CreativE

Tomasz Wiszkowski

COLLABORATORS			
	TITLE : CreativE		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Tomasz Wiszkowski	July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

CreativE

1.1 CreativE

CreativE 2.12.3

The AmigaE compilator

~~~~~
~~~~~

1.2 Introduction

At least after long fights against the difficulties I can bring to You some new stuff to compile Your E programs :). CreativE is an AmigaE compiler which offers You many new improvements and features. The current version seems to have really much less bugs than any other version [it seems to be a good idea - release new versions after heavier tests hehe ;)]. Just take a look what it brings to You 8D.

Please remember that You're using this compiler on Your own risk!. I won't take -no- responsibility of any damage caused by this program.

1.3 Compatibility

String formatting

There is a problem with a "%" sign. EC v3.3a used to placing a normal percent char there. CreativE lets you use C alike format strings (%ld, %s etcetera). That's why you MUST place two signs (%%) to get this character in output string [this bug concerns only functions using formatted output, i.e. WriteF, PrintF, Vfprintf etc.).

1.4 New commands

Alloc()

Chk()

CoerceMethod()

CoerceMethodA()

CtrlD()

CtrlE()

CtrlF()

DoMethod()

DoMethodA()

DoSuperMethod()

DoSuperMethodA()

Eof()

Fclose()

Fopen()

Free()

Get()

GetA4()

Gets()

Lsl()

Lsr()

PutF()

ReadB()

Set()

Sets()

Size()

WriteB()

1.5 Alloc()

Alloc()

SYNOPSIS

mem:=Alloc(size)

FUNCTION

Function allocates POOL memory if it is present.

INPUTS

size - size of memory to alloc

RESULT

mem - pointer to allocated memory or 0 if allocation failed

NOTE

- OPT POOL must be specified

SEE ALSO

Free()

1.6 Chk()

Chk()

SYNOPSIS

bool:=Chk(a)

FUNCTION

Function checks parameter and returns FALSE if it is equal to 0 or TRUE if it's not.

INPUTS

a - variable, expression or anything else to be checked

RESULT

bool - boolean value

SEE ALSO

1.7 CoerceMethod()/CoerceMethodA()

CoerceMethod()/CoerceMethodA()

SYNOPSIS

res:=CoerceMethod(class, object, message, ...)

res:=CoerceMethodA(class, object, message)

FUNCTION

Function invokes the supplied message on the specific object as though it were the specified class

INPUTS

class - pointer to boopsi class

object - pointer to boopsi object

message - method-specific message to be send

RESULT

res - class and message specific result

NOTE

This function is v36+ only!

SEE ALSO

DoMethod(), DoSuperMethod()

1.8 CtrlD()/CtrlE()/CtrlF()

CtrlD()/CtrlE()/CtrlF()

SYNOPSIS

bool:=CtrlD()

bool:=CtrlE()

bool:=CtrlF()

FUNCTION

Functions check for break signals and return TRUE if the signal was received

INPUTS

none

RESULT

bool - holds TRUE if break signal was received

SEE ALSO

1.9 DoMethod()/DoMethodA()

DoMethod()/DoMethodA()

SYNOPSIS

res:=DoMethod(object, message, ...)

res:=DoMethodA(object, message)

FUNCTION

Function invokes the supplied message on the specified object

INPUTS

object - pointer to boopsi object

message - method-specific message to be send

RESULT

res - object and message specific result

NOTE

This function is v36+ only!

SEE ALSO

CoerceMethod(), DoSuperMethod()

1.10 DoMethod()/DoMethodA()

DoSuperMethod()/DoSuperMethodA()

SYNOPSIS

res:=DoSuperMethod(class, object, message, ...)

res:=DoSuperMethodA(class, object, message)

FUNCTION

Function invokes the supplied message on the specified object though as it were the superclass of the specified class

INPUTS

class - pointer to boopsi class

object - pointer to boopsi object

message - method-specific message to be send

RESULT

res - class and message specific result

NOTE

This function is v36+ only!

SEE ALSO

CoerceMethod(), DoMethod()

1.11 Eof()

Eof()

SYNOPSIS

bool:=Eof(fh)

FUNCTION

Function checks if the EOF has been reached

INPUTS

fh - pointer to DOS filehandle structure

RESULT

bool - holds TRUE if the file reached EOF, otherwise it's false

SEE ALSO

Size()

1.12 Fclose()

Fclose()

SYNOPSIS

Fclose(fh)

FUNCTIONS

Function closes file opened previously with Fopen()

INPUTS

fh - filehandle obtained from Fopen()

RESULT

none

SEE ALSO

Fopen()

1.13 Fopen()

Fopen()

SYNOPSIS

fh:=Fopen(name, mode)

FUNCTION

function opens DOS file using standard Open() command and stores the filehandle in global list of filehandles. All the opened files are closed automatically at the end of program

INPUTS

name - name of file to be opened

mode - open file mode

RESULT

fh - filehandle that can be used with any DOS command

SEE ALSO

Fclose(), ReadB(), WriteB()

1.14 Free()

Free()

SYNOPSIS

Free(mem)

FUNCTION

Function disposes memory allocated previously with Alloc() command

INPUTS

mem - pointer to memory obtained from Alloc()

RESULT

none

SEE ALSO

Alloc()

1.15 Get()/Gets()

Get()/Gets()

SYNOPSIS

res:=Get(object, attr, store)

res:=Gets(object, attr)

FUNCTION

Ask specified object for a value assigned to specified attribute

INPUTS

object - pointer to boopsi object

attr - attribute tag id

store - pointer to storage for the answer

RESULT

res - value assigned to specified attribute (Gets);

FALSE if the inquiries of attribute are not provided by the object's class (Get)

NOTE

This function is v36+ only!

SEE ALSO

Set()

1.16 GetA4()

GetA4()

SYNOPSIS

GetA4()

FUNCTION

Restore A4 register

INPUTS

none

RESULT

none

NOTE

- This function doesn't have to be called before use, therefore You can use it only where needed.

- Doesn't work in library mode.

BUGS

None known.

SEE ALSO

1.17 PutF()

PutF()

SYNOPSIS

PutF(fh, formatstr, args...)

FUNCTION

Function writes formatted string to selected filehandle

INPUTS

fh - filehandle

formatstr - C or E alike formatstring

args - list of arguments

RESULT

none

SEE ALSO

1.18 ReadB()

ReadB()

SYNOPSIS

blks:=ReadB(fh, blksize, numblocks, mem)

FUNCTION

This function reads numblocks blocks of size specified in blksize into continuous memory starting at mem

INPUTS

fh - DOS filehandle

blksize - size of one block

numblocks - number of blocks to be read

mem - memory location to store blocks

RESULT

blks - number of read blocks

SEE ALSO

WriteB()

1.19 Set()/Sets()

Set()/Sets()

SYNOPSIS

Set(object, attr, value, ...)

