InstallerNG

COLLABORATORS			
	TITLE : InstallerNG		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 19, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

InstallerNG

1.1 Well, some nice quotes...

Life is a strange thing
Just when you think you learned how to use it
It`s gone
Shakespears Sister
We talk about our flights
In this queer dimension
And how we are afraid
To carry on alone
And finish our direction... flying home
Linda Perry
They lowered my body in a dark hole
The dry ground now covers my remains
I stood at their side and watched them crying
They can`t know, the tears they are in vain
Rage
Remember in this game we call life
That no one said it`s fair
Axl Rose
Und du rufst in die Welt
Daß sie dir nicht mehr gefällt
Du willst `ne schönere erleben
Doch es wird keine and`re geben
Witt/Heppner
Well who am I to complain
About a bit of earthly pain?

Tito & Tarantula

How `bout not equating death with stopping?

Alanis Morissette

D'Oh!

Homer Simpson

Hörst Du die Engel singen?

Hörst Du die Harfen klingen?

Hat sich das Leiden nicht gelohnt?

Fütter das Weiße Licht für mich!

OOMPH!

Go on

1.2 Installer NG

InstallerNG v1.2 pre-release

THIS IS A PUBLIC BETA VERSION

THUS THERE MAY BE MANY BUGS AND MISBEHAVIOURS AND MISSING FUNCTIONALITY.

USAGE IS YOUR OWN RISK.

Maybe an error occurs while you use my InstallerNG. If this happens, follow these

steps:

- note the Hunk/Offset trace and the error (if you use e.g. Mungwall/Enforcer)

- send me a bugreport and please include:

a) the script, which caused the error

b) complete error description (what you did, what happend...)

Introduction

The Installer

1.3 Introduction

If you know the original Installer tool, provided by the AmigaOS, then you don`t need to read this ...
You may know large packages of software, which come with several libraries, fonts, envs, which needs assigns, new directories and so on. Well, an installation on your own may be very hard for novice users. Okay, this tool makes this process easier: the author of a program should write a socalled "Installer-Skript", which consist of a special syntax and which uses the powerfull functionality of the InstallerNG. This script will be executed by this Installer tool. To learn about the language of these skripts, you should go here. If you want to know something about how to run such scripts, have a look at this.

1.4 Disclaimer

Since I really don`t like people, who release every damn shit as "Shareware", I decided to make this project "Freeware". Maybe a upcoming "developer version" (which includes a source-level debugger, a script creator etc...) will be "Shareware", but this InstallerNG won`t.
This version 1.0 pre-release is Freeware. You are allowed to spread this package, as long as the archive is complete and as long as no profit will be made. You are also allowed to put this InstallerNG into your own releases for installing your stuff, as long as you put the guide also into your release.
Suggestions, bugreports, money and sweet girls are always welcome ;)

1.5 I need your help!

Maybe you can help me with one (or more) of the following things:
- who would paint some nice icons for buttons and a cool logo?

1.6 That`s me ;)

Snail Mail
Jens Tröger
Hochschulstraße 48, 11-4
01069 Dresden
Germany
Phone
+49351/4701609
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

1.7 Distribution of the InstallerNG

The InstallerNG will be distributed and supported by
LightSpeed Communications GbR
c/o Jens Langner
Bergstraße 68
01069 Dresden
Germany
WWW
http://www.light-speed.de

1.8 I want to thank...

Special thanx for...
patience and testing the InstallerNG again and again
Jens Langner
Sven Steiniger
interesting talks about coding
Olaf Barthel
Michael Rock
Sven Steiniger

additional

James Maurice Battle - for his icon and the nice picture

And last, but really not least, all those people, who have sent me so many nice & fanatic mails and who are the important reason, that i will go on with writing this program...

Tino Wildenhain Marcin Orlowski

Robert Reiswig Jon Peterson

Lee Stoneman

Andrea Valinotto

Christian Hattemer

Daniel Confora

Frank Pagels

Henning Kiel

Jeff Grimmet

Kai Hofmann

Linus McCabe

Marko Seppanen

Markus Merding

Martin Steigerwald

Oywind Falch

Philippe Bovier

Pieter Frenssen

Thomas Klein

Soyeb Aswat

Tobias Abt

Alasdair Simpson

Jens Weichert

Dobes Vandermeer

Joel Newkirk

Grzegorz Kraszewski

John Pullicino

Dirk Stöcker

Rainer Müller

Herve Dupont

1.9 Still not 100% perfect

The C= Installer is a mess... buggy, bad language handling and weird
type constructs. I tried to do a better job, but one aim was, to stay
compatible with the C= installer. Thus, the InstallerNG has also some
'weird' features (note: these are no bugs, these are features!), but
on the other hand, I think I built a better compiler kernel and a much
better interpreter :)
If you want to compile 'buggy' scripts, you can switch on the LAZYCOMPILE
Tooltype/CLI-Argument to skip any check procedures!

