savage-installer

COLLABORATORS			
	TITLE : savage-installer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 14, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

savage-installer

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

Go on

1.2 Savage Installer

Savage-Installer v0.8 beta

THIS IS A PUBLIC BETA VERSION

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

USAGE IS YOUR OWN RISK.

Currently i am writing lots of self-calculating and self-layouting BOOPSI classes.

These classes will also be freeware (look for the SavGUI package) and the next version

of the SavInstaller will use these classes.

So please be patient and don`t be angry with me, when no new SavInstaller will come

up the next weeks :)

Since this is a beta release, please note:

- it needs MUI, but public release will make use of BOOPSI

- writing to "startup-sequence" does currently not work, but writing to

"user-startup" should work ok

- not yet supported: OPTIONAL/DELOPTS for COPYFILES/COPYLIB

- some mui layout errors?

- you may use the serial debug-output for finding uninitialized variables within a

script

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

steps:

1) use the debug version of the SavInstaller (included in this archive) and catch the

serial debug-output of this special debug-version

2) send me a bugreport and please include:

a) the script, which caused the error

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

c) the debug-output of the installer

d) maybe the enforcer-hits etc

Some people don`t like the name "Savage-Installer", but I do :O) Do you have good ideas for other names? ("Installer NG" or "Installer Millenium" or ...)

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 Savage-Installer. 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 coming "developer version" (which includes a source-level debugger, a script creator etc...) will be "Shareware", but this Savage-Installer won`t.
This version 0.8 BETA 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 Savage-Installer into your own releases, 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
E-Mail
jt18@irz.inf.tu-dresden.de
WWW
http://www.inf.tu-dresden.de/~jt18
IRC
Nick: _savage
Channel: #amigager

1.7 I want to thank...

Special thanx for...
patience and testing the SavInstaller again and again
Jens Langner
Sven Steiniger
interesting talks about coding

Michael Rock

Sven Steiniger

telling some secrets of the C= installer

Olaf Barthel

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

Robert Reiswig

Lee Stoneman

Andrea Valinotto

Christian Hattemer

Daniel Confora

Frank Pagels

Henning Kiel

James Maurice Battle

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

1.8 Still not 100% perfect

It is a real hard job to make the Savage-Installer in all parts compatible
to the C= Installer. One reason is the bad language description.
If you want to compile "buggy" scripts, you can switch on the LAZYCOMPILE
Tooltype/CLI-Argument to skip any check procedures!

1.9 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: 43.3! Additional
you can check, whether you run on the Savage-Installer or not by
testing the @savage-installer variable: the C= installer returns
a 0 (zero), but the Savage-Installer holds its version in this
variable.

```
(IF @savage-installer
(
; this savage-installer 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 Savage-Installer 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 Savage-Installer 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))
```

1.10 What ist left to do

Personal aims
- substitute the builtin MUI interface by a gadtools/boopsi interface and put
the MUI gui into a shared library, such that Savage-Installer can optional use
any user interface (which comes as shared library)
- 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
- variable spy, source-level debugger
Suggestions by others
- new tooltypes (DEFAULTSCREEN, DEFAULTBACKGROUND, ...) and some new functions
(SETBACKGROUND, ...) for manipulating the installer gui (Jens Langner)
- something like a "global logfile" which holds all installed
packages (Alasdair Simpson)
- image buttons (Danial Canfora)
- new function for showing images via datatypes (Andrea Vallinotto)
- 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 ?
- saving the compiled program as binary for a later reloading
- use XPK-functions for unpacking
- add constants to the language: (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 :)
Put the complete gui into a shared library (maybe a installergui.library
for gadtools/mui/bgui...) and create executables out of a script. Running
this executable has the same effect as running the installer and a script.
This should reduce the resources in any way.

1.11 How to start the Installer

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

1.12 Running from WB

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

1.13 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

1.14 The history of this Program

0.8 (current...)
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 Savage-Installer 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 SavInstaller 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 SavInstaller runs in debug mode, uninitialized vars and pattern errors are reported
- the SavInstaller 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 SavInstallers "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 SavInstaller 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 Savage-Installer 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 Savage-Installer
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 "@savage-installer" which is 0 iff the script does not run on the
savage-installer, otherwise it is set to the version of the savage-installer
(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.