Sets(object, attr, value)

FUNCTION

Assign a value assigned to specified attribute of the object
INPUTS
object - pointer to boopsi object
attr - attribute tag id
value - value to be assigned to the attribute
RESULT
none
NOTE
This function is v36+ only!
SEE ALSO
Get()

1.20 Size()

Size()
SYNOPSIS
len:=Size(fh)
FUNCTION
returns file size
INPUTS
fh - DOS filehandle
RESULT
len - file size
SEE ALSO
Eof()

1.21 WriteB()

WriteB()
SYNOPSIS
blks:=WriteB(fh, blksize, numblocks, mem)
FUNCTION
writes numblocks blocks of size specified in blksize from continuous memory starting at mem
INPUTS
fh - DOS filehandle
blksize - size of one block
numblocks - number of blocks to be read
mem - memory location storing blocks
RESULT
blks - number of written blocks
SEE ALSO
ReadB()

1.22 Lsd()

Lsl() / Lsr()

SYNOPSIS

x:=Lsl(y,s)

x:=Lsr(y,s)

FUNCTION

Performs logical shift left / right

INPUTS

y - variable to be shifted

s - number of shifts

RESULT

x - shifted output

SEE ALSO

1.23 New Keywords

CLEANUP

CODE

CONST

ENDPROC

EXTRA

FOR

FPEXP

INCLIB

INLINE

LINKABLE

LONG

NOSTARTUP

POOL

RUNBG

SETUP

STARTUP

UNION

UTILLIB

WHILE

1.24 LINKABLE

OPT LINKABLE

USAGE

OPT LINKABLE

ABOUT

This option allows You creating linkable object code (.o) instead of normal executable or library one.

NOTE

object has no references specified and therefore might be used only with E

1.25 NOSTARTUP

OPT NOSTARTUP

USAGE

OPT NOSTARTUP

ABOUT

This option lets You use Your own startup code. No libraries are opened and nothing is initialized (except execbase and stack) in Your output code, You have to do everythnig Yourself. It is made especially for those who want to make really small progs ;).

NOTE

arg string is placed in A0, not in the arg variable. You must initialize all the resources You want to use Yourself. Only memory allocated with New() and files opened with Fopen() are closed at the end.

1.26 k0002

OPT POOL

USAGE

OPT POOL (memtype, puddlesize, threshsize)

ABOUT

This switch lets You create pool that can be used later in Your program e.g. via Alloc or anything else. Pool pointer is stored in __pool variable. Parameters are optional, so You can write simply OPT POOL to use it.

NOTE

This is v39+ only!

1.27 UTILLIB

OPT UTILLIB

USAGE

OPT UTILLIB

ABOUT

This switch links utility.library support to Your code. Some utility functions are used by patched commands. All the offsets appear automatically when You simply switch this option on.

NOTE

This is v37+ only!

1.28 INLINE

OPT INLINE

USAGE

OPT INLINE

ABOUT

This option marks some -short- E internal commands to be placed in the code as inline functions. This makes Your programs faster, but also a bit longer. Inline command list can be found here

1.29 k0005

UNION

USAGE

UNION [[a],[b], ...]

ABOUT

Since 2.04 it is possible to UNION some members in object definition; the main rules are:

- All the members that needs to be unified must be placed in "[]"

- Each "[]" repressents one group of members to union

- Union must start with "[" and end with "]".

- All members must be separated with commas (",")

- members that follow each union start after the biggest unioned group

- union declaration may be spreaded into several lines

EXAMPLE

OBJECT a

UNION

[

[

a, b, c

],[

d:INT, e:INT, f:INT

],[

g:CHAR, h:CHAR, i:CHAR

]

]

j

ENDOBJECT

will produce:

(----) OBJECT a

(0) a:LONG

(4) b:LONG

(8) c:LONG

(0) d:INT

(2) e:INT

(4) f:INT

(0) g:CHAR

(1) h:CHAR

(2) i:CHAR

(12) j:LONG

(----) ENDOBJECT /* SIZEOF=16 */

1.30 Lib support

INCLIB

USAGE

INCLIB 'libname', 'libname'...

ABOUT

This stuff allows You to make use of object files.

The main requirement (hehe ;) is to have an "ELIB:" assignment (suggested place: "E:LIB"). For each object two files must exist:

- *.lib - the main LIB file [object]

- *.m - description module

module format is very simple; You can use only PROCs, INLINEs and STARTUPs. Comments are also supported - "//", "->" and "/* */".

To make a .lib file, You need a good assembler [AsmOne is recommended]. The internal variables You want to use must be XREFfed [see: Variables]. To get the proper output, You need to compile Your file and write it as linkable object. To add new functions to Your lib file, simply append new objects to the lib.

PROC

INLINE

STARTUP

Variables

NOTES

- only RELOC32 reloc hunk type is supported

- a small check is performed to find the best entry, when more than one command with the same name is met. It means that if You link two commands - one for OS33 and one for OS39 and set up the OSVERSION to 39, the first one is discarded even if is met first.

BUGS

None found

1.31 PROC

PROC Whatever(x,y,z...)

This works almost exactly same as the EC internal functions (e.g. WriteF). There's only one difference - linked procs don't support streams of parameters. If You want to send a stream, You must use an array of elements. The parameters are stored on stack in reverse order. Please note that the contents of regs D3-D7/A4/A5 MUST be kept, all the other registers may be changed. The result should be returned in D0 [and the second one in D1]. Now - example:

Write sth. like e.g.

MOVE.L 4(A7),A0

MOVE.L 8(A7),(A0)

RTS

Compile and write as link; note that ".lib" suffix is necessary . Now run an editor (ced, ged or whatever) and write sth like:

PROC PutLong(what,where)

and save with same name but different suffix (this time - ".m"). Now write a program, e.g.

INCLIB 'YourLib'

PROC main()

DEF a

PutLong(5, {a})

ENDPROC

This will change the internal "PutLong" function with the new one You wrote.

Big advantage of lib files here is that not whole lib is included; only used parts are linked.

1.32 INLINE

INLINE Whatever(x:Rx,y:Ry,z:Rz...)

INLINE lib function is a quite interesting thing ;). It works almost like PROCs, but the difference is that inlines are placed directly in call-place. Also, the structure is different. The first longword of code must contain the length (in bytes) of code to be copied; Example: Write sth. like e.g.

