo/redo
his initial settings or something like that.

ABORT

DEBUG

NOP
{NG}

ONERROR

EXIT

REBOOT
{NG}

RETRACE
{44.6}

SIMULATE-ERROR
{NG}

SWING
{NG}

TRACE
{44.6}

TRANSCRIPT

TRAP

USER

Workbench support

Starting with the new AmigaOS 3.5, there is an interface for tools to
handle disks, drawers etc as so called "Workbench Objects". For the user these
functions work, as the user itself had clicked on a drawer or tool and the
AmigaOS will perform the related action automatically. The Installer also
supports this interface with these functions:

CLOSEWBOBJECT
{44.7}

OPENWBOBJECT
{44.7}

SHOWWBOBJECT
{44.7}

SOOP support
  Only for the InstallerNG. I did this for fun and maybe someone makes use
  of these features?

GET-PROPERTY
{NG}

PUT-PROPERTY
{NG}

READ-PROPERTY-OBJECT
{NG}

REMOVE-PROPERTY
{NG}

SAVE-PROPERTY-OBJECT
{NG}

Others
  This is the rest of the functions

DELAY
{NG}

FLUSHLIBS
{NG}

LET
{NG}

PROCEDURE

RANDOM
{NG}

SET

SETENV
{NG}

Parameter Functions
  This set of functions is very special. It does not make sense to use
  them like the ones above, but you must use these functions as arguments
  to some other functions. These functions can modify the local environment
  of different functions like COPYFILES or the query functions.

ALL

APPEND

ASSIGNS

BACK

```
                {44.6}

                CHOICES

                COMMAND

                CONFIRM

                DEFAULT

                DELOPTS

                DEST

                DISK

                FILES

                FONTS
                    GETDEFAULTTOOL
    GETPOSITION
    GETSTACK
    GETTOOLTYPE

                HELP

                INCLUDE

                INFOS

                NEWNAME

                NEWPATH

                NOGAUGE

                NOPOSITION

                NOREQ

                OPTIONAL

                PATTERN

                PROMPT

                QUIET

                RANGE
                    RESIDENT

                SAFE

                SETTOOLTYPE

                SETDEFAULTTOOL
```

```
SETSTACK

SOURCE

SWAPCOLORS
```

## 1.20 Advanced features

```
Defining custom functions
Often it should be useful to define custom functions, which are called
as they were part of the Installer. Use the
PROCEDURE
function for this
purpose. The name of custom functions should start with a "P_" just to avoid
collisions with future builtin functions. In some cases it is very useful to ↩
   have
local variables for such a function. The old Installer does not provide this
feature, but with my InstallerNG you can define so called "let environments" ( ↩
   see
the
LET
function). This environment can be used to create a local ↩
   environment
for a custom function. You must do so, if your functions are recursive, i.e. ↩
   if
they call themselves.

A custom function is defined by its name, a number (even zero) of arguments
and the body of the function itself.


    /* convert a version number to a readable string */
    (PROCEDURE P_version-to-string                                  ; name
               #ver                                                 ; argument
               ("%ld.%ld" (/ #ver 65536) (BITAND #ver 65535))       ; body
    )

    /* count down recursively by using LET */
    (PROCEDURE P_recursive                                          ; name
               #arg                                                 ; argument
               (LET (SET #local #arg)                               ; body
                    (
                      (IF #local
                          (P_recursive (- #local 1))
                          (NOP)
                      )
                    )
               )
    )


String formatting
  Some may know the ANSI-C function sprintf(), which takes a template string and
```

```
  a number of arguments and creates a new string by replacing the wildcards in ↩
     the
  template string by the related argument. The Installer has this functionality ↩
     too.
  If the functional symbol (remember: this is the leftmost one, because this is ↩
     a
  prefix language) is of type string (and it does not matter whether it is a ↩
     string
  itself or a variable of type string!), then this string gets handled like a ↩
     format
  string (a template), and the following expressions are the format parameters. ↩
     Possible
  wildcards are (in fact, the Installer uses exec.RawDoFmt() so that you can ↩
     write
  every valid template string here):

    %s   - string
    %lc  - character
    %ld  - decimal number
    %lu  - unsigned decimal number
    %lx  - hex number

  For example, if you write


      ("string '%s' at 0x%lx has %ld chars" "bla" $0000a123 3)


  you will get the following as result of the evaluation:


      "string 'bla' at 0xA123 has 3 chars"


  Note: the Installer does no type checking for the arguments and every argument
  comes as a long value (32 bit).

Function groups
  You can join as many functions as you want into one block: simply put ↩
     parenthesis
  around the functions you want to yoin. The result of this block is the result ↩
     of
  the last evaluated function. This is often used, if you want more than one
  functions be part of an (e.g.) IF or just to make the code more readable.


      (IF (= #bla #surz)                    ; the condition
          (MESSAGE "#bla equals #surz")     ; "THEN" expression
          (                                 ; "ELSE" block
            (BEEP) {NG}
            (MESSAGE "#bla equals #surz")
          )
      )
```

## 1.21 Some theoretical stuff

```
Here you find some additional information about the InstallerNG. One can find
how the Interpreter itself works or some theoretical aspects of the language.

For a big overview about the specification and implementation (german only)
please have a look at my homepage (note that this script is obsolete with
version 0.3+).

Interpretation
  The interpreter does it`s job using "call-by-name" strategy. This means it  ↩
     first
  evaluates the expression at the first (functional) position of a function,  ↩
     which
  results into a function call. The called function then evaluates the arguments
  and uses the results of this as arguments. As you can see this process is  ↩
     recursive.

  An example: given the following functions (set i (+ 3 4)) the Installer  ↩
     produces
            such a tree:
                                      set
                                       ^
                                     i   +
                                         ^
                                        3 4

  Now the interpreter arrives at the top node "set". This means the interpreter
  calls the internal function "set" and gives as arguments its childs. These  ↩
     childs
  are an identifier "i" and a sub-tree. Now "set" knows it needs the value of  ↩
     the
  sub-tree (+ 3 4) so it calls the internal "add" function and this functions  ↩
     gets
  both, "3" and "4" as arguments. Now "add" evaluates to "7" and gives the  ↩
     result to
  "set" and now "i" is set to "7".

  To give this an other name: interpreting a program means to visit every node  ↩
     of
  the tree in depth-first-left-to-right-order. Or: go down every (sub)-tree from
  left to right.

Grammar
  The underlying grammer of the language is a context-free LL(1) grammar. Every
  functional symbol has some attributes (e.g. "Number of args" or "Scope"  ↩
     attributes).
  The parser is a top-down one. While parsing the source it calculates some  ↩
     attributes
  for the nodes of the syntax tree. When done with the tree the optimizer starts ↩
      to
  try to optimize the given tree. After this a special function checks whether  ↩
     the
  given tree is correct or not by comparing and calculating attributes. Don"t  ↩
     mix it
```

```
  up with an Attribute Grammar -- this is no one. I just took some ideas from  ↩
     this
  formalism to make the interpretation more stable.
```

## 1.22 Very important notes!!!

```
There are some very important things you must respect:

Version
  The variable @installer-version is set to the current version
  of the Installer. In this version, this variable contains the same
  value as the lates release of the C= installer: 44.6! Additional
  you can check, whether you run on the InstallerNG or not by
  testing the @installer-ng-version variable: the C= installer returns
  a 0 (zero), but the InstallerNG holds its version in this
  variable.


     (IF @installer-ng-version
       (
         ; this InstallerNG version
       )

       (
         ; the original amiga installer
       )
     )


Most public programming faults

  Uninitialized variables
    Most of the programmers forget to set the variables before use.
    The original installer accepts this and sets these variables to
    0 (zero). The InstallerNG warns you but behaves in the same
    way.
    Use the debug output to find uninitialized variables!

  Wrong usage of parameter functions
    There come some function calls like this:


       (ASKFILE (IF (= 0 #bla) (PROMPT "Blurp"))
                (IF (= 1 #bla) (PROMPT "Barg"))
                (IF (= 2 #bla) (PROMPT "Tirz"))
                (HELP "Help...")
                (DEFAULT "SYS:")
       )


    This results in a "Warning: wrong number of arguments", because ASKFILE
    is missing the PROMPT argument. Note, that this is only a semantic warning,
    the InstallerNG behaves in the right way! For future scripts use
    something like this:
```

```
    (ASKFILE (PROMPT (IF (= 0 #bla)
                         "Blurp"
                         (IF (= 1 #bla)
                             "Barg"
                             "Tirz"
                         )
                     )
             )
             (HELP "Help...")
             (DEFAULT "SYS:")
    )
```

Weird syntactic/semantic constructs
It is amazing what people code and more funny what the C= Installer compiles...

```
    (IF <condition> <then> <else> <what-the-hell-is-this>)
```

Or something like this:

```
    (ASKOPTIONS (CHOICES 1 2 3
                  (DEFAULT 1
                    (HELP "little help..."
                      (PROMPT "choose!")
                    )
                  )
                )
    )
```

Parameter functions at wrong positions
Some scripts come along with wrong positions for the parameter functions, e. ↩ g.

```
    (MAKEDIR (SAFE) (INFOS) "sys:new_dir")
```

This does not work and if you have a look at the original documentation of ↩ the
installer language, you will find the correct expression:

```
  (MAKEDIR "sys:new_dir" (SAFE) (INFOS))
```

## 1.23 All functions in alphabetical order

This is the index for all functions in alphabetical order. Go here

for
additional explanation of the functions and a grouped overview.