1.10 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))
```

1.11 What ist left to do

Personal aims
- make STARTUP with CONFIRM complete
- automatically turn off the UN!X-Dirs commodity to avoid path conflicts
- retry option for DOS errors
- reduce the memory overkill
- remove all known Bugs
- new function: CREATE-ICON
- implement the logfile and delete-skript creation (maybe an option to select whether the InstallerNG should create an uninstall shell-script or an uninstall installer-script)
- variable spy, source-level debugger
LISP like extensions
- CLOS: new functions DEFINE-CLASS, MAKE-INSTANCE, DEFINE-METHOD, INVOKE-METHOD etc.
- CATCH and THROW (or enhance the ONERROR and TRAP functions)
- continuations: CALL-CC (something like the TRACE/RETRACE functions)
Suggestions by others
- reading compressed skripts would be very useful (Tobias Abt)
- lowlevel function to partition and install alien file systems via special functions (Thomas Mache)
- when running the installer with a *.lha file instead of a script (new tooltype ARCHIVE ?) the installer will extract a file "install" from the archive and uses this for installing directly from the archive (Dirk Stöcker)
- something like a "global logfile" which holds all installed packages (Alasdair Simpson)
- image buttons (Danial Canfora)
- a function INCLUDE to read external sources (thus: writing a preprocessor?) (Kai Hofmann)
- modify startup-sequence optionally (Volker Schmitt)
- automatic creation of directories via ASKDIR (Alexander Reiffinger)
- online registration for software (Jürgen Haage)
- give the Installer an application window, so that one can "drop" the destination path/file (??)
- give Installer a "gauge-port" such that a command (started with the EXECUTE function) can signal its current working state (useful: pipe the lha-output to a special tool which converts it and sends working-state-signals to Installer) (Ignatios Sourvakis)

- show device-list when invalid directory was typed in ASKDIR/ASKFILE functions
(Jens Langner)
Does this make sense ?
- type inferencing
- saving the compiled program as binary for a later reloading
- use XPK-functions for unpacking
- add constants to the language: e.g. (SETCONST a 5)
- shared library interface for simple libcalls:
raw: (LIBCALL "dos" -198 "d1" 50) ; Delay(50)
(LIBCALL "intuition" -78) ; CloseWorkBench()
FD files: (LIBCALL "dos.Delay(50)")
(LIBCALL "intuition.CloseWorkBench()")
Wildest dream :)
Create executables. Running this executable has the same effect as running
the installer and a script. This should reduce the resources in any way.

1.12 How to start the Installer

There are two possibilities to run the Installer:
From WB
From Shell/CLI

1.13 Running from WB

SCRIPT
APPNAME
MINUSER
DEFUSER
LOGFILE
LANGUAGE
PRETEND
LOG
NOPRINT
ICONIFY
LAZYCOMPILE
DEBUGMODE
CREATEUNINSTALL
COPYFILECOMMENT
ALWAYSCONFIRM
NOSYSDELETE

1.14 Running from Shell/CLI

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

1.15 The history of this Program

1.2 pre-release (current...)
September 6, 1999 (my sister’s birthday)
- function EFFECT will fall back to the "Workbench" screen, if the custom
screen for the InstallerNG fails
- included the .license file (thanx to Jens Langner)
- removed an enforcer hit in the builtin gui
September 2, 1999
- when done with the installation, the "Proceed..." button gets renamed to
"Finish" (James Maurice Battle)
September 1, 1999
- fixed small problem with reading version strings (Jens Langner)
- if started from WB, then the variable @ICON was set to the script (wrong)
and is now set to the icon path
- @ICON is now a constant and, thus, cannot be modified
August 30, 1999
- RUN also supports @EXECUTE-DIR for now
- added the forgotten @ABORT-BUTTON support (builtin GUI only)
1.1 pre-release (Aug 30, 1999)
August 29, 1999
- modified the version-string handling and the catalog files
- builtin GUI: fixed some Enforcer-Hits in the Listview and in the makedir
window of the ASKFILE function
August 27, 1999
- COPYFILES crashed with the installergui.library (James Maurice Battle)
- builtin GUI: the InstallerNG now opens with an optimized window size,
if the gui was not snapshoted (Tobias Abt)
August 25, 1999
- if a WORKING follows another WORKING, the last one was not correctly
layouted (i.e. it appeared in the window frame) (Sven Steiniger)
- sometimes the InstallerNG asked the "Really quit" question two times

(Jens Langner, Dirk Stöcker)
- scanner could read wrong numbers in special cases
1.0 (August 24, 1999)
August 23, 1999
- if an condition (see IF, WHILE, UNTIL) evaluated to an zero-length
string value, this is now handled as value FALSE of type NUMBER
- moving the locked window to an EFFECT screen caused an enforcer hit
August 22, 1999
- every new InstallerNG process can open its own EFFECT screen
- now cares for the return value of Intuition.CloseScreen()
- simplified the GUI-API: igui_WakeApp(), igui_KickWakeApp() and igui_SleepApp()
are no longer part of the API, the gui has to care for this itself
August 17, 1999
- COPYLIB does nothing if soure-version equals dest-version (Jens Langner)
- reading the version of files with resident-structures (e.g. shared-libs
or devices) always returned version 0.0
- sometimes the GUI was locked, although the installer waits for
user input
August 15, 1999
- COMPARE evaluated an empty string to a number of value zero and, thus,
could produce type conflicts
- the installer now sets a return code: RETURN_OK (0) if everything was
fine or RETURN_FAIL (20) if any error occured (??)
- "@installer-version" now equals 44.6 (Installer version of OS3.5)
August 14, 1999
- new function NOP :-)
- finished the MUI API
- finished some new functions of the OS3.5 Installer tool:
EFFECT/TRACE/RETRACE/BACK
August 4, 1999
- hopefully finished the new builtin BOOPSI-GUI
- many, many, many internal fixes and rewrites and modifications and
other work and discussions with users
0.8 (June 23, 1999)
June 23, 1999
- when no error occured inside of a TRAP statement, then TRAP raised an
"Unknown error"
- pressing "Abort" inside of a TRAP statement caused the "Are you sure to quit"
requester to pop-up

- fixed an Enforcer-Hit in COMPARE
- setting a variable to @EACH-NAME (see FOREACH) caused Mung-Hits outside
of that FOREACH statement (Jens Langner)

June 18, 1999

- the system does no longer block when the installer closes down

June 8, 1999

- DATABASE doesn’t return "680x0/PPC" anymore; if you want to check for a
PPC, use the (DATABASE "ppc") statement

June 7, 1999

- EQU (= ...) raised a type conflict when first argument was a string and the
second one a string-encoded numer (Jens Langner)
- SELECT evaluated (n-1)th node instead of the n-th node (Jens Langner)

June 3, 1999

- ported the source code from StormC to SAS-C and recompiled the
complete projects with SAS-C

May 29, 1999

- removed the @system-language variable and modified the behavior
of @language to be compatible with C= (Jens Langner)
- COPYFILES now also shows the filename of the source (Jens Langner)

April 9/10, 1999

- reworked the error handling and the type system
- added functions PUT-PROPERTY, GET-PROPERTY and REMOVE-PROPERTY to realize