DC.L END-START

START:

MOVE.L D0,(A0)

END:

Compile and write as link; note that ".lib" suffix is necessary . Now run an editor (ced, ged or whatever) and write sth. like:

INLINE PutLong(what:D0,where:A0)

and save with same name but different suffix (this time - ".m"). Now write a program, e.g.

INCLIB 'YourLib'

PROC main()

DEF a

PutLong(5, {a})

ENDPROC

This will change the internal "PutLong" function with the new one You wrote.

Please note that the first longword determines the size of Your procedure which MUST BE even!

1.33 STARTUP

STARTUP CODENAME

this defines a startup code. The startup code is a code that initializes all the system variables before the main procedure is launched. The name of a startup code must be defined with capital letters. Startups do not take _any_ parameters. This type is a bit dangerous stuff for a novice programmer and hence shouldn't be used by them.

1.34 Variables

V A R I A B L E S

This is the list of accessible variables (LIB level only!). Note that the name parser is case insensitive, so You can write "_DOSBASE" and "_dosbase", which mean the same. Ok, the list. First - A4 offsets, normal code:

_dosbase - dos base

_gfxbase - graphics base

_intuitionbase - intuition base

_utilitybase - utility base

_mathieeesingbasbase - mathieeesingbas base

_mathieeesingtransbase - mathieeesingtrans base

_stdin - standard input

_stdout - standard output

_stdrast - standard rastport

_pool - pool pointer

_arg - argument string

_exception - exception variable

_exceptioninfo - exceptioninfo variable

_wbmessage - workbench message

A4 offsets, startup code - above plus:

_stkret - return stack store - used with CleanUp()

_memlist - memory list

_exitcode - exit code jump pointer

_clireturnvalue - value to be returned to CLI

_bottomstk - bottom stack pointer

_filelist - list of files (Fopen)

_osversion - minimal os version (set by OPT OSVERSION)

_cpuflags - cpu attn flag set

Others:

_stkframe - used for LINK A4 command, size: WORD

_stksize - alloc stack size, size: LONG

_delmem - delegates memory, size: LONG

_setup - the main routine, size: LONG

_a4storage - this is where a4 is stored. Please note You HAVE TO set this in startup code if You want to use the GetA4() cmd; size: LONG

_cleanup - the cleanup procedure; switches on open & close of math libs, console etc. size: LONG

All the A4 vars should be defined with XREF and used as <name>(A4), e.g. "_arg(A4)"

The "LONG" types should also be defined with XREF but used as i.e. "JSR _setup"

The _a4storage is a place the a4 register ptr will be stored at. It means that it needn't be defined with XREF but XDEF instead.

Now the variables that can (or should) be set:

__cpu - minimal cpu that can handle Your source; normally 000; possible settins:

- 0x0,

- 680x0,

- x

(i.e. 020 is same as 68020 and 2)

__fpu - fpu required to handle Your source; normally - FFP libs; possible settings:

- 88x,

- 6888x,

- 0x0,

- 680x0

(i.e. 881=68881, 060=68060)

__mmu - mmu that can handle Your source; normally - no mmu; possible settings:

- 0x0,

- 680x0,

- 851,

- 68851.

the variables mentioned above can be set for both procs and startups.

1.35 Custom Startup Code

OPT STARTUP

USAGE

OPT STARTUP='codename'

ABOUT

Keyword allows using custom startup code. It means that You can use self-made in Your programs. This makes output files shorter/larger than usual; You can i.e. only open the dos.library or does not open anything, just link and call the main routine.

The main code must link a4 register and jump to the setup routine (=call main() procedure). In case when only _setup is called (no _cleanup calls found), no additional routines (math libs, pool etc) are performed; To have it opened You must use _cleanup, too. Minimal startup code:

XREF _STKFRAME

XREF _SETUP

LINK A4,#_STKFRAME

JSR _SETUP

UNLK A4

RTS

This code _does not_ initialize _any_ field, it only calls the main() procedure.

NOTES

-only RELOC32 reloc hunk type is supported

BUGS

None found

1.36 LONG keyword and labels

LONG keyword and labels

Ok, saying not much - it works. It is now possible to use "LONG <label>" sentence in the source, and this will place a longword pointing directly to label in Your source. Also, LONGs, INTs and CHARs support now the whole immediate expressions :)

1.37 FOR loop enhancement

FOR loop enhancement

I managed to let the programmer use dynamic STEP size. This causes speed loss (well...) but allows You using the whole expressions. Please note that expression is calculated ONLY ONCE, not more, so the step cannot change while loop is executed. Therefore if You do sth like

a:=1

FOR b:=0 TO 100 STEP a DO a:=0

the assignment "a:=0" does not affect the loop execution.

1.38 ENDPROC enhancement

ENDPROC enhancement

C H A N G E D

I had to change the meaning of this keyword. I know I shouldn't do it but, as I said here last, it was a BETA feature and it could be changed or removed. Due to some requests, the code does no longer support the system exceptions. The current version can be used as a supervisor part code offering You such features as access to the CPU registers available only in supervisor mode. The form has not changed, so it's still

PROC blah()

...

ENDPROC exp WITH RTE

Procedure can be used later as code pointer for Supervisor(). Please don't blame me for this as it is MUCH more useful than the last version.

1.39 WHILE loop enhancement

WHILE-ENDWHILE loop enhancement

This great idea was brought by Martin Kuchinka - author of PowerD compiler. The WHILE-ENDWHILE loop is expanded with following keywords:

ELSEWHILE

Alternative loop part. Any number of ELSEWHIlE keywords can be used in each WHILE loop. Usage is same as for WHILE:

ELSEWHILE <condition>

with only one exception from the rule. As for now it DOESN'T support "DO" kword (it means: You cannot write ELSEWHILE x DO y as it will cause an error)

Every time the WHILE loop is executed, program searches for a condition that returns true and executes proper part of program. In other case it returns the loop

ALWAYS

This keyword is designed especially to be used together with ELSEWHILE. It declares part of program that will be called every time the loop is executed and at last one condition is true.

Sample code (as short as possible :):

PROC main()

DEF a=10

WHILE a=10

a:=a+1

WriteF('Incrementing "a" by 1\n');

ELSEWHILE a=11

a:=0

WriteF('Setting "a" to zero\n');

ALWAYS

WriteF('Loop is called again now\n');

ENDWHILE

WriteF('a=\d\n',a)

ENDPROC

will return:

Incrementing "a" by 1

Loop is called again now

Setting "a" to zero

Loop is called again now

a=0

NOTES

- The infinite loops are much easier to create :))

- Please don't use ALWAYS keyword without ELSEWHILE

1.40 FPExp-beta

OPT FPEXP

USAGE

OPT FPEXP

ABOUT

Allows generating code based on floating-point unit, so that float calculations speed up (depending on FPU)

NOTE

You also need to specify either the cpu like 040/060 or the fpu like 881/882 to switch this on.