=

<>

>

>=

<

<=

+

-

*

/

ABORT

ALL

AND

APPEND

ASKDIR

ASKFILE

ASKSTRING

ASKNUMBER

ASKCHOICE

ASKOPTIONS

ASKBOOL

ASKDISK

ASSIGNS

BACK
{44.6}

BEEP
{NG}

BITAND

```
BITNOT

BITOR

BITXOR

CAST-INT
{NG}

CAST-STRING
{NG}

CAT

CHOICES

CLOSEMEDIA
{44.6}

CLOSEWBOBJECT
{44.7}

COMMAND

COMPARE
{NG}

COMPLETE

CONFIRM

COPYFILES

COPYLIB

DATABASE
{+}

DEBUG

DEFAULT

DELAY
{NG}

DELETE

DELOPTS

DEST

DISK

EFFECT
{44.6}
```

```
EXECUTE

EXISTS
{+}

EXIT

EXPANDPATH

EARLIER

FILEONLY

FILES

FINDBOARD
{NG}

FLUSHLIBS
{NG}

FONTS

FOREACH

GET-PROPERTY
{NG}

GETASSIGN
    GETDEFAULTTOOL

GETDEVICE

GETDISKSPACE

GETENV
    GETPOSITION

GETSIZE
    GETSTACK

GETSUM
    GETTOOLTYPE

GETVERSION

HELP

ICONINFO

IF

IN

INCLUDE

INFOS
```

```
LET
{NG}

NEWNAME

MAKEASSIGN

MAKEDIR

MESSAGE

NEWPATH

NOGAUGE

NOP
{NG}

NOPOSITION

NOREQ

NOT

ONERROR

OPENWBOBJECT
{44.7}

OPTIONAL

OR

PATHONLY

PATMATCH

PATTERN

PROCEDURE

PROMPT

PROTECT

PUT-PROPERTY
{NG}

QUIET

QUERYDISPLAY
{44.6}

READ-PROPERTY-OBJECT
{NG}
```

```
RANDOM
{NG}

RANGE

REBOOT
{NG}

REMOVE-PROPERTY
{NG}

RENAME
    RESIDENT

RETRACE
{44.6}

REXX

RUN

SAFE

SAVE-PROPERTY-OBJECT
{NG}

SELECT

SET

SETDEFAULTTOOL

SETENV
{NG}

SETMEDIA
{44.6}

SETSTACK

SETTOOLTYPE

SHIFTLEFT

SHIFTRIGHT

SHOWMEDIA
{44.6}

SHOWWBOBJECT
{44.7}

SIMULATE-ERROR
{NG}

SOURCE
```

```
STARTUP

STRLEN

SUBSTR

SWAPCOLORS

SWING
{NG}

TACKON

TEXTFILE

TOOLTYPE

TRACE
{44.6}

TRANSCRIPT

TRAP

UNTIL

USER

WELCOME

WHILE

WORKING

XOR
```

## 1.24 ABORT

```
                This exits the installation with the given messages and executes
the {"ONERROR" link ONERROR} functions (if any)

Template
  (ABORT [msg]* )

Parameters
  [msg] - strings which will be concatenated an shown right before
          the InstallerNG starts to execute the ONERROR functions

Options

Result
  Type: NUMBER
  Returns 0
```

```
Note

Example

  (ABORT "Sorry, I have to quit cause: " #reason)


See also

               ONERROR
```

## 1.25 ADD

```
Add all the parameters

Template
  (+ [value]+ )

Parameters
  [value] - the value to be added

Options

Result
  Type: NUMBER
  Returns the sum of all arguments

Note

Example

See also
```

## 1.26 AND

```
The logical "and", i.e. AND deliveres true if all its arguments are true. AND
stops the evaluation with the first false-argument

Template
  (AND [value]+ )

Parameters
  [value] - the value which should logically be tested

Options

Result
  Type: NUMBER
  Returns 1 for true and 0 for false

Note
```

```
Example

See also
```

## 1.27 ASKDIR

```
Ask the user for a name of a directory. The Installer will show a dir requester ↩
    panel
whichs allows an easy selection of the requested directory.

Template
  (ASKDIR [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default directory; this can be a relative path
  NEWPATH - allows to use non-existent paths for DEFAULT
  DISK    - initially show a list of all drives
  ASSIGNS - logical assigns satisfy as well

Result
  Type: STRING
  Returns the user selected directory

Note
  - does return the DEFAULT without a request in "Novice" mode

Example

  (ASKDIR (PROMPT "select a directory")
          (HELP "...")
          (DEFAULT "C:")
  )


See also
```

## 1.28 ASKFILE

```
Ask the user for a file. The Installer will show a file requester panel
whichs allows an easy selection of the requested file.

Template
  (ASKFILE [option]+ )

Parameters

Options
```

```
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default file; this can be a relative one
  NEWPATH - allows to use non-existent paths for DEFAULT
  DISK    - initially show a list of all drives

Result
  Type: STRING
  Returns the user selected file (with expanded path)

Note
  - does return the DEFAULT without a request in "Novice" mode

Example

  (ASKFILE (PROMPT "where can i find your 'delete' command?")
           (HELP "...")
           (DEFAULT "C:Delete")
  )


See also
```

## 1.29 ASKSTRING

```
Ask the user for string. The Installer will show a panel where the
user can enter the desired text.

Template
  (ASKSTRING [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default string

Result
  Type: STRING
  Returns the string, typed by the user

Note
  - does return the DEFAULT without a request in "Novice" mode

Example

  (ASKSTRING (PROMPT "gimme your name")
             (HELP "...")
             (DEFAULT "Linda Perry")
  )
```

```
See also
```

## 1.30 ASKNUMBER

```
Ask the user for number. The Installer will show a panel where the
user can enter the number. Furthermore, your can specify a range and
the user cannot enter numbers outside of this range.

Template
  (ASKNUMBER [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default number
  RANGE   - lower and upper range (if any) for the requested number

Result
  Type: NUMBER
  Returns the number

Note
  - does return the DEFAULT without a request in "Novice" mode

Example

  (ASKNUMBER (PROMPT "gimme a small number")
             (HELP "...")
             (DEFAULT 0)
             (RANGE 0 99)
  )


See also
```

## 1.31 ASKCHOICE

```
Ask the user to select one out of 32 (max) choices. The Installer will show
a panel with several mx buttons and the user has to select one of these.

Template
  (ASKCHOICE [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default choice (default is 0)
  CHOICES - the list of choices, where the user has to select one
```

```
Result
  Type: NUMBER
  Returns the number of the selected choice (starting with zero)

Note
  - does return the DEFAULT without a request in "Novice" mode
  - an empty string for a choice means an invisible mx

Example

  ; should return either 0 (male) 1 (female) or 3 (don't know)
  (ASKCHOICE (PROMPT "what's your sex?")
             (HELP "...")
             (DEFAULT 1)
             (CHOICES "male" "female" "" "don' know")
  )


See also
```

## 1.32 ASKOPTIONS

```
Ask the user to select some out of 32 (max) options. The Installer will show
a panel with several radio buttons and the user has to select the desired ones.

Template
  (ASKOPTIONS [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default choice (default -1)
  CHOICES - the list of choices, where the user has to select one

Result
  Type: NUMBER
  Returns a bitmask of selected choices, where a set bit indicates, that
  the related choices was selected

Note
  - does return the DEFAULT without a request in "Novice" mode
  - an empty string for a choice means an invisible mx

Example

  ; should return 0 (nothing), 1 (upper), 2 (lower) or 3 (both)
  (ASKOPTIONS (PROMPT "what do you like for breakfest")
              (HELP "...")
              (DEFAULT 1)
              (CHOICES "tea" "toast)
  )
```

```
See also
```

## 1.33 ASKBOOL

```
Ask the user to just answer "Yes" or "No" like questions. The Installer
shows two mx buttons and the user selects the related button.

Template
  (ASKBOOL [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEFAULT - the default choice (default 0)
  CHOICES - replace at least one of the both "Yes" and "No" by custom ones

Result
  Type: NUMBER
  Returns 1 for "Yes" and 0 for "No"

Note
  - does return the DEFAULT without a request in "Novice" mode

Example

  (ASKBOOL (PROMPT "Amiga is really cool")
           (HELP "...")
           (DEFAULT 1)
           (CHOICES "Sure" "Never)
  )


See also
```

## 1.34 ASKDISK

```
Ask the user to insert a specific disk. As long as this disk is not
available, the Installer will wait.

Template
  (ASKDISK [option]+ )

Parameters

Options
  PROMPT
  HELP    - tell the user what's going to happen
  DEST    - the requested disk
```

```
    NEWNAME - a name to assign to the disk when inserted for later reference
    DISK    - show drives initially
    ASSIGNS - also accept logical devices

  Result
    Type: NUMBER
    Returns a bitmask of selected choices, where a set bit indicates, that
    the related choices was selected

  Note
    - the volume name must be supplied without a colon! (i.e. you must write
      "env" instead of "env:")

  Example

    ; waits until the user inserts the disk "bla:"
    (ASKDISK (PROMPT "insert disk 'BLA:'")
                (HELP "...")
                (DEST "bla")
    )


  See also
```

## 1.35 BEEP

```
  {NG} Simply flashes the screen and beeps.

  Template
    (BEEP)

  Parameters

  Options

  Result
    Type: NUMBER
    Returns 0

  Note
    This respects your prefs-settings when beeping.

  Example

    (BEEP)


  See also
```

## 1.36 BITAND

```
Does a bitwise AND with the arguments

Template
  (BITAND [value]{2} )

Parameters
  [value] - the arguments for the bitwise logical AND

Options

Result
  Type: NUMBER
  Returns the result of the bitwise AND

Note

Example

See also
```

## 1.37 BITOR

```
Does a bitwise OR with the arguments

Template
  (BITOR [value]{2} )

Parameters
  [value] - the arguments for the bitwise logical OR

Options

Result
  Type: NUMBER
  Returns the result of the bitwise OR

Note

Example

See also
```

## 1.38 BITXOR

```
Does a bitwise OR with the arguments

Template
  (BITXOR [value]{2} )

Parameters
  [value] - the arguments for the bitwise logical XOR
```

```
Options

Result
  Type: NUMBER
  Returns the result of the bitwise XOR

Note

Example

See also
```

## 1.39 BITNOT

```
Does a bitwise NOT with the argument

Template
  (BITAND [value] )

Parameters
  [value] - the value to be bitwise negated

Options

Result
  Type: NUMBER
  Returns the result of the bitwise NOT

Note

Example

See also
```

## 1.40 CAST-INT

```
{NG} Konvert an arbitrary value into a number

Template
  (CAST-INT <value>)

Parameters
  <value> - whatever

Options

Result
  Type: NUMBER
  Returns the number value of [value] or 0, iff the casting was
  not possible
```

```
Note
  Casting fails, iff [value] is a string value and does not contain
  a translateable character sequence (e.g. (CAST-INT "a") will fail)

  equivalent: (+ [value] 0)
              (* [value] 1)

Example

See also
```