1.15 The Installer Language

The language used by the Installer is a simple, imperative language. 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 instructrion, executes it, gets the next one, executes 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 function symbol is at first

position, followed by its parameters. Every statement must be enclosed by parenthesis. For example to simply add two numbers you must write: (+ 2 3)
A complete list of all functions and statements you find below. Of course you find everything of a good imperative language: conditional statements, variables, a big set of built-in functions, defining own procedures and more.
A program consists of zero or more statements. A statement, enclosed in parenthesis, consist of zero or expressions. A expression is a number, a string, a identifier or a statement again. But the first expression inside of a statement (the functional expression) must not be a number, but anything else.
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: - procedures/functions which are marked with a * are new in the Savage-Installer
- procedures/functions which are marked with a + provide enhanced functionality
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, statement list) which deliveres a result of type string or number.
SOOP support
GET-PROPERTY *
PUT-PROPERTY *
REMOVE-PROPERTY *
Conditional Statements
IF
UNTIL
WHILE
Procedures
ABORT
BEEP *
COMPLETE
CONCURRENT-DO *
COPYFILES
COPYLIB
DEBUG
DELAY *
DELETE
EXECUTE

EXIT
FOREACH
ICONINFO
LET *
MAKEASSIGN
MAKEDIR
MESSAGE
ONERROR
PROCEDURE
PROTECT
REBOOT *
RENAME
REXX
RUN
SET
SETENV *
SIMULATE-ERROR *
STARTUP
SWING *
TEXTFILE
TOOLTYPE
TRAP
USER
WELCOME
WORKING
Functions
=
<>
>
>=
<
<=
COMPARE *
+
-
*
/
AND
OR

XOR
NOT
BITAND
BITOR
BITXOR
BITNOT
IN
SHIFTLEFT
SHIFTRIGHT
ASKDIR
ASKFILE
ASKSTRING
ASKNUMBER
ASKCHOICE
ASKOPTIONS
ASKBOOL
ASKDISK
CAT
DATABASE +
EXISTS +
EXPANDPATH
EARLIER
FILEONLY
FINDBOARD *
GETASSIGN
GETDEVICE
GETDISKSPACE
GETENV
GETSIZE
GETSUM
GETVERSION
PATHONLY
PATMATCH
RANDOM *
SELECT
STRLEN
SUBSTR
TACKON
TRANSCRIPT

Parameter Functions

ALL

APPEND

ASSIGNS

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

1.16 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 Savage-Installer 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
In this version, Savage-Installer uses the MUI package for the layout. MUI is
(c) by Stefan Stunz. However, I am shure that public releases will not make
use of any GUI-creation libraries, but it will use BOOPSI
Comfortable WB-Start
If you run the Savage-Installer 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 Savage-Installer gets started.
Flexible interpretation
If an error raises while the interpretation process, Savage-Installer 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
@system-language - a string which (hopefully) holds the prefered systems language
(no more dozens of script icons for the different languages!)
@savage-installer - the version of the Savage-Installer
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 Savage-Installer is as lazy as the C= installer
is. that means, Savage-Installer skips its semantic check procedures
to be more compatible
DEBUGMODE: if set, then Savage-Installer will switch on it`s debugmode
CREATEUNINSTALL=CUI: if set, then Savage-Installer 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 SavInstaller can be interupted everytime by sending the CTRL-F signal to its process. If you want to break a sub-process (created by CONCURRENT-DO), you have to signal this sub-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.
Concurrent statements
The function CONCURRENT-DO allows you to interpret several statements at the same time.
SOOP - Simple Object Oriented Programing
With help of the new functions PUT-PROPERTY, GET-PROPERTY and REMOVE-PROPERTY the Savage-Installer 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) statement to the next. When reaching the last statement, the installation may proceed. This looks/works much like the MS-Setup program :)
Full installation control
If you want to, the Savage-Installer 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
CONCURRENT-DO
DELAY
FINDBOARD
LET
RANDOM
REBOOT

SETENV

SIMULATE-ERROR

SWING

GET-PROPERTY

PUT-PROPERTY

REMOVE-PROPERTY

1.17 Known Bugs, not yet removed *sorry*

My own bugs

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

Thats not my fault...