LISP-like property-lists for symbold and, thus, to implement a really simple
kind of OOP (let`s call this SOOP)

April 8, 1999

- modified the interpreter kernel for better multi-threading
- reworked CONCURRENT-DO and the stacktrace protocols

April 7, 1999

- CONCURRENT-DO crashed sometimes because of the stacktrace protocol; fixed this
bug, but, thus, the interpreter kernel is a bit slower now! (i should remove this
function... it`s a bit superfluous, isn`t it?)

March 26, 1999

- added new installmode NOSYSDELETE, which forbids to delete files from system
drawers (Alexander Reiffinger)
- COPYFILES/COPYLIB supports now OPTIONAL/DELOPTS (thus i rewrote the functions for
cloning and deleting files, hope they are still correct!)

March 24, 1999

- sending CTRL-F to any of the slave-processes, caused the MUI gui to hang up

March 23, 1999

- optimized code
- removed possible type conflicts in some math functions
- DELETE now supports OPTIONAL and DELOPTS

March 22, 1999

- TOOLTYPE now works correctly with SETTOOLTYPE (sorry, i simply forgot to implement this feature ;-)

March 21, 1999 (beginning of Spring ;)

- sending CTRL-F caused the installer to execute all the ONERROR statements
- ALWAYSCONFIRM overwrote PROMPT and HELP strings everytime
- by setting the tooltypes of the InstallerNG itself, you may now preset the working environment (Tino Wildenhain)

March 19, 1999

- removed/optimized some code

March 13, 1999

- a stacktrace is dumped in the case of a runtime error and if the SHOWDEBUG option is switched on
- GETSUM is now compatible to the algorithm of the C= installer (Marcin Orlowski)

March 11, 1999

- scanner/parser errors are now located correctly (Jens Langner)

March 10, 1999

- REXX did not support PROMPT/HELP/CONFIRM
- added new installmode (thus, new Tooltype/Shell-Argument) ALWAYSCONFIRM, which always asks the user for confirmation of every action (??)
- added new Tooltypes/Shell-Arguments DEBUGMODE, CREATEUNINSTALL, COPYFILECOMMENT (Jens Langner)

March 2, 1999

- added new function RANDOM

February 23, 1999

- added new function SWING to allow special UNDO/REDO environments

February 22, 1999

- DELETE failed if the destination file was nonexistent

February 21, 1999

- WELCOME and the other functions which use the applications name, now use the value of "@APP-NAME" (which is initially set to "APPNAME") instead of the "APPNAME" startup argument
- optionally set the comment for every copied file to the packages name (i.e. the value of "@APP-NAME") (??)

0.7 (February 16, 1999)

February 15, 1999

- GETASSIGN now returns NULL instead of an empty string (as noted in the original
installer documetation), if the assign is not valid (Tobias Abt)
- small face-liftings (Jens Langner)
- special thanx to Jens Langner: he wrote a function to center too long
texts inside of (buggy!) MUI Floattext objects
February 14, 1999
- GETASSIGN now expands the result, if this is a valid path (Jens Langner)
- FINDBOARD now returns the number of existing boards (Tobias Abt)
February 5, 1999
- the german catalog contained some mistakes
- the semantic-checker skiped the procedures (Jens Langner)
February 3, 1999
- COPYFILES and PATTERN didn`t work together
- quitting while COPYFILES was waiting for user confirmation left
file-locks unreleased
- the semantic checker now reports missing user-level of CONFIRM
- while creating a full path, MAKEDIR overwrote existing drawer-icons
January 22, 1999
- because of ignoring runtime-errors, some enforcer hits raised when the installer
used NULL-vars of type string (Tino Wildenhain)
- the builtin-variable @PRETEND was not of type number (Tino Wildenhain)
- added gauge to show the continuation of scanning/parsing (Tino Wildenhain)
- if debug-mode is on, then runtime-errors will also be written to the debug console
January 17, 1999
- facelifted the gui (Tino Wildenhain)
January 16, 1999
- variables named "ADD", "SUB", "MUL", "DIV" where scanned as function symbols
(Jens Langner)
December 17, 1998
- seems that MUIA_List_Format, "P=\33c" (ver 19.8 of Floattext class) does not
work - there is no line wrapping! thus the InstallerNG does not center the text
- the scanner now handles all C= Installer v42.6 escape sequences in the same way
- COPYLIB does nothing when source-files version equals dest-files version
- hopefully fixed the COPYFILES bug
December 14, 1998
- supports now binary and hex numbers
- calling empty procedures caused enforcer hits
November 25, 1998
- when InstallerNG runs in debug mode, uninitialized vars and pattern errors are

reported
- the InstallerNG uses several own console windows for debug-, uninstall- and
standard-output
- REXX now works (thanx to Olaf Barthel, Andrea Vallinotto, Jeff Grimmett)
November 24, 1998
- when InstallerNGs "Debug" option is on, every access to an uninitialized
variable will be reported
- OPTIONAL/DELOPTS are working correct (only as local definitions), but are not
yet supported by COPYFILES/COPYLIB/DELETE (Olaf Barthel)
- DELETE always removed the .info files
- error requester of DOS-errors was unreadable, when IoErr() delivered 0
- forgot to unlock the target directory, if DELETE failed
- the builtin pattern-matcher now works the same way like the original one (removed
Charles Bloom`s matcher, sorry) (Olaf Barthel)
November 23, 1998
- ONERROR did not execute all previous defined ONERROR statements (Olaf Barthel)
November 22, 1998
- hopefully re-bugfixed lost changes from Oct 1, 1998 till my crash on
Oct 9, 1998:
October 9, 1998
- the debug version deliveres much more information
- when running from WB, the builtin variable @icon did not hold the complete
path to the script (Marko Seppanen)
October 8, 1998
- several builtin variables weren`t accessable
- started to use a custom pattern matcher, to be more compatible to the
C= installer (thanx to Charles Bloom for CRBLIB source)
October 2, 1998
- added /* and */ for multi-lined comments to the language
October 1, 1998
- while the semantic checks, you may optionally skip the errors/warnings or you
can create a dump of them (Jens Langner, Sven Steiniger)
November 21, 1998
- TRAP now catches only the specified error and raises an interpreter error
for none-catched errors
November 19, 1998
- wrote the TRAP function (special thanx to Olaf Barthel)
November 16, 1998
- nesting TRAP/ONERROR/PROCEDURE confused the parser and caused weired structured

syntax-trees (Sven Steiniger)
- error-handling of unknown user functions (PROCEDURE) caused an enforcer hit
November 10, 1998
- debug output is ON by default
- removed scanner error for multilined comments
On November 9, my HD crashed and I lost all changes from October 1 till now. Thus
I have to rewrite lost parts, update some tools and so on. Special thanx to
Jens Langner for fast help and for beeing my "backup server" :)
0.6 (October 1, 1998)
October 1, 1998
- ASKNUMBER did show the "range" comment everytime
- TOOLTYPE`s SETPOSITION wrote zeros, although no SETPOSITION was given
September 28, 1998
- CONCURRENT-DO gives every new child process a specific number
- modified the scanner, but forgot to also modify the builtin variable names :)