060 FPU is also supported. In this case some commands are replaced

as described below:

FScc D0

EXT.W D0

EXT.L D0

changed to

MOVEQ #-1,D0

FBcc lab

MOVEQ #0,D0

lab: [...]

and

FMOVE.x #y,FPz

to

MOVE.L #y,D0

FMOVE.x D0,FPz

For both 040 and 060 processors special mnemonics are used (FSxxx) to speedup[?] calculations.

1.41 RunBG

OPT RUNBG

USAGE

OPT RUNBG (programname, pri)

PARAMETERS

Parameters are optional. It means that even a single "OPT RUNBG" is enough to activate the option. Programname is a string that contains a name of background task and Pri is it's priority.

ABOUT

This switch allows You to make a program that executes in background (this means that it does not block console and does't have to be RUN any more).

NOTE

- v37+ only

- Although it should work on most OS versions, I don't know if it will on future ones [3.5+] because it fakes the segment structure to avoid disposition of memory the program is located in.

- Please play with the priority carefully. Too high priority may freeze Your system ;)

- Files compiled with OPT RUNBG are NOT debuggable

1.42 Sections

SECTION

USAGE

SECTION type,mem

PARAMETERS

Second parameter is optional. First one decides about the section type [CODE/DATA], second - memory type [none=PUBLIC, FAST or CHIP]

ABOUT

This keyword allows a programmer creating programs splitted up to more than one hunk

NOTE

- OPT LARGE should be set if You want to use it.

- Although SECTION DATA allows storing code, please don't do it.

- it is allowed only in executable files, not in modules

and a small example - don't blame me for it, I want to make it as simple as possible ;)

MODULE 'tools/pt', 'dos/dos'

PROC main()

pt_play({module})

Wait(SIGBREAKF_CTRL_C)

pt_stop()

ENDPROC

SECTION DATA,CHIP

module:

INCBIN 'pt_module'

1.43 SETUP

SETUP

USAGE

PROC blah() SETUP

ABOUT

This feature is very useful for some advanced programmers who need to have some internal stuff initialized before main procedure is executed. All the normal E code is allowed in such procedures and these can also be called in the middle of the normal code.

NOTE

- Such procedure CAN NOT ask for any arguments because none are passed to it.

- This kind of procedure may be declared ONLY in modules.

1.44 CLEANUP

CLEANUP

USAGE

PROC blah() CLEANUP

ABOUT

This one is made for advanced programmers who need to make some additional code for module which has to be executed after the main program quits. Normal E code may be used in such type of the procedure and it might be called wherever in the normal code as a procedure.

NOTE

- Such procedure CAN NOT ask for any arguments because none are passed to it.

- This kind of procedure may be declared ONLY in modules.

1.45 EXTRA

EXTRA

USAGE

LIBRARY name,verstr,ver,rev EXTRA size IS...

PARAMETERS

name is the library name

verstr is the version string

ver is the library version

rev is the library revision

size is the size of extra space in the library base. This number must be in range 0 - 32000 and also must be even.

ABOUT

Keyword is used to get the extra space in the library base. You can define it if You want to use the base as Your own structure and store sth in it.

NOTE

- some internal datas are stored behind this space! Don't write anything there!

EXAMPLE

MODULE 'exec/libraries', 'exec/execbase', 'exec/nodes'

OBJECT libbase

lib:lib

long

ENDOBJECT

LIBRARY 'example.library', 'example.library 1.0 (07.03.2k)',

1, 0 EXTRA 4 IS a,b

DEF base:PTR TO libbase

PROC main()

base:=FindName(execbase.liblist, 'example.library')

ENDPROC

PROC a(text) IS base.long:=text

PROC b() IS EasyRequestArgs(0,

[20, 0, 'Request', 'Text: \s', 'Ok'], 0,

[base.long])

To see how does this program work, You'll need to write two clients, first one to call function "a" with some text, second - "b" with no args.

1.46 CODE

CODE

USAGE

OPT CODE FAST

OPT CODE SMALL

ABOUT

This switch selects the different types of output code that is to be generated; although CODE SMALL does nothing at this time [normally the output is optimized nowadays ;)], OPT FAST generates the code a bit longer, but uses only the fastest routines [mostly in multiplication - from 2 to 30 optimized at this time]. If You want to get the fastest code for new machines, please specify also processor [020+], FPEXP mode, osversion and INLINE. This should speed the output code really much.

NOTE

1.47 CONST

CONST

USAGE

CONST X=<val>

CONST X=<string>

ABOUT

Since 2.12 it is possible to declare string constants. A string constant is parsed as normal E string, therefore null-chars are also allowed. String constants may be used as normal strings and can be exported to module.

NOTE

1.48 New variables

utilitybase

points to utility.library if it was opened (see UTILLIB)

__pool

points to internal pool, if it was created (see POOL)

1.49 New constants

Constant Value

TAG_DONE 0

TAG_END 0

TAG_IGNORE 1

TAG_MORE 2

TAG_SKIP 3

TAG_USER $80000000

OFFSET_BEGINNING -1

OFFSET_CURRENT 0

OFFSET_END 1

READWRITE 1004

1.50 New operators

//

&

||

=>

=<

>>

<<

*=

==

1.51 //

The "//" sequence defines a short comment in the source [just like the "->"]

1.52 & and ||

These two sequences can be used as replacement of two keywords: "&" means "AND" and "||" means OR

1.53 => and =<

Thesetwo sequences means the same as ">=" ("=>") and "<=" ("=<") ;)

1.54 >> and <<

These two work a bit more like Shr and Shl, but no function is called. Rotation is placed directly in the code so it is much faster. Works exactly as in C/C++. Example source:

PROC rotate_left(v,n)

DEF x

x:=v<<n // here the v is shifted left n times

ENDPROC x

1.55 *= etc

That works mostly same as in C. You can easily omit the sequence such as

x:=x....

and start it without writing the above. It is especially useful when You want to perform an action on some long-named fields in Your object identifier or the variables with quite big names. The main sequence is:

<function>=

where <function> may be one of those: +, -, *, /, AND, &, OR, ||, <<, >>

example use:

DEF object:objectname

...

PROC increase(value)

object.member+=value

// same as: object.member:=object.member+value

ENDPROC

1.56 quick compare

This sequence lets You compare a single expression against any bounds or values [or even expressions, if needed] in a very fast way. The main use is:

<exp> == [<exp>, <lower> TO <upper>,...]

What is a good thing about it? Well, I think it's the speed. The output code is much shorter and the check procedure exits as soon as one comparison is true [for example, if an array has 100 entries and the first entry matches the given expression, it exits immediately with TRUE]. It might be very useful together with IF sentence.