- the "dirlist" class of mui uses wrong aligned fileinfoblocks

- 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

- when using MUI: if the CHOICES labels (see ASKCHOICE,ASKOPTIONS) have an underscore

this is interpreted as "underline a char"

1.18 The Errors

To understand these errors think of the syntactical structure of any program:
A program consists of zero or more statements. A statement, enclosed in
parenthesis, consist of zero or expressions. A expression is a number, a
string, a identifier or a statement again. But the first expression inside of
a statement (the functional expression) must not be a number, but anything
else.

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.

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 a opening bracket must be an identifier or a
string. What you wrote is maybe a number.
) expected
You forgot a ")" ???

1.19 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!

1.20 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

1.21 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

1.22 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

1.23 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 Savage-Installer starts to execute the ONERROR statements
Options
Result
Type: NUMBER
Returns 0
Note
Example
(ABORT "Sorry, I have to quit cause: " #reason)

1.24 COMPLETE

Template
Parameters
Options
Result
Note
Example

1.25 COPYFILES

Template
Parameters
Options
Result
Note
Example

1.26 COPYLIB

Template

Parameters

Options

Result

Note

Example

1.27 CONCURRENT-DO

This creates any number of slave processes, which are running concurrently
and interpreting a related statement list. This can be useful, if you want to
unpack several archives at the same time. The function returns, when all sub-
processes are finished.
Please don`t expect a "power up" of the script-execution. Since an Amiga has only
one CPU, the interpretation does not speed up (more precisely, the execution
can slow down, because of the task-scheduler context). That`s the reason, why I
will not go on with developing this function... (but, for sure, bugfixing it!)
When using CONCURRENT-DO, you must respect some rules:
- don`t use functions, which modify the GUI of the Installer in any way
- don`t use same parameter functions in different processes
- avoid to SET variables with the same name from different processes
- do not nest CONCURRENT-DO
Template
(CONCURRENT-DO <statements>)
Parameters
<statements> a list of statements. for every statement, the installer creates a
new process and this process evaluates the statement. since a
statement can either be a single statement or a list of statements,
you are able to create complex concurrent evaluation processes!
Options
Result
Type: NUMBER
Returns TRUE
Note
This is only for the advanced programmer. Please, be very careful when using
this function :)
An interpretation error of one process causes all other processes to quit.

The processes, created by CONCURRENT-DO, are numbered starting from 1 to n
for n <statements> and called "SavInstaller_SLAVE_..."
Example

```
(CONCURRENT-DO
; first process
(RUN "c:unpack t:arc.1")
; the second process
(RUN "c:unpack t:arc.2")
)
```