September 26, 1998
- when an error occurs, then InstallerNG also shows the name of the buggy function
September 25, 1998
- COPYFILES didn`t raise an error, if the SOURCE does not exist (Thomas Lenz)
September 24, 1998
- RUN, EXECUTE did not accept more than one argument to build the command, which
has to be started
- new function DELAY
- you can quit the installer everytime, by sending the CTRL-F signal
to its process (Oliver Brunner)
September 20, 1998
- changed the catalog file
- now DOS errors are also be shown
- execution of a script can now be continued with the next statement
(please be prepared, that this may lead into new errors!)
September 9, 1998
- new function REBOOT
- the scanner now removes "<ESC>[2p" sequences at the beginning of every
string (Jens Langner)
September 4, 1998
- set all "@..-help" variables to their correct text values
- localized the ASKBOOL function
September 3, 1998
- ASKFILE always returned the path part of the selected file

September 1, 1998
- DATABASE should now recognize the Picasso96 graphics system
(thanks to "DaMato" Jens Langner)
August 31, 1998
- first public beta version released
0.5 (August 28, 1998)
August 28, 1998
- removing of inner LET environments (LET env inside of a LET environment) failed
- DATABASE can now read the current system date and time
- new function LET
August 16, 1998
- new function CONCURRENT-DO
- started to make big parts of the InstallerNG reentrant, to prepare
the implementation of a new function (see above)
August 1, 1998
- DELETE should be much faster, if you don`t use pattern matching
- disabling the DEBUG output also disabled the interpretation of the
arguments of the DEBUG function
July 31, 1998
- SELECTing the n-th node of a list failed under different conditions
July 30, 1998
- switching on/off the creation of the uninstall-script did not work
- functions, which use CONFIRM/COMMAND/SETTOOLTYPE/GETTOOLTYPE did not work anymore
(because i modified the environment handling yesterday...)
July 29, 1998
- string-format routines of several functions optimized
- PROTECT was a bit buggy...
- MAKEASSIGN did not work correct with SAFE removing of system assigns
- reworked the whole environment handling
July 28, 1998
- some functions still overwrote the local environment of the outside-function
(if there was one)
- the raw body of the uninstall-script will be printed to CON:
- ASKNUMBER now corrects a DEFAULT which is out of RANGE
July 27, 1998
- COMPLETE reworked and optimized
- when using CONFIRM for the COPYFILES functions, the copy failed (Falk Zühlsdorff)
0.4 (July 26, 1998)
July 26, 1998

- added keyfile support: if no keyfile is available, then the InstallerNG
always runs in pretend mode
- functions using CONFIRM now asking for confirmation, even in pretend mode
July 25, 1998
- in pretend mode, COPYFILES/COPYLIB always raised the "bad source" error (Dirk Stöcker)
July 24, 1998
- updated the guide, espacially the "enhanced functions" (see What`s new)
- DATABASE now returns correct values and supports the checkvalue functionality
- if an identifier contained chars with ascii-code greater than 0x7F, then this
caused the scanner to interpret this char as an intertoken space
- ASKCHOICE/ASKOPTION skipped the last choice (Dirk Stöcker)
- "@installer-version" now equals 43.3 (as the latest C= installer version is)
(Dirk Stöcker)
- new variable "@installer-ng-version" which is 0 iff the script does not run on the
InstallerNG, otherwise it is set to the version of the InstallerNG
(Dirk Stöcker)
July 23, 1998
- MAKEASSIGN always caused an enforcer hit
- Menu "Quit" did not work
July 13, 1998
- added the FINDBOARD function
July 12, 1998
- updated the Known Bugs list :(
- optimized the lowlevel code of DATABASE
July 10, 1998
- dedicated to Dirk Stöcker: debug output also prints access to uninitialized variables
- EXISTS modified the wrong environment or caused an enforcer hit (Dirk Stöcker)
- ASKCHOICE/ASKOPTION returned wrong results if several CHOICES had zero length
argument(s) (Dirk Stöcker)
July 9, 1998
- DATABASE now identifies PPC processors and the CV graphics card
July 7, 1998
- ASKNUMBER/COPYLIB prompted weired CONFIRM message
- mixing strings with numbers for comparison results in a infinity loop
- added: ability to send the output of DEBUG to nil:
- the logfile will now be created as "T:installer_log_file"
- MAKEDIR stopped creating new directories before it reached the end of the
destination path (Sven Steiniger)
- the reduction of an arbitrary path reached a polling, iff the first component

of this path was an invalid device (Sven Steiniger)
July 5, 1998
- reworked the string-format routines (thanx for trick: Sven Steiniger)
- relational functions raised type conflict error when a number was compared to a
signed string number
July 3, 1998
- DELETE supports the OPTIONAL/DELOPTS settings
- reworked the pattern-match routines to prepare custom code for the matching
process itself
July 2, 1998
- COPYLIB has had problems with DEST path (Jens Langner)
- COPYLIB now makes correct version check even if CONFIRM is not set (Jens Langner)
- WELCOME now really quits when pressing the "Abort" button (Jens Langner)
July 1, 1998
- reworked my linker library to work with all data models (FAR, NEAR, NEAR_A6) and
found some small mistakes. hope i did not add new ones ;)
June 29, 1998
- OPTIONAL now modifies the global environment, not the local one of the functions
DELETE/COPYFILES/COPYLIB (Dietmar Eilert)
- string-format without args ("string") now returns correct type (Sven Steiniger)
- MAKEDIR now works correct too, when the path of the directory was given without
a slash (MAKEDIR "ram:bla") (Sven Steiniger)
- CONFIRM accepts no longer zero arguments (switch on LAZYCOMPILE to skip)
(Sven Steiniger)
- remove the "Scanning, parsing and... " message, when the installer starts to
interpret the statements before a WELCOME (Sven Steiniger)
June 26, 1998
- ASKFILE/ASKDIR should handle the SOURCE and DEST parameters correct (Falk Zühlsdorff)
- STARTUP now modifies "user-startup", but still does no modification to
"startup-sequence"
0.3 (June 26, 1998)
- every function which has related parameter-functions now creates a dynamic
runtime environment for the results of the parameter-functions
- added new functions: BEEP, SETENV, SIMULATE-ERROR, COMPARE
- added new tooltype/cli-argument: LAZYCOMPILE
- fixed dozens of silly bugs :)
0.2 (June 1, 1998)
- grammar rewritten (and thus, most parts of parser/tree-evaluator)
- bugfixing and optimisations

- added serial output to most of the functions (debug-beta version)

0.1 (somewhere in fall 1997)

Started in october 1997 this project. Earlier (august) first formal descriptions.
First working parts (scanner, parser ...) in november. Added the GUI (first
version: MUI by Stefan Stunz) in november too.
Fully functional versions in january/february 1998.

1.16 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

1.17 The symbols of the language

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 \o123 \x123"

'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.

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
*/
```