## 1.41 CAST-STRING

```
{NG} Konvert an arbitrary value into a string

Template
  (CAST-STRING <value>)

Parameters
  <value> - whatever

Options

Result
  Type: STRING
  Returns the string value of [value], i.e. return a string as it
  is and turn a number into a string, which contains this number

Note
  equivalent: (CAT [value])

Example

See also
```

## 1.42 CAT

```
Concatenate several strings.

Template
  (CAT [string]+ )

Parameters
  [string] - the strings to be concatenated

Options

Result
  Type: STRING
  Returns the concatenation of all argument strings

Note
```

```
  - CAT converts its number arguments into strings, such that you may
    use CAT for getting a string out of an number

Example

  (SET #longstring (CAT "this is a long string with numbers..." 4 5 "82" "!!!"))


See also
```

## 1.43 CLOSEMEDIA

```
                Closes an arbitrary media object, which must have been opened ↩
                   using

              SHOWMEDIA
                Template
  (CLOSEMEDIA [mediaobject] )

Parameters
  [mediaobject] - the object to be closed

Options

Result
  Type: NUMBER
  Returns 0

Note
  see
              SHOWMEDIA
                Example
  see
              SHOWMEDIA
                See also

              SETMEDIA
              ,
              SHOWMEDIA
```

## 1.44 CLOSEWBOBJECT

```
                Closes an arbitrary workbench object, which currently only can ↩
                   be a disk
or a drawer or a trashcan

Template
  (CLOSEWBOBJECT [wbobject] )

Parameters
  [wbobject] - the object to be closed
```

```
Options

Result
  Type: NUMBER
  Returns 1, if CLOSEWBOBJECT could perform this closing action, 0 if
  not or -1, if the machine (i.e. the Workbench) does not support this
  function.

Note

Example
  see
              OPENWBOBJECT
                See also
  see
              SHOWWBOBJECT
              ,
              OPENWBOBJECT
```

## 1.45 COMPARE

```
{NG}
This function compares two values of any, but the same type and returns
the result of this comparison.

Template
  (COMPARE [expr1] [expr2] )

Parameters
  [expr1] - first value
  [expr2] - value, which has to be compared with the first value

Options

Result
  Type: NUMBER
  Returns  1 - [expr1] greater than [expr2]
           0 - [expr1] equals [expr2]
          -1 - [expr1] is smaller than [expr2]

Note
  - both arguments must be of the same type. The Installer tries
    to convert a string into a number if the types are not equal

Example

  (COMPARE 2 2)              -> 0
  (COMPARE 2 "2")            -> 0
  (COMPARE "bla" "nana")     -> -1


See also
```

## 1.46 COMPLETE

```
Inform the user about the completion of an installation process.
This message will be printed in the title bar of the installer
window.

Template
  (COMPLETE [done])

Parameters
  [done] - a number between 0 and 100 which means the amount
           of work, which is already done

Options

Result
  Type: NUMBER
  Returns the argument [done]

Note

Example

  (COMPLETE 75)  ; print, that 75% of the installation is done


See also
```

## 1.47 COPYFILES

```
Copy a number of files from a source to a destination. The Installer
shows the files and the user may select/deselect, which files of the
predefined files should be copied.

Template
  (COPYFILES [option]+ )

Parameters

Options
  PROMPT
  HELP       - tell the user what's going to happen
  SOURCE     - the name of the source file or directory (may be relative)
  DEST       - name of the destination file or directory (may be relative)
               Note: the destination directory will be created, if it does
               not exist
  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 ↩
     this
               pattern should be copied
  FILE       - Only copy files; by default the Installer will match and copy
```

```
                subdirectories
  INFOS       - switch to copy icons along with other files/directories.
  NOPOSITION - reset the position of every icon copied.
  FONTS       - switch to not display ".font" files, yet still copy any that ↩
     match
                a directory that is being copied
  NOGAUGE     - don't display the status indicator
  OPTIONAL    - 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".
                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     - 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.

Result
  Type: NUMBER
  Return 0

Note
  - the options ALL/CHOISES/PATTERN are mutually exclusive
  - PATTERN only accepts standard AmigaOS patterns

Example

  ; just copy files beginning with "C" or "F" to t:
  (COPYFILES (SOURCE "c:")
             (DEST "T:")
             (PATTERN "(C#?|F#?)")
  )
```

## 1.48 COPYLIB

```
Copies only one file using version checking, i.e. it only overwrites the
destination file (if it exists) if the new file has a version/revision
higher than the existing file. If the destination directory does not
exist, it will be created.

Template
  (COPYLIB [option]+ )

Parameters
```

```
Options
  PROMPT
  HELP       - tell the user what's going to happen
  CONFIRM    - if this option is present, user will be prompted to confirm the
               copy operation, else the files will be copied silently. Note that
               an EXPERT user will be able to overwrite a newer file with an
               older one.
  SOURCE     - the name of the source file(may be relative)
  DEST       - name of the destination directory (may be relative)
  NEWNAME    - if the file is to be renamed, this is the new name
  INFOS      - switch to copy icons along with other files/directories.
  NOPOSITION - reset the position of every icon copied.
  NOGAUGE    - don't display the status indicator
  OPTIONAL   - 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".
               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    - removes options set by "optional"
  SAFE       - copy the file even if in PRETEND mode.

Result

Note
  - the destination directory will be created, if it does not exist

Example

  (COPYLIB (SOURCE "libs/mylib.library_020")
           (DEST "libs:")
           (NEWNAME "mylib.library")
  )
```

## 1.49 DATABASE

```
Returns information about the AMIGA that the InstallerNG is running on. The  ↩
   second
argument [checkvalue] is meant to be optional. If you do not use this argument,  ↩
   DATABASE
always returns a string with the result (see below for valid results). When  ↩
   using the
[checkvalue], then InstallerNG returns a number which is either 0 or 1.

Template
  (DATABASE [feature] [checkvalue]{0-1} )

Parameters
```

```
  [feature]    This string argument describes the information you are looking
               for. Valid features are:
                 "CPU"            - which type of CPU
                                    ("68000", "68010", "68020", "68030",  ↩
                                       "68040", "68060")
                 "PPC" {NG}       - checks for PPC; returns "PPC" if there is a  ↩
                    PPC installed,
                                    "" otherwise
                 "FPU"            - which type of FPU ("68881", "68882", "FPU040 ↩
                    ", "FPU060")
                 "MMU" {NG}       - which type of MMU ("68851", "MMU040", " ↩
                    MMU060")
                 "OS-VER" {NG}    - the version of exec (e.g. "40")
                 "GRAPHICS-MEM"   - amount of free chip memory
                 "FAST-MEM" {NG}  - amount of free fast memory
                 "TOTAL-MEM"      - total free memory
                 "CHIPREV"        - the revision of the graphic chipset
                                    ("AA", "ECS", "AGNUS")
                 "GFXSYSTEM" {NG} - the installed graphics system
                                    ("CyberGraphics", "Picasso96")
                 "DATE" {NG}      - the current date of your computer
                 "TIME" {NG}      - the current time of your computer
                 "GUI" {NG}       - type of the used GUI

  [checkvalue] optional; when given, this has to be a string. After evaluating  ↩
     the
               [feature], the result-string is compared to [checkvalue]. If this
               comparison matches, then DATABASE returns the number 1, otherwise ↩
                   the
               number 0

Options

Result
  the only parameter is [feature]
     a string containing the requested information or "unknown" if [feature] is  ↩
        an
     illegal string

  both parameters [feature] and [checkvalue] specified
     a number; 1 if [checkvalue] matches the result of [feature], otherwise 0

Note
  - InstallerNG accepts patterns for the [checkvalue] string, which will not  ↩
     work
    with the C= installer

Example

  (DATABASE "cpu")                ; e.g.  "68060"
  (DATABASE "bla")                ; "unknown"
  (DATABASE "cpu" "68000")        ; 1 iff you run on a 68000, otherwise 0

  ; this worx on every installer!!!
  (IF @installer-ng-version
      (
        (DATABASE "cpu" "(68040|68060)")
```

```
      )
      (
        (PATMATCH "(68040|68060)" (DATABASE "cpu"))
      )
  )                                ; 1 iff you run on a 68040 or 68060,  ↩
     otherwise 0
```

See also

## 1.50 DEBUG

```
Print anything to the InstallerNG-DEBUG console. You can supress
this output with switching off the "Show debug" option or by not
setting the DEBUGMODE shell-argument/tooltype.

Template
  (DEBUG [debuginfo]+ )

Parameters
  [debuginfo] - this can be anything: a number, a string, an expression.
                DEBUG prints the evaluation-result of [debuginfo] to the
                console window, followed by a linefeed.

Options

Result
  Type: STRING
  The result of the last [debuginfo] - evaluation

Note
  - if [debuginfo] is an uninitialized variable, then DEBUG prints
    an "<NIL>" to warn the user

Example

  (SET a 0)
  (DEBUG 1 "does not equal" a b)


See also
```

## 1.51 DELAY

```
{NG}
Sometimes it is useful to wait a specific time. Use the DELAY function for this
purpose.

Template
  (DELAY [ticks])

Parameters
```

```
  [ticks] - a number whichs defines the ticks. A tick is 1/50 second.

Options

Result
  Type: NUMBER
  Returns the [ticks]

Note

Example

  (DELAY 50)  ; wait a second


See also
```

## 1.52 DELETE

```
Delete a specific file

Template
  (DELETE [file] [options]* )

Parameters
  [file] - the path and name of the file, which has to be deleted (may be ↩
     relative)

Options
  PROMPT
  HELP      - tell the user what's going to happen.
  CONFIRM   - if this option is present, the user will be asked for confirmation ↩
     ,
              otherwise the delete proceeds silently
  OPTIONAL  - should deletions be forced.  options:
              FORCE: unprotect destination
              ASKUSER: ask user if the file should be unprotected (but not in  ↩
                 novice
                mode) In the case of ASKUSER, the default for "Novice" mode is  ↩
                   an
               answer of "No". Therefore, you may want to use FORCE to make
               the novice mode default answer appear to be "Yes"
  DELOPTS   - removes options set by OPTIONAL
  SAFE      - delete even if in "Pretend" mode
  INFOS     - also delete corresponding info file. Do not use this option
              together with ALL
  ALL       - check all matching subdirectories, too

Result
  Type: NUMBER
  Returns 0

Note
  - you are allowed to specify an AmigaOS pattern for the [file] parameter and
    by setting the ALL option, DELETE will delete all matching entries
```

```
Example

  ; deletes all libraries from LIBS:, which are named bla....library
  (DELETE "libs:bla#?.library"
          (SAFE)
          (ALl)
  )


See also
```