Example use:

PROC checkrange(value)

IF value*16-2 == [2, 10 TO 20, xyz()-blah+3] THEN

DisplayBeep(NIL)

ENDPROC

is equivalent to:

PROC checkrange(value)

IF (value*16-2=2) OR ((value*16-2>10) AND (value*16-2<20)) OR

(value*16-2=(xyz()-blah+3)) THEN DisplayBeep(NIL)

ENDPROC

1.57 LONG strings

\x

This lets You insert any value into your "LONG" string '\x' MUST be followed by two HEX digits describing the ascii number You want to put instead of \x. Function will cause an error if You write something wrong. Please note you ALWAYS have to put TWO digits, even if the whole number fits in one. You can't use signs here!

1.58 Normal strings

\x

This lets You insert any value in string. "\x" MUST be followed by two digits describing HEX number which is the number of ASCII char You want to put there. Compiler returns an error if You do something wrong.

\!

This inserts a BELL ($07) char to Your string. It causes the screen-flash (DisplayBeep()) when put on console.

\v

This inserts a vertical tab ($0B)

1.59 Formatted I/O functions

\u

This puts unsigned decimal number. This is equal to %lu (RawDoFmt)

1.60 Assembler part

because of the number of added commands, I have put only the most-important informations about improvements here. Sorry, folx.

- No more "weird" operand sizes (like RTS.x or MOVE.S)

- Multiplication and division can operate on longs (020+)

- Quite big instruction set - support for CPUs (68k family),

FPUs and MMUs.

- Over 400 assembler commands

- Support to extended addressing mode with suppressed regs

- Lib offset access from assembler (i.e. MOVE.L #Open,D0)

- EA constructor fixed (no more D0.W.L.W.L.W.W.W etc)

- Scale factor added for some 020+ addressing modes

- CCR and SR access added

- xx(Ax) = (xx, Ax) (same for other similar addressing modes)

Not supported commands:

-Pack

-Unpk

-Cas

-Cas2

-Chk2

-Cmp2

-CallM

-RtM

-Bgnd

Short note about PPC - CreativE understands all the PPC mnemonics except Altivec; no matter of this all, PPC support has not been fully added to current release of CreativE, therefore don't expect it to generate PPC executable; moreover, PPC support is planned in future releases of CreativE; I'd like to thank some ppl for offering their help in betatesting.

Thanks a lot!

1.61 The Patcher

The patcher changes some commands to generate the code optimized for different cpus/fpus/osversions and others; Currently those functions are patched

Kickstart

WriteF() [37+]

PutF() [37+]

PrintF() [37+]

Please note that the patcher will be useless soon...

1.62 I wish to thank to...

It's a list of people I wish to thank

Wouter van Oortmerssent

... for writing the best programming language ever!

Dietmar Eilert

... for GoldEd - the greatest editor ever made!

Martin Kuchinka

... for his great ideas, betatesting and emails :)

Mauro Fontana, Rainer Müller

... for staying alive as a beta testers

Also, thanks to

- all people mailing me =)

- all E developers

- all Amiga users

Amiga, forever!

1.63 Hi, it's me! :)

Ok, shortly: feel free to email me, snail me or even call me. Here are my addresses (snail and email ofcourse :)

snailmail:

Tomasz Wiszkowski

Katowicka 23/4

44-335 Jastrzebie Zdroj

POLAND

email:

error@albedo.art.pl

phone:

+48-103332-471-23-21

Any new ideas? Write to me, too!!!

That's all! Enjoy using CreativE!

1.64 E Module Viewer

Well now, I wrote this little toy far away ago, but since I suffer from Alzheimer disease, I forgot to put it inside the archive ;). This time I won't make the same mistake.

ShowMod is a little tool that should help You to inquire any E module [this also includes E modules] and view them in a gadtools window. I decided to use gadtools because every amiga [I suppose] has at least OS 2.0, thus everyone should be able to use it. Now a small description on usage.

At the beginning a small window should appear containing only two gadgets - New and Quit. As You may guess "New" opens a new windows and loads a module, while "Quit" exits program ;). There is no way to quit unless all the clients are closed. Number of clients is limited by free memory only, thus You may open more than one module at once.

Client window has a small gui built from two lists [hmm] and two gadgets; First list shows available tags in module [i.e. constants, procedures and so on], second list is to view contents of each tag. I believe the gadgets placed below the lists [Open and Quit] don't have to be introduced. If You don't know what these do, check it Yourself :).

1.65 PreProcessor

List of preprocessor commands I have added to CreativE:

#date

1.66 #date

This preprocessor keyword is very useful when You want to place compilation date of Your program. It's very easy to use

#date store fmtstring

Ok. This macro keyword allows You to insert current date in any type You wish. The format is defined in format string. Supported keys:

%d - day nr

%m - month nr

%y - year nr (4 digits)

%D - day (name)

%M - month (name)

%Y - year nr (2 digits)

%aD - day (abbreviated name)

%aM - month (abbreviated name)

So, if You want a e.g. version string, You can write sth like:

#date Version '$VER: SpaceSheep v0.0 (%d.%aM.%Y) by LittleGreenMan'

and put it somewhere in your source... the only thing You must take care of is the version ;).

1.67 Common problems

These are the problems You may have when compiling Your source with CreativE. Those are not the bugs :)

Q: When I compile my source with CreativE, the compiler crashes my system after writing the executable

A: Please increase Your stack size. Usually 20k should be enough, but for bigger programms [200k+] You may find it too small.

Q: Compiler complains about stacksize. Why did You make this restriction?

A: Well, CrtvE is very stack-consuming stuff :(. In some cases even 20k isn't enough for it. Please remember to keep an eye on Your stack!

1.68 New error and warning messages

A short description of error mesages and warnings...

unknown HEX value after \x

value expected

')' expected

even number expected

you need a newer OS for this

unable to open resource

this instruction needs a newer OS version (see OPT)

illegal size

fpu register expected

':' expected

mmu register expected

control register expected

cpu register expected

this instruction works only in pool mode

not allowed in library mode

a4 storage not defined in startup code

only RTR and RTE allowed

illegal scale factor

address/pc register expected

value does not fit in 16 bit

value does not fit in 8 bit

address/data register expected

reg args not allowed in methods

illegal section definition

680x0 code not allowed in powerpc program

powerpc code not allowed in 680x0 program

general purpose register expected

unknown/illegal mnemonic

condition register expected

ppc floating point register expected

fp condition register expected

special purpose register expected

time base register expected

better CPU/FPU/MMU/OSVERSION required

stack too small; need at least 20000 bytes

W A R N I N G S

040/060 emulated instruction(s) used

1.69 Unknown HEX value after \x

unknown HEX value after \x

PROBLEM:

This error will appear every time when You use a "\x" sentence in Your string with incorrect HEX number (e.g. "\xZG" etc). Please note "\x" always eats TWO characters, not one.