1.18 Builtin functions

NOTE: - functions marked with a * are new in the InstallerNG

- functions marked with a + provide enhanced functionality

- functions marked with v44+ are also new with v44 of the C= installer and AmigaOS 3.5

SOOP support

GET-PROPERTY *

PUT-PROPERTY *

REMOVE-PROPERTY *

Conditional

IF

UNTIL

WHILE

Multimedia support (needs datatypes)

CLOSEMEDIA v44+

SETMEDIA v44+

SHOWMEDIA v44+

Functions

=

<>

>

>=

<

<=

COMPARE *

+

-

*

/

AND

OR

XOR

NOT

BITAND

BITOR

BITXOR

BITNOT

IN

SHIFTLEFT

SHIFTRIGHT

ABORT

ASKDIR

ASKFILE

ASKSTRING

ASKNUMBER

ASKCHOICE

ASKOPTIONS

ASKBOOL

ASKDISK

BEEP *

CAT

COMPLETE

COPYFILES

COPYLIB

DATABASE +

DEBUG

DELAY *
DELETE
EFFECT v44+
EXECUTE
EXISTS +
EXIT
EXPANDPATH
EARLIER
FILEONLY
FINDBOARD *
FOREACH
GETASSIGN
GETDEVICE
GETDISKSPACE
GETENV
GETSIZE
GETSUM
GETVERSION
ICONINFO
LET *
MAKEASSIGN
MAKEDIR
MESSAGE
NOP *
ONERROR
PATHONLY
PATMATCH
PROCEDURE
PROTECT
QUERYDISPLAY v44+
RANDOM *
REBOOT *
RENAME
RETRACE v44+
REXX
RUN
SELECT
SET
SETENV *

SIMULATE-ERROR *
STARTUP
STRLEN
SUBSTR
SWING *
TACKON
TEXTFILE
TOOLTYPE
TRACE v44+
TRANSCRIPT
TRAP
USER
WELCOME
WORKING
Parameter Functions
ALL
APPEND
ASSIGNS
BACK v44+
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

1.19 Builtin variables

@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).