## 1.53 DIV

```
Divide a number by an oter one

Template
  (/ [value1] [value2])

Parameters
  [value1]
  [value2] - both values

Options

Result
  Type: NUMVER
  Returns the result of value1/value2

Note

Example
```

## 1.54 EARLIER

```
Check, whether a file is "younger" than another file

Template
  (EARLIER [file1] [file2])

Parameters
  [file1]
  [file2] - the files

Options

Result
  Type: NUMBER
  Returns 1 if [file1] is earlier than [file2]; 0 otherwise

Note
```

```
Example
```

## 1.55 EFFECT

```
If the script contains an EFFECT function, then this function will be
executed before any other function. EFFECT opens a new screen (with same
properties as the "Workbench") and forces the Installer to work on
that new screen. Additionally, you can define simple grafix effects on
this screen

Template
  (EFFECT [position] [effect] [color1] [color2])

Parameters
  [position] - moves the Installer window to a special position; valid
               strings are - "upper_left"
                           - "upper_center"
                           - "upper_right"
                           - "center_left"
                           - "center"
                           - "center_right"
                           - "lower_left"
                           - "lower_center"
                           - "lower_right"
  [effect]   - specify the effect for the screens background; valid
               strings are - "horizontal" (fades with horizontal lines)
                           - "radial" (fade with circles)
  [color1]
  [color2]   - set the fading colors; both are NUMBERS and specify the
               24 bit RGB value

Options

Result

Note
  - using an own screen makes it impossible to use InstallerNG's drag-n-drop
    features
  - the effect "radial" only works on true-color screens and falls back to
    "horizontal" on non-true-color screens

Example

  ; fade from black to white
  (EFFECT "center" "horizontal" $00000000 $00ffffff)


See also
```

## 1.56 EQU

```
                Checks, whether an expression equals an other expression

Template
  (= [value1] [value2])

Parameters
  [value1]
  [value2] - the values to be compared

Options

Result
  Type: NUMBER
  Returns 1 if both values are equal, 0 otherwise

Note
  - see
             COMPARE
                 - causes a "type conflict" error, if both types are not equal  ↩
                    and
    when they were not convertable

Example
  see
             COMPARE
               See also
```

## 1.57 EXECUTE

```
Execute an AmigaDOS script with the given arguments

Template
  (EXECUTE [script] [args]* [option]* )

Parameters
  [script] - the script, which has to be executed
  [args]   - the arguments for the script

Options
  PROMPT
  HELP    - tell the user what's going to happen
  CONFIRM - ask the user for confirmation
  SAFE    - execute even in "Pretend" mode

Result
  Type: NUMBER
  Returns the return value of the script

Note
  - the secondary result will be stored in the variable @ioerr

Example
```

## 1.58 EXIT

```
                Causes a normal termination of a script. The
               ONERROR
                functions are not
evaluated.

Template
  (EXIT [message]* [option])

Parameters
  [message] - these strings are concatenated and displayed as the final report

Options
  QUIET - skip the final message

Result
  Type: NUMBER
  Returns 0

Note

Example
```

## 1.59 EXISTS

```
Checks if a given path is valid or not. The result is a number, which describes
the type of the path.

Template
  (EXISTS [path] [option]* )

Parameters
  [path] - this string is the object, which has to be examined, e.g. "s:blurp"

Options
  NOREQ - when specified, then no requester will pop up, if [path] is not on an
          mounted volume

Result {NG}
  Type: NUMBER
  Returns  0 - [path] does not exist
           1 - [path] is a file
           2 - [path] is a directory
           3 - [path] is a link to a file
           4 - [path] is a link to a directory

Note
  The old Installer just returns either 0 or not 0

Example

  (EXISTS "s:startup-sequence")     ; should be 1
```

```
  (EXISTS "C:")                     ; should be 2
  (EXISTS "grfm:hlbzs/hsjs")        ; maybe 0
```

See also

## 1.60 EXPANDPATH

Get the full path of a file or a logical assign

Template

```
  (EXPANDPATH [path])
```

Parameters

[path] - the path which should be expanded

Options

Result

Type: STRING
Returns the full path of [path]

Note

Example

```
  (EXPANDPATH "c:") ; may deliver "System:C" or whatever
```

See also

## 1.61 FILEONLY

Returns the file part (i.e. the last path component) of a given path

Template

```
  (FILEPART [path])
```

Parameters

[path] - the path

Options

Result

Type: STRING
Returns the file part of the [path]

Note

Example

```
See also

               PATHONLY
```

## 1.62 FINDBOARD

```
{NG}
This functions makes you able to find a specific hardware expansion board in
the system.

Template
  (FINDBOARD [manufacturer] [product])

Parameters
  [manufacturer] - the manufacturer id of the board. this id is unique for
                   every (registered!) hardware producer and is assigned by C=
  [product]      - the number of the product of a specific manufacturer.

Options

Result
  Type: NUMBER
  Returns the number of found boards

Note
  To get a list of valid manufacturers and their products, please have a look
  at the "board.library" package or related tools like "ShowBoardsMUI" by
  Torsten Bach

Example

  (SET #boardcount (FINDBOARD 8512 67))  ; how many CV64/3D gfx-cards has the  ↩
     system?


See also
```

## 1.63 FLUSHLIBS

```
{NG}
Removes every actually unused library from the system. This can be useful if
you install new libraries and want to use them without rebooting the entire
system.

Template
  (FLUSHLIBS)

Parameters

Options
```

```
Result
  Type: NUMBER
  Returns zero

Note
  To flush a system shared library, it must not be used while the remove process ↩
     !

Example

  ; now remove unused libraries from the system
  (FLUSHLIBS)


See also
```

## 1.64 FOREACH

```
For each file of a directory, which matches a given pattern,
a sequence of functions will be executed. The variables @each-name
and @each-type will hold the name and the AmigaDOS object type
(file/directory) for each of the matching files.

Template
  (FOREACH [dir] [pattern] [fun]+)

Parameters
  [dir]     - the directory which will be used for the walk
  [pattern] - an AmigaDOS pattern, which specifies the files, for which
              some functions will be executed
  [fun]     - functions for matching files; these functions should make
              use of the variables @each-name and @each-type

Options

Result
  Type: NUMBER
  Return 0

Note
  - @each-type is less than zero for files; greater than zero for directories

Example

  ; recursively print a directory tree of a given dir
  (PROCEDURE P_PrintDirTree #d
             (LET (SET #dir #d)
                  (FOREACH #dir "#?"
                           (
                             (DEBUG @each-name)
                             (IF (> @each-type 0) (P_PrintDirTree (tackon #dir  ↩
                                @each-name)))
                           )
                  )
             )
```

```
  )

See also
```

## 1.65 GE

```
                Checks, whether an expression is greater or equal to an other  ↩
                   expression

Template
  (>= [value1] [value2])

Parameters
  [value1]
  [value2] - the values to be compared

Options

Result
  Type: NUMBER
  Returns 1 if [value1] is greater or equal than [value2], 0 otherwise

Note
  - see
             COMPARE
                 - causes a "type conflict" error, if both types are not equal  ↩
                    and
    when they were not convertable

Example
  see
             COMPARE
               See also
```

## 1.66 GETASSIGN

```
Returns the pathname of an AmigaDOS object.

Template
  (GETASSIGN [name] [spec]{0-1} )

Parameters
  [name] - the name of the object, for which the path should be found
  [spec] - specifies, in which list the installer should search; valid
           strings are: - "a" for the list of assigns (default)
                        - "v" for the volume list
                        - "d" for the device list
           Note that you are allowed to specify several lists by appendig
           the specification characters (i.e. if you want to search all
           three list, simply write "adv")
```

```
Options

Result
  Type: STRING
  Returns the pathname or an empty string, if the pathname could not
  be found

Note
  - without setting a [spec], only the assign list will be checked
  - [name] must be specified without colons; i.e. instead of "ENV:" you
    must write "ENV"

Example

See also
```

## 1.67 GETDEVICE

```
Returns the name of the device, on which a given path resides

Template
  (GETDEVICE [path])

Parameters
  [path] - the path, for which the device name should be found

Options

Result
  Type: STRING
  Returns the device name

Note
  - the device name comes without colons

Example

  ; find out, on which device the mountlist resides
  (GETDEVICE "devs:mountlist") ; may return "System"


See also
```

## 1.68 GETDISKSPACE

```
Returns the available free diskspace in bytes on the disk given by
a path.

New for v44+
Additionally, you may specify the unit for the calculated space.
Typical hard drives are larger than 4 G today and partitions may
```

```
also be larger than 2 G. Older versions of Installer returns illegal
values for partitions larger than 2 GB. The new installer returns the
maximum integer (2147483647) if the partition is too large.

Template
  (GETDISKSPACE [path] [unit]{0-1} )

Parameters
  [path] - the path, which specifies device
  [unit] - optional and defines the unit for the returned disk space:
           "B" (or omitted) is "Bytes", "K" is "Kilobytes", "M" is "Megabytes"
           and "G" is "Gigabytes"

Options

Result
  Type: NUMBER
  Returns the free disk space or -1, if [path] was illegal

Note
  - you should use at least unit "K" in new installer scripts to
    avoid overflows with large harddrives.

Example

See also
```

## 1.69 GETENV

```
Returns the content of a environment variable, which is usually
located in the "ENV:" drawer

Template
  (GETENV [varname])

Parameters
  [varname] - the name of the variable

Options

Result
  Type: STRING
  Returns the content of the variable

Note
  - currently, the content is limited to 64 bytes, which should be
    enough in most cases
  - binary data are not supported

Example

See also
```