HELP:

Check Your strings for an incorrect "hex" numbers

1.70 value expected

value expected

PROBLEM:

Error appears every time You use e.g. a variable when a value is expected, e.g. when You use a variable describing extra place in Your library definition

HELP:

Simply place an immediate value or constant in such place.

1.71 ')' expected

")" expected

PROBLEM:

You've probably forgotten to close a bracket :)... most commands does not support it, yet, but they surely will. Example: GetA4(

HELP:

Put a closing bracket

1.72 even number expected

even number expected

PROBLEM:

An even number is required. This pops up especially when You try to create a library space with odd number of bytes

HELP:

Round Your value to the nearest even number

1.73 you need a newer OS for this

you need a newer OS for this

PROBLEM:

You have probably used a compiler option which is not supported by Your operating system. This message will usually pop when You're tring to use some compiler ops (e.g. #date) on a pre-2.0 operating system

HELP:

Well... You must try to remove the object that causes this error or map Your ROM/buy a new kickstart

1.74 unable to open resource

unable to open resource

PROBLEM:

CreativE is unable to open a required resource (e.g. battclock.resource to use with #date preprocessor macro)

HELP:

Reboot Your amiga and try again

1.75 This instruction needs a newer OS version (see OPT)

this instruction needs a newer OS version (see OPT)

PROBLEM:

The command(s) You've used are not supported by the system you're compiling a program to.

HELP:

change OSVERSION setting (e.g. OPT OSVERSION=37)

1.76 illegal size

illegal size

PROBLEM:

The assembler size is not supported by mnemonic (e.g. ADDA.B)

HELP:

fix up the size or remove it.

1.77 fpu register expected

fpu register expected

PROBLEM:

The assembler mnemonic requires at last one FPU register (usually all Fxxxx do) or at last one FPU control register

HELP:

Check out the command syntax

1.78 ':' expected

":" expected

PROBLEM:

a colon is required

HELP:

put a colon in a proper place

1.79 mmu register expected

mmu register expected

PROBLEM:

The assembler mnemonic requires at last one MMU register (usually all Pxxxx do) or at last one MMU control register

HELP:

Check out the command syntax

1.80 control register expected

contol register expected

PROBLEM:

(usually appears with MOVEC mnemonic) - the assembler command requires at last one control register

HELP:

Check out the command syntax/put a control register in a proper place

1.81 cpu register expected

cpu register expected

PROBLEM:

The mnemonic You've used requires a CPU register

HELP:

Check out the command syntax/put a control register in a proper place

1.82 this instruction works only in pool mode

this instruction works only in pool mode

PROBLEM:

You've used probably an instruction which needs a pool to be present (e.g. Alloc())

HELP:

add a POOL keyword to OPT settings

1.83 not allowed in library mode

not allowed in library mode

PROBLEM:

The instruction You've used does not work in library mode (e.g. GetA4())

HELP:

Remove or replace the instruction.

1.84 a4 storage not defined in startup code

a4 storage not defined in startup code

PROBLEM:

You have probably used GetA4() command together with some "custom" startup routine, which hasn't got defined a field storage for A4 register

HELP:

If the startup code was made by You, You should provide extra space for A4 register in it and also store it. Otherwise, You should either use other startup routine or avoid using "GetA4" command.

1.85 only RTR and RTE allowed

only RTR and RTE allowed

PROBLEM:

The keyword You used requires You to chose only between RTR and RTE (as for now - only ENDPROC WITH xxx)

HELP:

simply fix up the keyword.

1.86 illegal scale factor

illegal scale factor

PROBLEM:

You've set a wrong scale factor in Your adressing mode

HELP:

Check if it's one of [1, 2, 4, 8]. These are only possibilities

1.87 address/pc register expected

address/pc register expected

PROBLEM:

The adressing mode You've used requires either Ax or PC register

HELP:

Check up the adressing mode and put the reg in a proper place

1.88 value does not fit in 16 bit

value does not fit in 16 bit

PROBLEM:

The value/offset You've used does not fit in 16 bit

HELP:

fix up the value/offset

1.89 value does not fit in 8 bit

value does not fit in 8 bit

PROBLEM:

Value/offset You've used does not fit in 8 bit range

HELP:

fix the value/offset up

1.90 address/data register expected

address/data register expected

PROBLEM:

The adressing mode You've used requires adressing or data

register

HELP:

Check up the adressing mode and put the reg in a proper place.

1.91 reg args not allowed in methods

reg args not allowed in methods

PROBLEM:

You're asking for arguments stored in regs for method i.e. PROC blah(a=D1) OF x

HELP:

Please fix up the PROC definition. You cannot ask for regs.

1.92 illegal section definition

illegal section definition

PROBLEM:

Your section definition is not declared properly, i.e.

SECTION CHIP,CODE

HELP:

Read the Section keyword documentation

1.93 680x0 code not allowed in powerpc program

680x0 code not allowed in powerpc program

PROBLEM:

You're trying to use a MC680x0 code in powerpc executable

HELP:

Replace the mnemonics with PPC code

1.94 powerpc code not allowed in 680x0 program

powerpc code not allowed in 680x0 program

PROBLEM:

You're trying to use powerpc code in 680x0 executable

HELP:

Replace the mnemonics with 680x0 code

1.95 general purpose register expected

general purpose register expected

PROBLEM:

You've probably missed some general purpose register in powerpc code

HELP:

locate the correct place and put there a proper register

1.96 unknown/illegal mnemonic

unknown/illegal mnemonic

PROBLEM:

You’re trying to use unknown form of powerpc mnemonic [i.e. STWCX]

HELP:

Take a look at the powerpc developers documentation :)

1.97 condition register expected

condition register expected

PROBLEM:

The instruction You’re trying to use requires the condition register

HELP:

Locate the proper place and put the condition register

there

1.98 ppc floating point register expected

ppc floating point register expected

PROBLEM:

A powerpc floating point register is missed

HELP:

Locate the proper place for a floating point register

1.99 fp condition register expected

fp condition register expected

PROBLEM:

This command requires a fpu condition register [to be fixed soon]

HELP:

place the correct condition register in a correct place

1.100 special purpose register expected

special purpose register expected

PROBLEM:

The instruction You’ve used takes a special purpose register as an argument

HELP:

Please place the correct special purpose register in a correct place

1.101 time base register expected

time base register expected

PROBLEM:

The instruction You've used needs a time base register

HELP:

Put the correct time base register in a correct place

1.102 better cpu/fpu/mmu/osversion required

better cpu/fpu/mmu/osversion required

PROBLEM:

Ehhm.. This error appears only when using some lib (.o) files with set some special flags [__cpu, __fpu, __mmu or __osversion]

HELP:

Please try to set up the proper values for each of these.