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

@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

@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

Version of the Installer. Note: this does NOT equal the version of the

InstallerNG!

Default: 0x002c0006 (which is a 44 in the upper word and a 6 in the lower one)

Type: NUMBER

@error-msg

@special-msg

@ioerr

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

Default: depends

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: (...)

Type: STRING

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. With v44 of the original Installer, there are still no local variables. Use the InstallerNG function LET, if you want to use locals.

```
(PROCEDURE version-to-string ; name
#ver ; argument
("%ld.%ld" (/ #ver 65536) (BITAND #ver 65535)) ; body
)
```

String formatting

If the functional symbol (remember: this is the leftmost one, because this is a prefix language) is of type string, then this string gets handled like a format string, and the following expressions are the format parameters. The InstallerNG supports all C like (see printf()...) formatsymbols.

For example, if you write

```
("Hi, I'm %s and I am %ld years old" "Nick" 29)
```

you will get the following as result of the evaluation:

"Hi, I'm Nick and I am 29 years old"

Function lists

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
(MESSAGE "#bla equals #surz")
(BEEP)
)
)
```

1.21 What is new here???

Here you find all the things I added to the Interpreter.
Note: if you 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 has some terrible restrictions: maximum string-length,
maximum size of a string value. The InstallerNG makes none of these:
a string (and the value of a string variable too) can be as long as it fits
into your memory.
Nice GUI
The 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".
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 now 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, InstallerNG provides
to continue at the very next statement. 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
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 statements)

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 (thanks to Oliver
Brunner for this suggestion)

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/RETRACE/BACK functions (have a look at the C= installer documentation)

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

Enhanced Functions

DATABASE

EXISTS

New Functions

BEEP

COMPARE

DELAY

FINDBOARD

LET

NOP

RANDOM

REBOOT

SETENV

SIMULATE-ERROR

SWING

GET-PROPERTY

PUT-PROPERTY

REMOVE-PROPERTY

1.22 Known Bugs, not yet removed *sorry*

My own bugs

- maybe some open resources (locks, some bytes of mem) when quitting

Thats not my fault...

- DATABSE crashes the system if there is no ppc-processor in the system but
the ppc.library is installed

- trying to open "powerpc.library" without having a PPC processor or WarpOS installed
causes lots of enforcer hits

1.23 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 statement

(or function expected

The Installer needs the beginning of a new statement 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 ")" ???

1.24 Trouble with other soft

UN!X-Dirs
Some people may use this commodity. But please note: if you work
with *X, then something like '/bla' means to be a mounted volume
(the amiga equivalent is 'bla:'). A software package may now come
with several subdirs and, thus, an install script would like to
copy from '/mytools' to your destination. This collides with UN!X-
Dirs and a "Volume not mounted" requester pops up.
SOLUTION: Turn off UN!X-Dirs before installing soft!

1.25 IF

Conditionally execute statements. 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> expression/statements are executed if <condition> is TRUE
<else> expression/statements are executed if <condition> is FALSE
Options
Result
Returns the result of <then> or <else>
Note
Example
(IF (= 2 4) ; condition
(MESSAGE "TRUE") ; then
(; else

(MESSAGE "FALSE")

(BEEP)

)

)

See also

1.26 WHILE

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

Template

(WHILE <condition> <statements>)

Parameters

<condition> any function

<statements> a list of statements which are executed as long as <condition> is TRUE

Options

Result

Returns the result of the last <statement>

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

1.27 UNTIL

A list of statements will be executed until the condition holds (or: while

this condition does not hold)

Template

(UNTIL <condition> <statements>)

Parameters

<condition> any function

<statements> a list of statements which are executed as long as <condition> is FALSE

(or until <condition> is TRUE)

Options

Result

Returns the result of the last <statement>

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

1.28 ABORT

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

Template

(ABORT <msg> <msg> ...)

Parameters

<msg> - strings which will be concatenated an shown right before
the InstallerNG starts to execute the ONERROR statements

Options

Result

Type: NUMBER

Returns 0

Note

Example

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

1.29 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 done work

Options

Result

Type: NUMBER

Returns the argument <done>

Note

Example

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

See also

1.30 COPYFILES

Template

Parameters

Options

Result

Note

Example

1.31 COPYLIB

Template

Parameters

Options

Result

Note

Example

1.32 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 <whatever> <whatever> ...)

Parameters

<whatever> - this can be anything: a number, a string, an expression.
DEBUG prints the evaluation-result of <whatever> to the
console window, followed by a linefeed.

Options

Result

Type: STRING

The result of the last <whatever> - evaluation

Note

If <whatever> is an uninitialized variable, then DEBUG prints

an "<NIL>" to warn the user

Example

(SET a 0)

(DEBUG 1 "does not equal" a b) ; results in "1 does not equal 0 <NIL>"

See also

1.33 DELAY

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

1.34 DELETE

Template

Parameters

Options

Result

Note

Example

1.35 EXECUTE

Template

Parameters

Options

Result

Note

Example

1.36 EXIT

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

Template

(EXIT <string> <string> ... (QUIET))

Parameters

<string> - these strings are concatenated and displayed as the final report

Options

(QUIET) - skip the final message

Result

Type: NUMBER

Returns 0 (zero)

Note

Example

1.37 FOREACH

Template

Parameters

Options

Result

Note

Example

1.38 LET

This function creates a new environment. This means, you can declare new variables within the <init> statements and use them in the <body> statements. 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 statement, 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 statements, which use this local environment

Options

Result

LET returns the result of the last statement 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
)
)
)
```