## 1.70 GET-PROPERTY

```
                {NG}
Read a specific property of a symbol.

Template
  (GET-PROPERTY <symbol> <property>)

Parameters
  <symbol>   - the target symbol
  <property> - the desired property of the symbol

Options

Result
  Type: depends on the propertys type
  Returns the value of the property

Note
  Raises an error, if the property does not exist.

Example

  (SET #bla "savage is cool :-)")      ; declare a symbol #bla
  (READ-PROPERTY-OBJECT #bla)          ; try to read saved properties
  (PUT-PROPERTY #bla "property" 20)    ; add property "property" to the symbol # ↩
     bla
  (MESSAGE                             ; get the value of #bla`s property " ↩
     property"
    (GET-PROPERTY #bla "property")
  )
  (SAVE-PROPERTY-OBJECT #bla)          ; save the properties of #bla
  (REMOVE-PROPERTY #bla "property")    ; remove "property" from #bla


See also

              PUT-PROPERTY

              REMOVE-PROPERTY

              READ-PROPERTY-OBJECT

              SAVE-PROPERTY-OBJECT
```

## 1.71 GETSIZE

```
Returns the size of a file in bytes

Template
  (GETSIZE [filename])

Parameters
```

```
    [filename] - the path and name of the file

  Options

  Result
    Type: NUMBER
    Returns the size of the file

  Note

  Example

  See also
```

## 1.72 GETSUM

```
  Calculate a checksum for a file. This could be used for checking
  version or if the content of files differs

  Template
    (GETSUM [filename])

  Parameters
    [filename] - the name of the file, for which you wanna calc the checksum

  Options

  Result
    Result: NUMBER
    Returns the checksum for a file

  Note
    - use the "GetSum" shell command (provided with the InstallerNG package)
      to calculate checksums for files from a shell

  Example

  See also
```

## 1.73 GETVERSION

```
  This returns the version of a file. The file must have a valid RomTag
  structure or a valid AmigaDOS 2.x version string. If you do not provide
  the filename, this simply returns the version of the OS.

  Template
    (GETVERSION [name]{0-1} [option]* )

  Parameters
    [name] - the file, for shich you need the version

  Options
```

```
  RESIDENT - specifying this, causes the Installer to search the systems
             library and device lists for the [name] entry

Result
  Type: NUMBER
  Returns the version of the file or OS; returns 0 if [name] was invalid

Note
  The result is a 32 bit value; the upper 16 bits contain the version
  and the lower 16 bits the revision.

Example

  (GETVERSION) ; returns the version of the OS
  (GETVERSION "c:dir") ; returns DIR's version
  (GETVERSION "dos.library" (RESIDENT)) ; returns the version of the dos.library

  ; this function converts a version number to a readable string
  (PROCEDURE version-to-string #ver ("%ld.%ld" (/ #ver 65536) (BITAND #ver  ↩
     65535)))
  (MESSAGE "OS version of your Amiga: " (version-to-string (GETVERSION)) " !!!")


See also
```

## 1.74 GT

```
                Checks, whether an expression is greater than an other  ↩
                   expression

Template
  (> [value1] [value2])

Parameters
  [value1]
  [value2] - the values to be compared

Options

Result
  Type: NUMBER
  Returns 1 if [value1] is greater than [value2], 0 otherwise

Note
  - see
              COMPARE
                  - causes a "type conflict" error, if both types are not equal  ↩
                     and
    when they were not convertable

Example
  see
              COMPARE
                See also
```

## 1.75 ICONINFO

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

Template
  (ICONINFO [option]+ )

Parameters

Options
  PROMPT
  HELP           - tell the user what's going to happen.
  DEST           - the name of the icon to be modified. There is no need to  ↩
     specify a
                   ".info" extension.
  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
  GETTOOLTYPE    - the tooltype name and result symbol name string.
  GETDEFAULTTOOL - symbol name for the default tool name of a project.
  GETSTACK       - symbol name for the current stack size of the icon.
  GETPOSITION    - Two symbol names for the saved icon position in X and Y  ↩
     direction.
                   Do not use this lightly. It is intended to keep icon
                   positions on updates with help of the parameter SETPOSITION
                   of the TOOLTYPE function. Arbitrarily changing icon positions
                   will lead to annoyed users due to different Workbench and  ↩
                      font
                   setups.  If the icon doesn't have a position set, -1 is  ↩
                      returned
                   for the respective position value. This may be passed to
                   TOOLTYPE

Result
  Type: NUMBER
  Returns 0

Note

Example

  ; show the initial size of the stack of the InstallerNG
  (ICONINFO (DEST "C:InstallerNG")
            (GETSTACK "stack")
  )
  (MESSAGE stack)
```

```
See also
```

## 1.76 IF

```
Conditionally execute functions. If [condition] is TRUE (i.e. not 0) then
the [then] will be executed, otherwise [else]

Template
  (IF [condition] [then] [else] )

Parameters
  [condition] - any expression
  [then]      - functions which are executed if [condition] is TRUE
  [else]      - functions which are executed if [condition] is FALSE

Options

Result
  Type: depends
  Returns the result of [then] or [else]

Note

Example

  (IF (= 2 4)               ; condition
      (MESSAGE "TRUE")      ; then
      (                     ; else
        (MESSAGE "FALSE")
        (BEEP)
      )
  )


See also
```

## 1.77 IN

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

Template
  (IN  [value] [bitnum]+ )

Parameters
  [value]  - the value to be checked
  [bitnum] - the numbers of bits, which are checked, whether they are
             set in [value] or not

Options

Result
```

```
  Type: NUMBER
  Returns 0 or a mask

Note

Example

See also
```

## 1.78 LE

```
                Checks, whether an expression is less or equal to an other  ↩
                   expression

Template
  (<= [value1] [value2])

Parameters
  [value1]
  [value2] - the values to be compared

Options

Result
  Type: NUMBER
  Returns 1 if [value1] is less or equal than [value2], 0 otherwise

Note
  - see
             COMPARE
                 - causes a "type conflict" error, if both types are not equal  ↩
                    and
    when they were not convertable

Example
  see
             COMPARE
               See also
```

## 1.79 LET

```
{NG}
This function creates a new environment. This means, you can declare new  ↩
   variables
within the <init> functions and use them in the <body> functions. If you define
local variables, which have the same name like existing ones, you "replace" the  ↩
   existing
by the local variables. Nevertheless you can access existing variables, which  ↩
   are
not overwritten.
Imagine of the new environment as a layer, which overwrites variables with the  ↩
   same name
```

```
but keeps all other variables.
Put this function as the first into a PROCEDURE definition and write the body of ↩
    the
PROCEDURE as the body of the LET function! Now you have private variables for  ↩
   the
procedure :)

Template
  (LET <init> <body> )

Parameters
  <init> - one function, which initializes the local environment. It does not  ↩
     make
           sense to use other functions than SET here
  <body> - the body of a LET function are the functions, which use this local
           environment

Options

Result
  LET returns the result of the last function of <body>

Note
  Since LET is a simple function, you can create LET environments inside of LET
  environments inside of...

Example

  ; this "creates" the value 7 by adding values of the  local environment
  (LET (SET x 3 y 4)
       (+ x y)
  )

  ; a procedure with local variables
  (PROCEDURE P_bla #arg1 #arg2
    (LET (SET #local_x #arg1
              #local_y #arg2
         )

         (
            ; do anything with #local_x and #local_y
         )
    )
  )
```

## 1.80 LT

```
                Checks, whether an expression is less than an other expression

Template
  (< [value1] [value2])

Parameters
  [value1]
```

```
    [value2] - the values to be compared

  Options

  Result
    Type: NUMBER
    Returns 1 if [value1] is less than [value2], 0 otherwise

  Note
    - see
                COMPARE
                    - causes a "type conflict" error, if both types are not equal  ↩
                       and
      when they were not convertable

  Example
    see
                COMPARE
                  See also
```

## 1.81 MAKEASSIGN

```
  Assigns an assign to a path or removes a specific assign.

  Template
    (MAKEASSIGN [assign] [path]{0-1} [option]* )

  Parameters
    [assign] - the name for the assign
    [path]   - optional; the path, which should be assigned to [assign]

  Options
    SAFE - if specified, the assign will be created even in "Pretend" mode

  Result
    Type: NUMBER
    Returns 0

  Note
    - omit [path] to clear the assign

  Example

  See also
```

## 1.82 MAKEDIR

```
  Just create a new directory. Furthermore you are allowed to
  specify a complete path and the Installer will create all the
  necessary sub-directories
```

```
  Template
    (MAKEDIR [name] [option]* )

  Parameters
    [name] - the name or path of the directory

  Options
    PROMPT
    HELP     - tell the user what's going to happen
    CONFIRM  - ask for confirmation; otherwise the directory will be
               created silently
    INFOS    - also create an icon for the drawer
    SAFE     - make the directory even in "Pretend" mode

  Result
    Type: NUMBER
    Returns 0

  Note

  Example

  See also
```

## 1.83 MESSAGE

```
 ~Display some text to the user

  Template
    (MESSAGE [string]* [option]* )

  Parameters
    [string] - the strings, which, all concatenated into one single string,
               gets displayed

  Options
    ALL - Show the text also to "Novice" users

  Result
    Type: STRING
    Returns the displayed text

  Note
    - in "Novice" mode, the text will not be shown as long as you do not
      specify the ALL option

  Example

  See also
```

## 1.84 MUL

```
Multiply some values

Template
  (* [value]* )

Parameters
  [value] - all the values, from which the result will  be calculated

Options

Result
  Type: NUMBER
  Returns the product of the multiplication

Note

Example

See also
```

## 1.85 NE

```
                Checks, whether an expression is equals an other expression

Template
  (<> [value1] [value2])

Parameters
  [value1]
  [value2] - the values to be compared

Options

Result
  Type: NUMBER
  Returns 1 if [value1] does not equal [value2], 0 otherwise

Note
  - see
              COMPARE
                  - causes a "type conflict" error, if both types are not equal  ↩
                     and
    when they were not convertable

Example
  see
              COMPARE
                See also
```

## 1.86 NOP

```
{NG}
Does nothing...
Since my language definition does not allow empty function-lists,
I thought it would be useful to have a NOP function for this case :)
Additional, if you give an expressin as argument, NOP evaluates this
expression and returns it's value instead of 0 (weird...)

Template
  (NOP [expr]{0-1})

Parameters
  expr - if given, NOP evaluates this expression

Options

Result
  Type: depends on [expr] or NUMBER (if no [expr] is given)
  Returns [expr] or 0 (if no [expr] is given)

Note
  You may ask, why i did this. The reason is simple: I needed something
  like a "return (x)" in a PROCEDURE...

Example

  ; if @bla equals 5, then do nothing, else beep and pop a message
  (IF (= @bla 5)
      (NOP) {NG}
      (
        (BEEP) {NG}
        (MESSAGE "@bla does not equal 5")
      )
  )


See also
```

## 1.87 NOT

```
Negates the boolean value of an expression

Template
  (NOT [value])

Parameters
  [value] - the value to be negated

Options

Result
  Type: NUMBER
  Returns the boolean "Not" of the value
```

Note

Example

See also

## 1.88 ONERROR

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.