1.103 stack too small; need at least 20000 bytes

stack too small; need at least 20000 bytes

PROBLEM:

You need a bigger stack to use CreativE; Your stack is too small

HELP:

Use the CLI "STACK" command to set the stack size to 20000 or more bytes. CreativE needs at least 20k for safe compilation.

1.104 040/060 emulated instruction(s) used

040/060 emulated instruction(s) used

PROBLEM:

You've used some emulated instructions in Your assembly code; it means that Your programm will work slower on 040/060 CPUs

1.105 Inline commands

Since 2.01 I've enhanced E with Inline commands. It means that such commands are NO LONGER called (with BSRs); they're placed directly in place of call, so work as fast as if were a simple "+" or "*" in the expression. To use the inline commands please look at description of INLINE keyword. Patched are:

- Abs

- And

- Car

- Cdr

- Char

- CleanUp
- Div (020+)
- Eor
- Eval
- Even
- Fabs
- Facos
- Fatan
- Fasin
- Fceil
- Fcos
- Fcosh
- Fexp
- Ffieee
- Ffloor
- Flog
- Flog10
- Fpow
- Fsin
- Fsincos
- Fsinh
- Fsqrt
- Ftan
- Ftanh
- Ftieee
- Int
- Lsl
- Lsr
- Long
- Mod (020+)
- MouseX
- MouseY
- MsgCode
- MsgIaddr
- MsgQual
- Mul (020+)
- Not
- Odd
- Or

- PutChar
- PutInt
- PutLong
- RndQ
- Shl
- Shr

1.106 Other things

Yes... Well, this chapter contains only stuff I don't know where to put... hehe q:)c=

Object members

Another IF format

Modules vs inlines

Expressions swap

ARRAY declarations

SIZEOF and vars

Multiple pointers

Typed variables

CONST declaration

Vars declarations

1.107 Member

Object members

Since 2.01 it is possible to make an assignment to any object's member inside the immediate list. It's now possible to perform this:

a:=[b[c].d:=e]:x

1.108 Another IF expression format

Another IF format

I'm sure that many ppl will blame me for using C/C++ features in E compiler but I think that if there's a way to do sth easier, it should be implemented.. Ok, and here it is. The format is:

<exp> ? <texp> : <fexp>

Of course this feature can be a bit deeper (read: you can use one in another, like brackets). <exp> stands for any expression to be checked, <texp> stands for expression to be calculated in case of true and <fexp> is the one to be calculated in case of false.

1.109 Modules vs inline code

Modules vs inline code

Yes... this was the main problem the programs compiled with INLINE option were crashing because of. Now (hehe... a bit too late) the compiler is a bit smarter - checks whether the inline code is used by the module to encounter whether copy it or not.

1.110 Expressions swap

Expressions swap

I think it’s useful feature ;). Imagine You can calculate at once two expressions and store them in specified variables later. It means that i.e. You don’t have to define additional vars for temporary use; now:

<exp1>:=:<exp2>

will solve this problem. The results will be stored in the last-used variables (i.e. in a+b/c*d, "d" is the last one). Multiple use of it is possible but the variables are set after each two (i.e. when You write sth like:

a:=:b:=:c

first a is swapped with b and then b is swapped with c)

I know it’s a bit weird but it’s very difficult to explain it. Maybe a little example:

PROC main()

DEF a=1, b=10, c=2

WriteF(’a=\d, b=\d, c=\d\n’, a, b, c)

a+15:=:b/2:=:c*2

WriteF(’a=\d, b=\d, c=\d\n’, a, b, c)

ENDPROC

will produce:

a=1, b=10, c=2

a=5, b=4, c=16

so, first a+15 is calculated (=16) and put on stack; then b/2 is calculated

and stored in "a". Then, b is set with value from stack and another swap

is done.

IMPORTANT NOTE!

This "thing" is only for variables, NOT for objects and members. SO! If you

do sth. like:

a.b.c:=:d.e.f

then ONLY POINTERS are set with expression values! In this case it is equal

to sth. like:

t:=a

a:=d.e.f

d:=t.b.c

1.111 ARRAY declarations

ARRAY declarations

Hence I got several mails about it I decided to allow programmer using the faster way to declare arrays. Since 2.10 it is possible to skip the "ARRAY OF" keyword set and for CHAR - even everything. To declare an array You can write simply:

DEF x[100]:LONG

to get an "ARRAY OF LONG".

1.112 SIZEOF and variables

SIZEOF and vars

Hmm.. Have You ever used "SIZEOF" keyword? It becomes very awkward in situations when You need a size of an object_with_very_long_name. So... here's the way to do it a bit different. Example:

MODULE 'dos/dos'

PROC main()

DEF a:fileinfoblock

WriteF('FileInfoBlock size: \d\n', SIZEOF a)

ENDPROC

will output

FileInfoBlock size: 260

As You can see, it is now possible to use the variable to obtain object's size. Now some technical info:

-. You can ONLY get the destination object's size (i.e., when You declare a as ARRAY OF fileinfoblock, You'll always get "260", no matter how many fields are in the array

-. You won't get the size of an identifier (I mean: declaring "a" as a LONG, You won't get "4" as a SIZEOF value, because "a" is also same as PTR TO CHAR (=you'll get "1" instead of "4"). Now, if You specify "a" as PTR TO INT, the result will be "2", not "4" (as this is the destination object's size).

1.113 Multiple pointers

Multiple pointers

Yess! At last :). I had a couple of emails about multiple pointers (PTR TO PTR TO ... <object>). Although it's added as a beta feature, it should work in most cases (if not in all). Ok, some source code to show the use in practise:

PROC main()

DEF args:PTR TO PTR TO PTR TO CHAR,b=0

ReadArgs('FILES/M', args, NIL)

WHILE a[0][b]<>NIL

WriteF('\s\n',a[0][b])

b++

ENDWHILE

ENDPROC

Hmm, yes, this source also shows the exceptions. There is only one for three identifiers: CHAR, INT and LONG. Let me give You an example: when You declare a variable as a PTR TO CHAR, You won't access the pointer typing a[x], because "a" already points to char (so You can only access the field in a CHAR stream). It's same here. Declaring the variable as a PTR TO PTR TO PTR TO CHAR (as above) is equivalent to: first branch of tree is a LONG pointing to another, which also is LONG and points to the last one which is CHAR typed.

Now, some words about the use IRL...