1.39 MAKEASSIGN

Template
Parameters
Options
Result
Note
Example

1.40 MAKEDIR

Template
Parameters
Options
Result
Note
Example

1.41 MESSAGE

Template
Parameters
Options
Result
Note
Example

1.42 NOP

Does nothing...
Since my language definition does not allow empty statement-lists,
I thought it would be useful to have a NOP function for this case :)
Template
(NOP)
Parameters
Options
Result
Type: NUMBER
Returns 0
Note
Example
(NOP)
See also

1.43 ONERROR

Template

Parameters

Options

Result

Note

Example

1.44 PROCEDURE

Template

Parameters

Options

Result

Note

Example

1.45 PROTECT

Template

Parameters

Options

Result

Note

Example

1.46 RENAME

Template

Parameters

Options

Result

Note

Example

1.47 REXX

Template

Parameters

Options

Result

Note

Example

1.48 RUN

Template

Parameters

Options

Result

Note

Example

1.49 SET

Template

Parameters

Options

Result

Note

Example

1.50 SETENV

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

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

1.51 SIMULATE-ERROR

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 statement.
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

1.52 STARTUP

Template

Parameters

Options

Result

Note

Example

1.53 SWING

This allows you to jump (inside of this block) from one statement to its neighbour statement. 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 statements. SWING will jump between them

Options

Result

Type: number

Returns 0

Note

You MUST NOT nest SWING functions.