Template
(ONERROR [fun]* )

Parameters
[fun] - some functions, which will be executed in case of not trapped errors

Options

Result
Type: NUMBER
Returns 0

Note

Example

```
; execute this in case of an error
(ONERROR (BEEP) {NG}
         (DEBUG "bad error!!!")
         (EXIT (QUIET))
)
```

See also

TRAP

## 1.89 OR

The logical "or", i.e. OR deliveres true if at least one of its arguments is true.
OR stops the evaluation with the first true-argument

Template
(OR [value]+ )

```
Parameters
  [value] - the value which should logically be tested

Options

Result
  Type: NUMBER
  Returns 1 for true and 0 for false

Note

Example

See also
```

## 1.90 OPENWBOBJECT

```
                Open a workbench object, which can be either a disk, a drawer, a ↩
                    trashcan, a
tool or a project

Template
  (OPENWBOBJECT [wbobject] [option]* )

Parameters
  [wbobject] - the object to be opened

Options
  PROMPT
  HELP     - tell the user what's going to happen
  CONFIRM  - ask for confirmation
  SAFE     - open the [wbobject] even in "Pretend" mode

Result
  Type: NUMBER
  Returns 1 if OPENWBOBJECT succeeded, 0 if the [wbobject] could
  not be found, or -1 if the machine (i.e. the Workbench) does not
  support this function

Note

Example

  ; open the "SYS:" drawer
  (OPENWBOBJECT "sys:" (PROMPT "now opening the \"sys:\" drawer...")
                       (HELP "...")
                       (CONFIRM expert) {NG}
  )

  ; scroll to or make the "Prefs" drawer visible
  (SHOWWBOBJECT "sys:prefs")

  ; close the system drawer
  (CLOSEWBOBJECT "sys:")
```

```
  ; run the multiview, as the user had double-clicked its icon
  (OPENWBOBJECT "sys:tools/multiview" (PROMPT "simulating a workbench start for ↩
     multiview")
                                      (HELP "...")
                                      (CONFIRM novice) {NG}
  )
```

See also
  see
SHOWWBOBJECT
,
OPENWBOBJECT

## 1.91 PATHONLY

Returns the non-file part of a given path by extracting the last component (the file part) from the path

Template
  (PATHONLY [path])

Parameters
  [path] - the path, from which you need the path part

Options

Result
  Type: STRInG
  Returns the path part of [path]

Note

Example

See also

FILEONLY

## 1.92 PATMATCH

Determines, if a given string matches an AmigaDOS pattern or not. The pattern has to fulfill the conventions for patterns of the AmigaDOS!

Template
  (PATMATCH [pattern] [string])

Parameters
  [pattern] - an AmigaDOS pattern

```
    [string]  - the string, which gets matched against the [pattern]

  Options

  Result
    Type: NUMBER
    Returns 1 if the string matches the pattern, 0 otherwise

  Note

  Example

  See also
```

## 1.93 PROCEDURE

```
                  Using this function, you can define your own functions.

  Template
    (PROCEDURE [name] [params]* [fun]+ )

  Parameters
    [name]   - an dentifier, which defines the name of the procedure
    [params] - a list of parameters for the function
    [fun]    - the body of the function

  Options

  Result
    (internally, this does not execute like other functions and, thus,
    has not return value/type)

  Note
    - for future compatibility, please name your functions starting with
      "P_" as prefix, so that collisions with new functions will be avoided
    - PROCEDURE should be named "DEF-FUNCTION" or something similar, but
      for compatibility I kept the name for the InstallerNG

  Example
    see
                GETVERSION
                 or
                FOREACH
                 for sample functions

  See also
```

## 1.94 PROTECT

```
  Either get or set the protection values of a given file or
  directory. You can define the protection mask by a string or by
```

```
a number mask.

Template
  (PROTECT [file] [mask]{0-1} [option])

Parameters
  [file] - the file for this operation
  [mask] - optional; either a string or a decimal number, which specifies the
           mask of bits for the file. The bits and the related characters
           are defined as follows

             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
             +---------------- /


Options
  SAFE - change protection even in "Pretend" mode

Result
  Type: NUMBER
  Returns 0 (for failure) or 1 (for success) if you changed the protection
  bits of a file or, if you want to read a protection (in this case, omit
  the [mask] argument), returns the mask of protection bits of the given file

Note
  - this follows the AmigaOS rules for protection bits
  - you must not use the "H" bit since this is currently not supported
    by the AmigaOS

Example

See also
```

## 1.95 PUT-PROPERTY

```
                {NG}
Bind a property to a symbol. Imagine of a "property" as an attribut,
i.e. a property-name and a related value. If the property already exists,
its value just gets updated.

Template
```

```
    (PUT-PROPERTY <symbol> <property> <value>)

  Parameters
    <symbol>   - the target symbol
    <property> - the property you wish to create or modify
    <value>    - the (new) value of the property

  Options

  Result
    Type: depends on the type of <value>
    Returns <value>

  Note
    If the <property> for the <symbol> already exists, the value of
    the property will be changed to <value>

  Example
    see
                GET-PROPERTY
                  See also

                GET-PROPERTY

                REMOVE-PROPERTY

                READ-PROPERTY-OBJECT

                SAVE-PROPERTY-OBJECT
```

## 1.96 QUERYDISPLAY

```
 ~Returns information about the current display of the Installer window

  Template
    (QUERYDISPLAY [object] [attribute])

  Parameters
    [object]    - the target object, must be "screen" or "window"
    [attribute] - the attribute of the object, which has to be examined;
                  valid attributes are for
                  windows: "width", "height", "upper", "lower" "left" "right"
                  screens: "width", "height", "depth", "colors"

  Options

  Result
    Type: NUMBER
    Returns the value of the attribute

  Note

  Example
```

```
See also
```

## 1.97 RANDOM

```
{NG}
This results in a random number, which ranges in given bounds

Template
  (RANDOM <lower> <upper>)

Parameters
  <lower>
  <upper> - the numbers which specify the range, where the result ranges in

Options

Result
  Type: NUMBER
  Returns a random number from <lower> ... <upper>

Note

Example

  (RANDOM 20 50) ; give a number between 20 and 50


See also
```

## 1.98 READ-PROPERTY-OBJECT

```
                {NG}
Read all the properties of an object. See
              SAVE-PROPERTY-OBJECT
                for the construction of the filename, where the InstallerNG
stores the data.

Template
  (READ-PROPERTY-OBJECT <ident>)

Parameters
  <ident> - the identifier, for which the InstallerNG should
            read the properties

Options

Result
  Type: NUMBER
  Returns 0

Note
  The InstallerNG reads only properties of type STRING. If you
```

```
  need a NUMBER value and you are not sure, whether or not the
  InstallerNG imlicitly casts to NUMBER, you should use the

              CAST-INT
               function, to convert a STRING value into a NUMBER value.

Example
  see
              GET-PROPERTY
                See also

              GET-PROPERTY

              PUT-PROPERTY

              REMOVE-PROPERTY

              SAVE-PROPERTY-OBJECT
```

## 1.99 REBOOT

```
{NG}
This function causes a reboot of your Amiga. Several scripts may need this to
mount new drivers to the system. Be careful with this ;)

Template
  (REBOOT <options>)

Parameters

Options
  (SAFE) - specifying this will cause a reboot even in pretend mode

Result  :)
  Type: NUMBER
  Returns 0

Note

Example

  (REBOOT)         ; reboots, but not in pretend mode
  (REBOOT (SAFE))  ; always reboot


See also
```

## 1.100 REMOVE-PROPERTY

```
                {NG}
Remove a specific property from a symbol. This does really remove the property
```

```
itself, not only a reset of the property!

Template
  (REMOVE-PROPERTY <symbol> <property>)

Parameters
  <symbol>   - the target symbol
  <property> - the property you wish to remove

Options

Result
  Type: NUMBER
  Returns 0

Note
  Raises an error, if the property does not exist.

Example
  see
              GET-PROPERTY
                See also

              GET-PROPERTY

              PUT-PROPERTY

              READ-PROPERTY-OBJECT

              SAVE-PROPERTY-OBJECT
```

## 1.101 RENAME

```
Rename a file/directory or a disk.

Template
  (RENAME [oldname] [newname] [option* )

Parameters
  [oldname] - the source file/directory or the disk to be renamed; in
              case of an disk, the name must contain a colon (e.g. "DF0:")
  [newname] - new name; in case of a disk, the new name must NOT contain
              the colon

Options
  PROMPT
  HELP     - tell the user what's going to happen
  CONFIRM  - if this is present, then the user will be asked for confirmation
  DISK     - you must specify this, if you want to rename a disk
             (called "relabeling)
  SAFE     - rename even in "Pretend" mode

Result
  Type: NUMBER
```

```
  Returns 0

Note

Example

See also
```

## 1.102 RETRACE

```
                This will skip the last evaluated TRACE an does continue to work ↩
                    at
the previous one. A tracepoint gets lost if the evaluation leaves the
scope of the TRACE.

Template
  (RETACE)

Parameters

Options

Result
  Type: NUMBER
  Returns 0

Note

Example

  (TRACE) ; set the first tracepoint
  (MESSAGE "now followes an IF")
  (IF (= 1 1)
      (
        (TRACE) ; this tracepoint gets lost when the installer
                ; leaves this then-block!
        (MESSAGE "then")
      )
      (MESSAGE "else")
  )
  (TRACE)
  (RETACE)


  This will result in an infinite loop, because the inner TRACE (situated
  in the then-block) gets lost and RETRACE skips to the trailing TRACE

See also

              TRACE
```