1.28 DEBUG

Template
Parameters
Options
Result
Note
Example

1.29 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

1.30 DELETE

Template

Parameters

Options

Result

Note

Example

1.31 EXECUTE

Template

Parameters

Options

Result

Note

Example

1.32 EXIT

Template

Parameters

Options

Result

Note

Example

1.33 FOREACH

Template

Parameters

Options

Result

Note

Example

1.34 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
)
)
)
```

1.35 MAKEASSIGN

Template

Parameters

Options

Result

Note

Example

1.36 MAKEDIR

Template

Parameters

Options

Result

Note

Example

1.37 MESSAGE

Template

Parameters

Options

Result

Note

Example

1.38 ONERROR

Template

Parameters

Options

Result

Note

Example

1.39 PROCEDURE

Template

Parameters

Options

Result

Note

Example

1.40 PROTECT

Template

Parameters

Options

Result

Note

Example

1.41 RENAME

Template

Parameters

Options

Result

Note

Example

1.42 REXX

Template

Parameters

Options

Result

Note

Example

1.43 RUN

Template

Parameters

Options

Result

Note

Example

1.44 SET

Template

Parameters

Options

Result

Note

Example

1.45 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

1.46 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

1.47 STARTUP

Template
Parameters
Options
Result
Note
Example

1.48 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
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)
)
)
)

1.49 TEXTFILE

Template
Parameters
Options
Result
Note
Example

1.50 TOOLTYPE

Template

Parameters

Options

Result

Note

Example

1.51 TRAP

Template

Parameters

Options

Result

Note

Example

1.52 USER

Template

Parameters

Options

Result

Note

Example

1.53 WELCOME

Template

Parameters

Options

Result

Note

Example

1.54 WORKING

Template

Parameters

Options

Result

Note

Example

1.55 EQU

Template

Parameters

Options

Result

Note

Example

1.56 NE

Template

Parameters

Options

Result

Note

Example

1.57 GT

Template

Parameters

Options

Result

Note

Example

1.58 GE

Template

Parameters

Options

Result

Note

Example

1.59 LT

Template

Parameters

Options

Result

Note

Example

1.60 LE

Template

Parameters

Options

Result

Note

Example

1.61 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 Savage-Installer 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

1.62 ADD

Template

Parameters

Options

Result

Note

Example

1.63 SUB

Template

Parameters

Options

Result

Note

Example

1.64 MUL

Template

Parameters

Options

Result

Note

Example

1.65 DIV

Template

Parameters

Options

Result

Note

Example

1.66 AND

Template

Parameters

Options

Result

Note

Example

1.67 OR

Template

Parameters

Options

Result

Note

Example

1.68 XOR

Template

Parameters

Options

Result

Note

Example

1.69 NOT

Template

Parameters

Options

Result

Note

Example

1.70 BITAND

Template

Parameters

Options

Result

Note

Example

1.71 BITOR

Template

Parameters

Options

Result

Note

Example

1.72 BITXOR

Template

Parameters

Options

Result

Note

Example

1.73 BITNOT

Template

Parameters

Options

Result

Note

Example

1.74 SHIFTLEFT

Template

Parameters

Options

Result

Note

Example

1.75 SHIFTRIGHT

Template

Parameters

Options

Result

Note

Example

1.76 IN

Template

Parameters

Options

Result

Note

Example

1.77 ASKDIR

Template

Parameters

Options

Result

Note

Example

1.78 ASKFILE

Template

Parameters

Options

Result

Note

Example

1.79 ASKSTRING

Template

Parameters

Options

Result

Note

Example

1.80 ASKNUMBER

Template

Parameters

Options

Result

Note

Example

1.81 ASKCHOICE

Template

Parameters

Options

Result

Note

Example

1.82 ASKOPTIONS

Template

Parameters

Options

Result

Note

Example

1.83 ASKBOOL

Template

Parameters

Options

Result

Note

Example

1.84 ASKDISK

Template

Parameters

Options

Result

Note

Example

1.85 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

1.86 CAT

Template

Parameters

Options

Result

Note

Example

1.87 DATABASE

Returns information about the AMIGA that the Savage-Installer 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 Savage-Installer 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", "68010", "68030", "68040", "68060", "68040/PPC", "68060/PPC")

"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
<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, <feature> and <checkvalue>
a number; 1 if <checkvalue> equals the result of <feature>, otherwise 0
Note
Savage-Installer 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 @savage-installer
(
(DATABASE "cpu" "(68040|68060)")
)
(
(PATMATCH "(68040|68060)" (DATABASE "cpu"))
)
) --> 1 iff you run on a 68040 or 68060, otherwise 0
```