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)
)
)
)
```

1.54 TEXTFILE

Template

Parameters

Options

Result

Note

Example

1.55 TOOLTYPE

Template

Parameters

Options

Result

Note

Example

1.56 TRAP

Template

Parameters

Options

Result

Note

Example

1.57 USER

Template

Parameters

Options

Result

Note

Example

1.58 WELCOME

Template

Parameters

Options

Result

Note

Example

1.59 WORKING

Template

Parameters

Options

Result

Note

Example

1.60 EQU

Template

Parameters

Options

Result

Note

Example

1.61 NE

Template

Parameters

Options

Result

Note

Example

1.62 GT

Template

Parameters

Options

Result

Note

Example

1.63 GE

Template

Parameters

Options

Result

Note

Example

1.64 LT

Template

Parameters

Options

Result

Note

Example

1.65 LE

Template

Parameters

Options

Result

Note

Example

1.66 COMPARE

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 InstallerNG tries always
to convert a string into a number if this is needed.
Example
(COMPARE 2 2) -> 0
(COMPARE 2 "2") -> 0
(COMPARE "bla" "nana") -> -1
See also

1.67 ADD

Template
Parameters
Options
Result
Note
Example

1.68 SUB

Template
Parameters
Options
Result
Note
Example

1.69 MUL

Template

Parameters

Options

Result

Note

Example

1.70 DIV

Template

Parameters

Options

Result

Note

Example

1.71 AND

Template

Parameters

Options

Result

Note

Example

1.72 OR

Template

Parameters

Options

Result

Note

Example

1.73 XOR

Template

Parameters

Options

Result

Note

Example

1.74 NOT

Template

Parameters

Options

Result

Note

Example

1.75 BITAND

Template

Parameters

Options

Result

Note

Example

1.76 BITOR

Template

Parameters

Options

Result

Note

Example

1.77 BITXOR

Template

Parameters

Options

Result

Note

Example

1.78 BITNOT

Template

Parameters

Options

Result

Note

Example

1.79 SHIFTLEFT

Template

Parameters

Options

Result

Note

Example

1.80 SHIFTRIGHT

Template

Parameters

Options

Result

Note

Example

1.81 IN

Template

Parameters

Options

Result

Note

Example

1.82 ASKDIR

Template

Parameters

Options

Result

Note

Example

1.83 ASKFILE

Template

Parameters

Options

Result

Note

Example

1.84 ASKSTRING

Template

Parameters

Options

Result

Note

Example

1.85 ASKNUMBER

Template

Parameters

Options

Result

Note

Example

1.86 ASKCHOICE

Template

Parameters

Options

Result

Note

Example

1.87 ASKOPTIONS

Template

Parameters

Options

Result

Note

Example

1.88 ASKBOOL

Template

Parameters

Options

Result

Note

Example

1.89 ASKDISK

Template

Parameters

Options

Result

Note

Example

1.90 BEEP

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

1.91 CAT

Template

Parameters

Options

Result

Note

Example

1.92 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>])

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" - checks for PPC; returns "PPC" if there is a PPC installed,
"" otherwise

"FPU" - which type of FPU ("68881", "68882", "FPU040", "FPU060")

"MMU" - which type of MMU ("68851", "MMU040", "MMU060")

"OS-VER" - the version of exec (e.g. "40")

"GRAPHICS-MEM" - amount of free chip memory

"FAST-MEM" - amount of free fast memory

"TOTAL-MEM" - total free memory

"CHIPREV" - the revision of the graphic chipset
("AA", "ECS", "AGNUS")

"GFXSYSTEM" - the installed graphics system
("CyberGraphics", "Picasso96")

"DATE" - the current date of your computer

"TIME" - the current time of your computer

"GUI" - type of the used GUI

<checkvalue> 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

1.93 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> <options>)

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

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

Example

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

(EXISTS "C:") --> should be 2

(EXISTS "grfm:hlbzs/hsjs") --> maybe 0 ;)

See also

1.94 EXPANDPATH

Template

Parameters

Options

Result

Note

Example

1.95 EARLIER

Template

Parameters

Options

Result

Note

Example

1.96 FILEONLY

Template

Parameters

Options

Result

Note

Example

1.97 FINDBOARD

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

1.98 GETASSIGN

Template

Parameters

Options

Result

Note

Example

1.99 GETDEVICE

Template

Parameters

Options

Result

Note

Example

1.100 GETDISKSPACE

Template

Parameters

Options

Result

Note

Example

1.101 GETENV

Template

Parameters

Options

Result

Note

Example

1.102 GET-PROPERTY

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

Example

(SET #bla "savage is cool :-)") ; declare a symbol #bla

(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")

)

(REMOVE-PROPERTY #bla "property") ; remove "property" from #bla

See also

PUT-PROPERTY

REMOVE-PROPERTY

1.103 GETSIZE

Template

Parameters

Options

Result

Note

Example

1.104 GETSUM

Template

Parameters

Options

Result

Note

Example

1.105 GETVERSION

Template

Parameters

Options

Result

Note

Example

1.106 PATHONLY

Template

Parameters

Options

Result

Note

Example

1.107 PATMATCH

Template

Parameters

Options

Result

Note

Example

1.108 PUT-PROPERTY

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

1.109 RANDOM

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

1.110 REBOOT

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)

Parameters

Options

Result :)

Type: NUMBER

Returns 0

Note

Example

(REBOOT)

See also

1.111 REMOVE-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

Example

see GET-PROPERTY

See also

GET-PROPERTY

PUT-PROPERTY

1.112 SELECT

Template

Parameters

Options

Result

Note

Example

1.113 STRLEN

Template

Parameters

Options

Result

Note

Example

1.114 SUBSTR

Template

Parameters

Options

Result

Note

Example

1.115 TRANSCRIPT

Template

Parameters

Options

Result

Note

Example

1.116 TACKON

Template

Parameters

Options

Result

Note

Example

1.117 ALL

Template

Parameters

Options

Result

Note

Example

1.118 APPEND

Template

Parameters

Options

Result

Note

Example

1.119 ASSIGNS

Template

Parameters

Options

Result

Note

Example

1.120 CHOICES

Template

Parameters

Options

Result

Note

Example

1.121 COMMAND

Template

Parameters

Options

Result

Note

Example

1.122 CONFIRM

Template

Parameters

Options

Result

Note

Example

1.123 DEFAULT

Template

Parameters

Options

Result

Note

Example

1.124 DELOPTS

Template

Parameters

Options

Result

Note

Example

1.125 DEST

Template

Parameters

Options

Result

Note

Example

1.126 DISK

Template

Parameters

Options

Result

Note

Example

1.127 FILES

Template

Parameters

Options

Result

Note

Example

1.128 FONTS

Template

Parameters

Options

Result

Note

Example

1.129 HELP

Template

Parameters

Options

Result

Note

Example

1.130 INCLUDE

Template

Parameters

Options

Result

Note

Example

1.131 INFOS

Template

Parameters

Options

Result

Note

Example

1.132 NEWNAME

Template

Parameters

Options

Result

Note

Example

1.133 NEWPATH

Template

Parameters

Options

Result

Note

Example

1.134 NOGAUGE

Template

Parameters

Options

Result

Note

Example

1.135 NOPOSITION

Template

Parameters

Options

Result

Note

Example

1.136 NOREQ

Template

Parameters

Options

Result

Note

Example

1.137 OPTIONAL

Template

Parameters

Options

Result

Note

Example

1.138 PATTERN

Template

Parameters

Options

Result

Note

Example

1.139 PROMPT

Template

Parameters

Options

Result

Note

Example

1.140 QUIET

Template

Parameters

Options

Result

Note

Example

1.141 RANGE

Template

Parameters

Options

Result

Note

Example

1.142 SAFE

Template

Parameters

Options

Result

Note

Example

1.143 SETTOOLTYPE

Template

Parameters

Options

Result

Note

Example

1.144 SETDEFAULTTOOL

Template

Parameters

Options

Result

Note

Example

1.145 SETSTACK

Template

Parameters

Options

Result

Note

Example

1.146 SOURCE

Template

Parameters

Options

Result

Note

Example

1.147 SWAPCOLORS

Template

Parameters

Options

Result

Note

Example

1.148 STRPART

Template

Parameters

Options

Result

Note

Example

1.149 NUMPART

Template
Parameters
Options
Result
Note
Example

1.150 Technical information

Here you find some additional information about the Installer. One can find
how the Interpreter itself works or the theoretical aspects of the language.
Have fun ;-)
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
Syntax
Grammar Info

1.151 The interpretation process

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 statements (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.

1.152 The syntax of the language

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.
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.

1.153 Information about the Grammer

For those who are interested in this: 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.