## 1.103 REXX

```
Executes a given ARexx script with the given arguments.

Template
  (REXX [script] [arg]* [option]* )

Parameters
  [script] - the script to be executed
  [arg]    - arguments for the script

Options
  PROMPT
  HELP    - tell the user what's going to happen
  CONFIRM - if specified, the user will be asked for confirmation
  SAFE    - execute, even in "Pretend" mode

Result
  Type: NUMBER
  Returns the primary result of the script and stores the secondary
  result in @ioerr

Note
  - this needs an active ARexx server

Example

See also
```

## 1.104 RUN

```
Executes a binary programm with the given parameters

Template
  (RUN [command] [arg]* [option]* )

Parameters
  [command] - the command, which should be executed
  [arg]     - arguments for the command

Options
  PROMPT
  HELP    - tell the user what's going to happen
  CONFIRM - if specified, the user will be asked for confirmation
  SAFE    - run command, even in "Pretend" mode

Result
  Type: NUMBER
  Returns the primary result of the command and stores the secondary
  result in @ioerr

Note

Example
```

## 1.105 SAVE-PROPERTY-OBJECT

```
                {NG}
Save the properties of an identifier to a file. The filename is
built by appending the name of the identifier to the application
name (i.e. the value of the builtin variable @APP-NAME). By saving
such a property list, you will get a file, which contains all the
properties in this format:

  propertyname=value
  propertyname=value
  ...

You may use this to store configurations or whatever by simply
using such property lists.

Template
  (SAVE-PROPERTY-LIST <ident>)

Parameters
  <ident> - an identifier, which properties should be saved to a file

Options

Result
  Type: NUMBER
  Returns 0

Note

Example
  see
              GET-PROPERTY
                See also

              GET-PROPERTY

              PUT-PROPERTY

              REMOVE-PROPERTY

              SAVE-PROPERTY-OBJECT
```

## 1.106 SELECT

```
Execute only one special out of more functions.

Template
  (SELECT [num] [fun]* )
```

```
Parameters
  [num] - specify the number of function, which should be evaluated
          (starting with zero)
  [fun] - several functions

Options

Result
  Type: depends
  Returns the result of the evaluated functions

Note

Example

  ; writes "zap"
  (SELECT 2 (MESSAGE "bla") (MESSAGE "burp") (MESSAGE "zap"))


See also
```

## 1.107 SET

```
Set a value to a variable. If this is the first setting, then this
value will be declared. Access to not initialized variables will
cause a runtime warning and will deliver the number zero.

Template
  (SET [[name] [value]]* )

Parameters
  [name]  - the name of the variable
  [value] - the (new) value for this variable

Options

Result
  Type: depends
  Returns the last setting

Note

Example

See also
```

## 1.108 SETENV

```
                {NG}
Sets a system variable. This is only temporary done in the ENV: directory
and the variable will be lost after a reset.
```

```
Template
  (SETENV <varname> <value>)

Parameters
  <varname> - a string which is the name of the variable
  <value>   - this string must contain the value for the variable

Options
  (RESIDENT) - specify this to write the variable to both, ENV: and
               ENVARC: directories (needs v39+ or AmigaOS3.0+)

Result
  Type: STRING
  Returns <value>

Note
  The variable is only temporary set to ENV:

Example

  (SET var "MY_TEMP_VARIABLE")
  (SETENV var "the value of my temp variable")


See also

              GETENV
```

## 1.109 SETMEDIA

```
                Modify properties of a media object.

Template
  (SETMEDIA [object] [action] [actionparam]{0-1} )

Parameters
  [object]      - the media object identifier; {NG} can be either a
                  string or an identifier of type STRING
  [action]      - the action, which has to be performed with the media object  ↩
     and
                  depends on the objects type; valid actions strings are
                    "pause"
                    "play"
                    "contents"
                    "index"
                    "retrace"
                    "browser_prev"
                    "browser_next"
                    "command"
                    "rewind"
                    "fastforward"
                    "stop"
                    "locate"
```

```
  [actionparam] - if [action] is "command" or "locate", then this will hold the  ↩
     command
                  string argument

Options

Result
  Type: NUMBER
  Returns 0

Note

Example
  see
              SHOWMEDIA
                See also

              SETMEDIA
              ,
              SHOWMEDIA
```

## 1.110 SHIFTLEFT

```
Bit oriented shifting of a value. Zeros are shifted in on the
opposite side.

Template
  (SHIFTLEFT [value] [shiftamount])

Parameters
  [value]       - the value to be shifted
  [shiftamount] - the amount of shifts

Options

Result
  Type: NUMBER
  The left-shifted [value]

Note

Example

See also
```

## 1.111 SHIFTRIGHT

```
Bit oriented shifting of a value. Zeros are shifted in on the
opposite side.

Template
```

```
  (SHIFTRIGHT [value] [shiftamount])

Parameters
  [value]       - the value to be shifted
  [shiftamount] - the amount of shifts

Options

Result
  Type: NUMBER
  The right-shifted [value]

Note

Example

See also
```

## 1.112 SHOWMEDIA

```
This opens a datatype object (you need at least AmigaOS 3.0 for this)
and presents it to the user. Depending on the type of the media object,
this function can open a custom window to show the file.

Template
  (SHOWMEDIA [name] [file] [position] [size] [borderflag] [attr]* )

Parameters
  [name]       - a string, which specifies the name for this media object; this
                 will be used by SETMEDIA and CLOSEMEDIA functions later
  [file]       - the name of the file to show
  [position]   - if the media object needs a window (e.g. pictures or animations ↩
     )
                 this defines the (relative) size of the window; valid strings  ↩
                    are:
                   "upper_left"
                   "upper_center"
                   "upper_right"
                   "center_left"
                   "center"
                   "center_right"
                   "lower_left"
                   "lower_center"
                   "lower_right"
  [size ]      - if the media object needs a window, then this defines the
                 (relative) size of the window; valid strings are:
                   "none"
                   "small"
                   "small_medium"
                   "small_large"
                   "medium"
                   "medium_small"
                   "medium_large"
                   "large"
                   "large_small"
```

```
                    "large_medium"
  [borderflag] - if set to 1, then the window will have scrollers in its borders ↩
     ,
                  otherwise (i.e. if 0) it gets no borders
  [attr]       - some attributes, which specify the datatypes attributes, valid
                  strings are:
                    "wordwrap"
                    "panel"
                    "play"
                    "repeat"

Options

Result
  Type: NUMBER
  Returns 1 if the datatype could be opened, 0 otherwise

Note
  Every viewer runs as an own process, i.e. you may continue script execution,
  while the viewer process shows pictures, anims, AmigaGuides® and so on.

Example

See also
```

## 1.113 SHOWWBOBJECT

```
                Makes an arbitrary workbench object visible, i.e. it scrolls the ↩
                    view
of a workbench drawer, until the named object becomes visible

Template
  (SHOWWBOBJECT [wbobject] )

Parameters
  [wbobject] - the object to be viewed

Options

Result
  Type: NUMBER
  Returns 1, if SHOWWBOBJECT succeeded, 0 if the [wbobject] could
  not be found, or -1 if the machine (i.e. the Workbench) does not
  support this function

Note

Example
  see
              OPENWBOBJECT
                See also
  see
              SHOWWBOBJECT
              ,
              OPENWBOBJECT
```

## 1.114 SIMULATE-ERROR

```
                {NG}
A runtime error will be simulated. This is very useful for testing and
debugging scripts.

Template
  (SIMULATE-ERROR <error>)

Parameters
  <error> - a number value which ranges from 1 to 5. The meaning of the
            numbers are:  1 - Quit
                          2 - Out of mem
                          3 - Error in script
                          4 - DOS error (@ioerr is set to 236 ( ↩
                              ERROR_NOT_IMPLEMENTED))
                          5 - Bad parameter data
            every other number simulates the "Out of range" error.

Options

Result
  Type: NUMBER
  Returns <error>

Note
  The <error> argument numbers are the same as used by the
              TRAP
               function.

Example

  (ONERROR (
             (BEEP)
             (MESSAGE "Damn, an error!")
           )
  )
  (SIMULATE-ERROR 2)

  ...

  (SET #err (TRAP 3 (SIMULATE-ERROR 3)
            )
  )
  (IF (= #err 3) (MESSAGE "There was an error in the script..."))


See also

              ONERROR
              ,
              TRAP
```

## 1.115 STARTUP

```
Using this function, you can add commands to the users startup files.
First, the Installer tries to modify the "user-startup" and if this
failes, it creates a new "user-startup" and adds a call to this
"user-startup" file to the "startup-sequence" (but asks for confirmation
befor it writes to the "startup-sequence"). Old modifications of the
application will be replaced by these new ones.

Template
  (STARTUP [appname] [option]* )

Parameters
  [appname] - The Installer will comment the modifications by noting the
              name of the application, which caused the modifications; use
              the @app-name variable here

Options
  PROMPT
  HELP    - tell the user what's going to happen
  CONFIRM - if specified, the user will be asked for confirmation
  COMMAND - used to declare an AmigaDOS command line, which will be added to
            the startup script.

Result
  Type: NUMBER
  Returns 0

Note

Example

See also
```

## 1.116 STRLEN

```
Calculates the length of a given string, i.e. the number of characters

Template
  (STRLEN [string])

Parameters
  [string] - the string

Options

Result
  Type: NUMBER
  Returns the length of the string
```

Note

Example

## 1.117 SUB

Subtract all the parameters

Template
(- [value]+ )

Parameters
[value] - the values to be subtracted, starting with the first one

Options

Result
Type: NUMBER
Returns the result of this chain of subtractions

Note

Example

See also

## 1.118 SUBSTR

Returns a substring of a given string by extracting a part of the string

Template
(SUBSTR [string] [offset] [count]{0-1} )

Parameters
[string] - the original string
[offset] - number of the first character of the new substring
[count] - optional; the length of the new substring