See also

1.88 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

1.89 EXPANDPATH

Template
Parameters
Options
Result
Note
Example

1.90 EARLIER

Template
Parameters
Options
Result
Note
Example

1.91 FILEONLY

Template

Parameters

Options

Result

Note

Example

1.92 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

1.93 GETASSIGN

Template

Parameters

Options

Result

Note

Example

1.94 GETDEVICE

Template

Parameters

Options

Result

Note

Example

1.95 GETDISKSPACE

Template

Parameters

Options

Result

Note

Example

1.96 GETENV

Template

Parameters

Options

Result

Note

Example

1.97 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

1.98 GETSIZE

Template

Parameters

Options

Result

Note

Example

1.99 GETSUM

Template

Parameters

Options

Result

Note

Example

1.100 GETVERSION

Template

Parameters

Options

Result

Note

Example

1.101 PATHONLY

Template

Parameters

Options

Result

Note

Example

1.102 PATMATCH

Template

Parameters

Options

Result

Note

Example

1.103 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

1.104 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

1.105 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

1.106 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

1.107 SELECT

Template

Parameters

Options

Result

Note

Example

1.108 STRLEN

Template

Parameters

Options

Result

Note

Example

1.109 SUBSTR

Template

Parameters

Options

Result

Note

Example

1.110 TRANSCRIPT

Template

Parameters

Options

Result

Note

Example

1.111 TACKON

Template

Parameters

Options

Result

Note

Example

1.112 ALL

Template

Parameters

Options

Result

Note

Example

1.113 APPEND

Template

Parameters

Options

Result

Note

Example

1.114 ASSIGNS

Template

Parameters

Options

Result

Note

Example

1.115 CHOICES

Template

Parameters

Options

Result

Note

Example

1.116 COMMAND

Template

Parameters

Options

Result

Note

Example

1.117 CONFIRM

Template

Parameters

Options

Result

Note

Example

1.118 DEFAULT

Template

Parameters

Options

Result

Note

Example

1.119 DELOPTS

Template

Parameters

Options

Result

Note

Example

1.120 DEST

Template

Parameters

Options

Result

Note

Example

1.121 DISK

Template

Parameters

Options

Result

Note

Example

1.122 FILES

Template

Parameters

Options

Result

Note

Example

1.123 FONTS

Template

Parameters

Options

Result

Note

Example

1.124 HELP

Template

Parameters

Options

Result

Note

Example

1.125 INCLUDE

Template

Parameters

Options

Result

Note

Example

1.126 INFOS

Template

Parameters

Options

Result

Note

Example

1.127 NEWNAME

Template

Parameters

Options

Result

Note

Example

1.128 NEWPATH

Template

Parameters

Options

Result

Note

Example

1.129 NOGAUGE

Template

Parameters

Options

Result

Note

Example

1.130 NOPOSITION

Template

Parameters

Options

Result

Note

Example

1.131 NOREQ

Template

Parameters

Options

Result

Note

Example

1.132 OPTIONAL

Template

Parameters

Options

Result

Note

Example

1.133 PATTERN

Template

Parameters

Options

Result

Note

Example

1.134 PROMPT

Template

Parameters

Options

Result

Note

Example

1.135 QUIET

Template

Parameters

Options

Result

Note

Example

1.136 RANGE

Template

Parameters

Options

Result

Note

Example

1.137 SAFE

Template

Parameters

Options

Result

Note

Example

1.138 SETTOOLTYPE

Template

Parameters

Options

Result

Note

Example

1.139 SETDEFAULTTOOL

Template

Parameters

Options

Result

Note

Example

1.140 SETSTACK

Template

Parameters

Options

Result

Note

Example

1.141 SOURCE

Template

Parameters

Options

Result

Note

Example

1.142 SWAPCOLORS

Template

Parameters

Options

Result

Note

Example

1.143 STRPART

Template

Parameters

Options

Result

Note

Example

1.144 NUMPART

Template

Parameters

Options

Result

Note

Example

1.145 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

1.146 The interpretation process

The interpreter does it`s job using "call-by-name" strategy. This means it first interprets the symbol at the first position of a statement which results into a function call. The called function then interprets the arguments and uses the results of this argument-interpretation. As you can see this process is recursive.
An example: given the following statements (set i (+ 3 4)) the parser 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.

1.147 The syntax of the language

These are the rules for the parser. Keep this in mind and you always write syntactically correct scripts ;)
<...> means to be a nonterminal symbol and the words written in upper case are the terminal symols (key-words of the language). EPSILON means the "epsilon" symbol, which means, there is no right-hand-side of the rule.
<prog> -> <stmtlist> EOF
<stmtlist> -> <stmt> <stmtlist>
| EPSILON
<stmt> -> (<stmttail>
<stmttail> -> FUNCTION <exprlist> <rparen>
| <firstexpr> <exprlist> <rparen>
<exprlist> -> <expr> <exprlist>
| EPSILON

<expr> -> NUMBER

| <firstexpr>

<firstexpr> -> IDENT

| STRING

| <stmt>

<rparen> ->)

Below you find the EBNF description of the syntax.

<prog> ::= [<stmt>]*

<stmt> ::= "(" <ident> [<expr>]* ")"

| "(" <string> [<expr>]* ")"

| "(" [<stmt>]* ")"

<expr> ::= <number>

| <string>

| <ident>

| <stmt>

1.148 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 function/procedure 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.