In objects the situation becomes a bit handy. As You might have noticed, in pointers the objects are stored in one stream (one just after another). So the use of variable declared as below

DEF blah:PTR TO PTR TO oblah

must look like

blah[x][y].field

and will give You the Y'th stream object's field. Note it is a stream (!), just like an array.

and there is no way to skip the second index ([y]). This will change in the nearest future.

NOTES:

-. The multiple pointers cannot be typed (::) in the middle of the code (to be fixed)

-. In arrays, the first (highest) stream is allocated automatically but the rest IS NOT (!). Same to pointers. Writing code like:

PROC main()

DEF a:PTR TO PTR TO PTR TO LONG

a[0][0][0]:=3

ENDPROC

is illegal (writing to unknown memory region) and may cause system crashes.

1.114 Typed variables

Typed variables

I know it was a bit stupid to use another variables to access i.e. the execbase field and type it as pointer to execbase. Therefore I made a simple thing that will allow You to use these variables (execbase, intuitionbase, gfxbase, wbmessage and dosbase) as a typed variables, all You have to do is to include the modules such as 'exec/execbase', 'intuition/intuitionbase' etc. etc., depending on what do You want to have.

1.115 Constants declarations

Constants declarations

Since 2.12 it is possible to use the following signs in the constants assignments:

<< - logical shift left

>> - logical shift right

1.116 oth00000b

Variables declaration

A small thing I've been asked for several times. Well, since 2.12 You can declare Your variables nearly everywhere inside proc, but You have to remember that DEF cannot be covered by anything but PROC and ENDPROC [what I mean is: You cannot define any variable in WHILE/ENDWHILE loop and so on].

A few more words about it: Each variable declared that way is accessible from anywhere in a proc, even before it was declared, but it doesn't mean that it is initialized. What's the difference? Type down this program:

PROC main()

...

WriteF('a equals to: \d\n', a)

DEF a=3

...

ENDPROC

As You may see, a holds some random value, because it gets initialized in place of declaration. It means, that if You change that program to this:

PROC main()

...

DEF a=3

WriteF('a equals to: \d\n', a)

...

ENDPROC

You will get the proper value, I mean: 3.

The same thing happens whenever a variable is a structure or array. Such variable is a pointer to the structure till it is initialized, that is, if You write sth like:

PROC main()

...

z[4].blerk:=33

DEF z[100]:ARRAY OF blah

...

ENDPROC

Your computer will probably crash. You must remember that "z" becomes an array of blah after the declaration, so the proper version is:

PROC main()

...

DEF z[100]:ARRAY OF blah

z[4].blerk:=33

...

ENDPROC

That's all, I suppose ;)

1.117 CreativE's History

2.12.3

2.12.2

2.12

2.10

2.06

2.05

2.03

2.02

2.01

2.00

1.118 CreativE v2.00

- Assembler for all the cpu's (010-060) except ~8 commands
- Enhanced string format codes
- Enhanced "long" strings format codes
- New keywords
- New variables
- New commands
- New error reports
- New warnings
- New expression components
- Patcher

1.119 CreativE v2.01

- Inline commands! Faaaaast! 8D
- x<<a.b now works correctly
- Patcher (expanded)
- Enhanced library declaration ("EXTRA" field)

1.120 CreativE v2.02

- Few optimizations
- Expressions generated for 020+ (partially)
- Few fixes
- New IF statement format (<exp> ? <exp> : <exp>)

1.121 CreativE v2.03

- Pointer typing for variables (e.g. execbase::execbase.thistask)
- Val() and CleanUp() work as they did
- Enhanced inline instruction list
- added "INLINE" keyword

1.122 CreativE v2.05

- Smarter version -> determines if the "inline" code is used by module(s)
- Unions in objects!!! At last!
- "lib" files support! began
- expression swap (":=:")

1.123 CreativE v2.06

- Extended EA support! At last!!!
- custom STARTUP code support
- LONG <label> now possible
- Reloc hunks in lib files
- variable access from lib files
- FOR loop now support expression for STEP
- CCR/SR access from assembler
- arrays, object members assignments inside array
- extended array/object member access (+=, &=...)
- pc relative access with direct values (e.g. 10(PC))

1.124 CreativE v2.10

- One can use the suggested by Motorola adressing modes
- FOR loop now determines the type of params - faster than Wouter's
- ELSEWHILE/ALWAYS (WHILE loop enhancements)
- FPU can be used to calculate the floating-point expressions
- "ARRAY OF" can be omitted (DEF a[100]:LONG)
- PROCs now can get params via registers
- LIBRARY mode fixed (bug in lib compiled w. EXTRA keyword)
- "IF IF" sentence fixed
- "ENDPROC WITH" sentence (for system exception handlers)
- A couple of new error messages
- Fixed a small bug in EA creator (D0.W.W.W.W.L etc)
- Scale factors in EAs
- Object members can be now odd-sized (i.e. a[3]:CHAR)
- Changed moduleversion to 11 because of some module-improvements
- SIZEOF <var> now possible
- Fixed a bug in a few commands (i.e. Fatan())
- Multiple pointers - beta feature
- Variable typing (::)
- Object files are created properly now
- Sources released in another archive

1.125 CreativE v2.12

- Hunk support!!! YEEEAA!
- Added the RUNBG option for use with executables
- Internal variables became typed
- Fixed some bugs in OPT FPEXP
- Another fixes made to the module format
- Shifts may be used in constants declaration
- AmigaE support FTP site organized
- OPT OPTI generates the RELOC16 hunk
- "==" for checking expression with multiple values/ranges etc.
- Creative needs at least 20k of stack [!!!]
- SETUP and CLEANUP procedures for modules
- linklibs support fixed
- "/" and "*" produces the DIVS.L and MULS.L for 020+
- some various fixes/improvements
- variables can be defined nearly anywhere in proc
- Some new commands added
- Heavy optimizations done - output code is shorter than in Wouter's version in some cases :)
- String constants introduced.
- Extended interim buffer - most weird crashes shouldn't appear anymore

1.126 CreativE v2.12.2

- Fixed: "Linking arrays" does no longer appear when compiling with QUIET parameter
- Fixed: Immediate shifts (<< and >>) use "MOVEQ" when shift count is within <-128;127>
- Fixed: FOR loops are no longer executed when initial parameters are already out-of-range
- Fixed: Problem with storing negative numbers on stack does no longer appear
- A few more or less important fixes have been made.

1.127 CreativE v2.12.3

- Fixed: EXIT now works properly with FOR loops
- Fixed: #date prep keyword doesn't cause enforcer hits
- Fixed: additional optimization reduced in order to make conditional loops work properly
- A few more or less important fixes have been made.