Options

Result
Type: STRING
Returns the created substring

Note

Example

```
(SUBSTR "this is cool, isn't it?" 8 4) ; returns "cool"
```

```
See also
```

## 1.119 SWING

```
{NG}
This allows you to jump (inside of this block) from one function to its  ↩
    neighbour
function. Thus, you may use all the ASK... functions to set the installation
environment AND to have an undo/redo option

Template
  (SWING <stmt> ...)

Parameters
  <stmt> - one or more functions. SWING will jump between them

Options

Result
  Type: number
  Returns 0

Note

Example

  (SET number 5
       text "bla"
  )

  (SWING
    (SET number (ASKNUMBER (PROMPT "Enter a number")
                           (HELP "...")
                           (DEFAULT number)
                )
    )
    (SET text (ASKSTRING (PROMPT "Enter a text")
                         (HELP "...")
                         (DEFAULT text)
              )
    )
  )
```

## 1.120 TACKON

```
                  Add a component to an existing path. This function also cares  ↩
                     for
handling the ":" and "/" delimiters correctly

Template
  (TACKON [path] [file])
```

```
Parameters
  [path] - the path, which has to be expanded
  [file] - the file, which will tacked on

Options

Result
  Type: STRING
  Returns a new path

Note

Example
  see
              FOREACH
                See also
```

## 1.121 TEXTFILE

```
Creates a new text file from other text files or by strings. This
is useful to create configuration files scripts or environments.

Template
  (TEXTFILE [option]* )

Parameters

Options
  PROMPT
  HELP     - tell the user what's going to happen
  CONFIRM  - if present, then the user will be asked for confirmation
  SAFE     - create the file even in "Pretend" mode
  APPEND   - write this string to the text file
  INCLUDE  - include the given file into th new text file

Result
  Type: NUMBER
  Returns 0

Note

Example

See also
```

## 1.122 TOOLTYPE

```
Modify the tooltypes of an existing tool. I.e. you can set,
change and delete tooltypes and the related values.
```

```
Template
  (TOOLTYPE [option]* )

Parameters

Options
  PROMPT
  HELP           - tell the user what's going to happen
  CONFIRM        - if specified, the user will be asked for confirmation
  SAFE           - modify even in "Pretend" mode
  DEST           - the name of the icon to be modified; there is no need
                   to spevify e ".info" extension
  SETTOOLTYPE    - the tooltype name and its value string
  SETDEFAULTTOOL - specify the default tool for the icon
  SETSTACK       - the stack value
  NOPOSITION     - clear the position of the icon
  SETPOSITION    - two numbers to specify the position for the icon
  SWAPCOLORS     - <obsolete, ignored>

Result
  Type: NUMBER
  Return 0

Note

Example

  (TOOLTYPE "InstallerNG"
            (SETSTACK 50000)
            (NOPOSITION)
            (SETTOOLTYPE "MINUSER")  ; remove the MINUSER tooltype
            (SETTOOLTYPE "DEFUSER" "AVERAGE")  ; set the DEFUSER=AVERAGE  ↩
               tooltype
            (SETTOOLTYPE "ALWAYSCONFIRM" "")  ; specify the ALWAYSCONFIRM
  )


See also
```

## 1.123 TRACE

```
                Set a "Tracepoint" somewhere in the code. Use the function  ↩
                   RETRACE or BACK
to jump to this point. You are allowed to set as many tracepoints as you want.

Template
  (TRACE)

Parameters

Options

Result
  Type: NUMBER
  Returns 0 (zero)
```

```
Note

Example
  see
              RETRACE
                See also

              RETRACE

              BACK
```

## 1.124 TRANSCRIPT

```
Write some text to the logfile.

Template
  (TRANSCRIPT [string]* )

Parameters
  [string] - the strings to write to the log. All strings will be concatenated
             and appended by a linefeed.

Options

Result
  Type: STRING
  Returns the written text

Note

Example

See also
```

## 1.125 TRAP

```
Used for catching errors. Works much like C "longset" function, i.e. when
an error occurs while interpreting the functions inside of the TRAP, control
is passed to the function rights after TRAP.
[flags] determine which errors are trapped. The trap function itself
returns the error type or zero if no error occurred.

Template
  (TRAP [flags] [fun]* )

Parameters
  [flags] - specify the error, which should be catched; valid error codes
            are 1 - user abort
                2 - out of memory
                3 - error in script
```

```
                4 - DOS error
                5 - bad parameter data
  [fun]   - the functions, which are interpreted inside of the TRAP

Options

Result
  Type: NUMBER
  Returns the error code itself or zero, if no error occured.

Note

Example

  ; #errcode holds 1 in case of an error or 0, if no error occured
  (SET #errcode (TRAP 1 ( /* do anything here, what should be trapped */ ))) {NG ↩
     }


See also
```

## 1.126 UNTIL

```
A list of functions will be executed until the condition holds (or: while
this condition does not hold)

Template
  (UNTIL [condition] [fun]* )

Parameters
  [condition] - a boolean expression
  [fun]       - a list of functions which are executed as long as [condition] is ↩
      FALSE
                (or until [condition] is TRUE)

Options

Result
  Type: depends
  Returns the result of the last function

Note

Example

  (SET i 5)                      ; set a variable i to value 5
  (UNTIL (= i 0)                 ; check whether i equals to zero
         (                       ; if i doesnt equal to zero then:
           (MESSAGE "i = " i)    ;   - print the value of i
           (SET i (- i 1))       ;   - decrement i with 1
         )
  )
```

```
See also
```

## 1.127 USER

```
Change the user mode. You must not SET the @user variable - use USER
for this.

Template
  (USER [level])

Parameters
  [level] - the new user level. Must be either 0 (or "novice"), 1 (or
            "average") or 2 (or "expert")

Options

Result
  Type: NUMBER
  Returns [level]

Note

Example

See also
```

## 1.128 WELCOME

```
Use this function to show the Welcome panel of the Installer. If the
Installer cannot find WELCOME in your script, it pretends that its
first function is WELCOME and, thus, initially shows the Welcome panel.

Whithin the Welcome panel you select the User mode (Novice, Average or
Expert) and set the Logfile, the installation mode (Real or Pretend)
and with the InstallerNG you can also set advanced features.

In addition, WELCOME sets the @user-level and @pretend variables.

Template
  (WELCOME [string]* )

Parameters
  [string] - The string arguments are prepended to the standard help text
             for whichever of the two initial displays appears first

Options

Result
  Type: STRING
  Returns the concatenated strings
```

Note

Example

See also

## 1.129 WHILE

Execute a list of functions as long as a condition holds.

Template
  (WHILE [condition] [fun])

Parameters
  [condition] - a boolean expression
  [fun]       - a list of functions which are executed as long as [condition] is ↩
      TRUE

Options

Result
  Type: depends
  Returns the result of the last function

Note

Example

```
  (SET i 5)                       ; set a variable i to value 5
  (WHILE (> i 0)                  ; check whether i is greater then zero
         (                        ; if i is greater than zero then:
           (MESSAGE "i = " i)     ;   - print the value of i
           (SET i (- i 1))        ;   - decrement i with 1
         )
  )
```

See also

## 1.130 WORKING

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).

Template
  (WORKING [string]* )

Parameters
  [string] - the strings for the working text

```
Options

Result
  Type: STRING
  Returns the text

Note

Example

See also
```

## 1.131 XOR

```
The logical "xor", i.e. XOR deliveres true if exactly one argument is true.

Template
  (XOR [value1] [value2])

Parameters
  [value1]
  [value2] - the values which should logically be tested

Options

Result
  Type: NUMBER
  Returns 1 for true and 0 for false

Note

Example

See also
```

## 1.132 ALL

```
Template

Parameters

Options

Result

Note

Example
```

## 1.133 APPEND

Template

Parameters

Options

Result

Note

Example

## 1.134 ASSIGNS

Template

Parameters

Options

Result

Note

Example

## 1.135 BACK

Template

Parameters

Options

Result

Note

Example

See also

## 1.136 CHOICES

Template

Parameters

Options

Result

Note

Example

## 1.137 COMMAND

Template

Parameters

Options

Result

Note

Example

## 1.138 CONFIRM

Template

Parameters

Options

Result

Note

Example

## 1.139 DEFAULT

Template

Parameters

Options

Result

Note

Example

## 1.140 DELOPTS

Template

Parameters

Options

Result

Note

Example

## 1.141 DEST

Template

Parameters

Options

Result

Note

Example

## 1.142 DISK

Template

Parameters

Options

Result

Note

Example

## 1.143 FILES

Template

Parameters

Options

Result

Note

Example

## 1.144 FONTS

Template

Parameters

Options

Result

Note

Example

## 1.145 HELP

Template

Parameters

Options

Result

Note

Example

## 1.146 INCLUDE

Template

Parameters

Options

Result

Note

Example

## 1.147 INFOS

Template

Parameters

Options

Result

Note

Example

## 1.148 NEWNAME

Template

Parameters

Options

Result

Note

Example

## 1.149 NEWPATH

Template

Parameters

Options

Result

Note

Example

## 1.150 NOGAUGE

Template

Parameters

Options

Result

Note

Example

## 1.151 NOPOSITION

Template

Parameters

Options

Result

Note

Example

## 1.152 NOREQ

Template

Parameters

Options

Result

Note

Example

## 1.153 OPTIONAL

Template

Parameters

Options

Result

Note

Example

## 1.154 PATTERN

Template

Parameters

Options

Result

Note

Example

## 1.155 PROMPT

Template

Parameters

Options

Result

Note

Example

## 1.156 QUIET

Template

Parameters

Options

Result

Note

Example

## 1.157 RANGE

Template

Parameters

Options

Result

Note

Example

## 1.158 SAFE

Template

Parameters

Options

Result

Note

Example

## 1.159 SETTOOLTYPE

Template

Parameters

Options

Result

Note

Example

## 1.160 SETDEFAULTTOOL

Template

Parameters

Options

Result

Note

Example

## 1.161 SETSTACK

Template

Parameters

Options

Result

Note

Example

## 1.162 SOURCE

Template

Parameters

Options

Result

Note

Example

## 1.163 SWAPCOLORS

Template

Parameters

Options

Result

Note

Example