allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

BASIC Manual

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Contents

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Overview

This is the online manual of CPBasic, the BASIC interpreter of allpccloud. The aim of CPBasic is to provide an easy-to-use platform for writing macros and small applications which are needed in a configuration process. Many features are included by CPBasic that enable users to build powerful programs which have important functionality.
CPBasic runs in a sandbox on the allpccloud server side. This means the source code is also stored on the server and cannot be accessed by a third party. It always runs in a container with limites resources to prevent the web service from being blocked by the interpreter.
CPBasic has limited file I/O capabilities which are also constrained by the server sandbox.

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Getting started

The first program

We start with a very simple CPBasic script which is the pendant to the famous ``Hello, world!"-introduction in C. Create a macro plugin and insert the following text:

```
print("Hello, world!")
```

To execute the program, submit the source code (Upload and run button, blue).

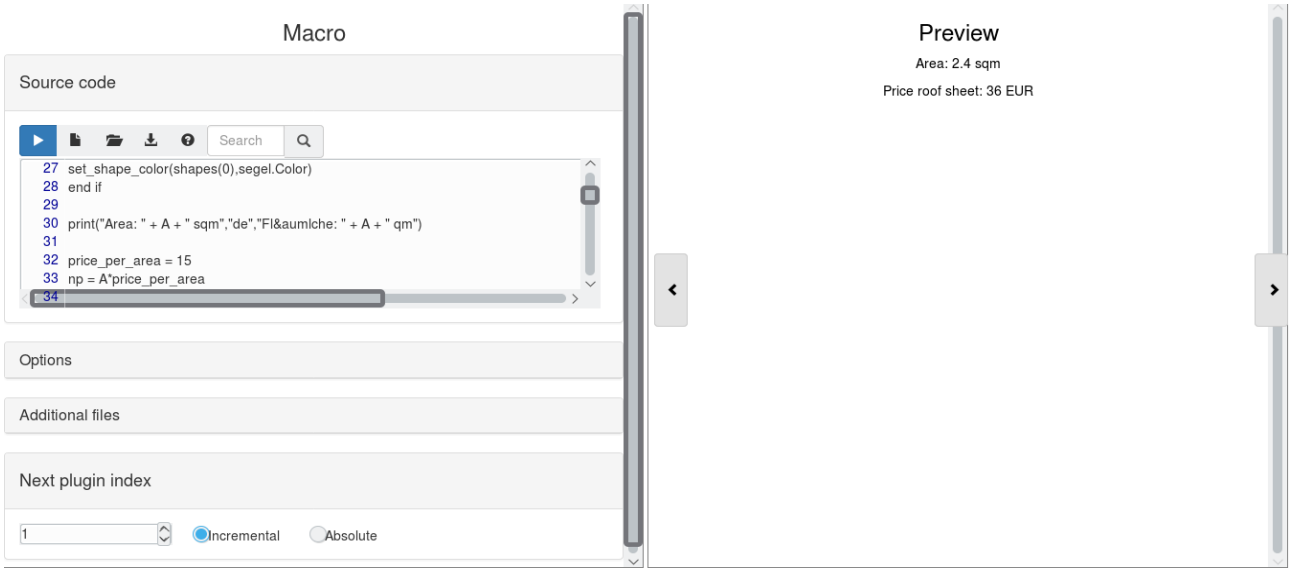

Figure 1: allpccloud macro editor

You can also load and save the source code from/to your local file system or create a new source file.

Additionally, you can opt in extended functionality options which are needed e.g. for CAD operations but will somewhat decrease the interpreter performance.

Additional files can be supplied and uploaded in the corresponding (expandable) panel to be available in the interpreter sandbox.

The program output is shown on the right hand side of the screen – this is also a preview for the program output in a shopper view frontend.

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Operators

CPBasic knows the operators '+', '-', '*', '/', '^', 'NOT', 'AND', 'OR' and 'XOR'. They implement basic arithmetic and logical operations as well as string concatenation. Most of these operators do what you expect from them, but of course it is not possible to apply a logical AND to real numbers.

Examples:

```
print(3 + 5)
print(5 - 2)
print(7 * 3)
print(21 / 2)
print(2 ^ 2)
print(2 ^ 0.5)
print(true and true)
print(true or false)
print(true xor true)
print(not true)
```

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Loops

Now we introduce the for ... next loop:

```
for i=0 to 5
  print("i = " + i)
next i
print("loop ends")
```

This loop increments the index \verb|i| as long as it does not exceed the parameter given in the to-directive (5 here). If this parameter is greater than the assignment expression, the index will be incremented by 1 each time the loop is repeated. If not, it will be decremented.

The step size can also be specified explicitly:

```
for i=0 to 5 step 0.5
  print("i = " + i)
next i
print("loop ends")
```

There are two other loop types known in CPBasic:

```
i = 0
while i < 5
  print("i = " + i)
  i = i+1
wend
```

```
print("loop ends")
```

This is the while .. wend-loop which is repeated as long as the conditions given in its header is fulfilled (here: i < 5). This condition is checked in the beginning of each loop run, in difference to the do .. while-loop:

```
i = 0
do
  print("i = " + i)
  i = i+1
while i < 5
print("loop ends")
```

Here, the condition is checked in the end of the loop body.

Conditionals

Of course CPBasic also knows conditional expressions:

```
i = 0
str = "hello"

if i = 0 then
  print("conditional 1 fulfilled")
end if

if str = "bye" then
```

```
  print("conditional 2 fulfilled")
end if
```

As you see, only the first condition yields a string output since the second condition is not fulfilled. By the way a CPBasic variable is polymorph, since data of any type can be assigned. This is true for numbers and strings but also for more complex data types which will be discussed later.

To handle situations where conditionals are not fulfilled more flexible, you can use an \verb|else|-block:

```
str = "hello"

if str = "bye" then
  print("conditional fulfilled")
else
  print("conditional not fulfilled, str is " + str)
end if
```

You can also catenate several conditions with the boolean operators and, or and xor. Here is an example:

```
a = 1
b = 5
c = 0

REM List is processed beginning with the leftmost item
if a < 1 and b > 5 xor c = 0 then
  print("if part")
```

```
else
  print("else part")
end if
```

In this example, the single items of the condition are processed from left to right. This means that the current item acts as the right operand of the preceding operator, while the left operand comes from the part of the conditional that has been evaluated so far. In the given example the first two items would be processed (a < 1 and b > 5). They are and'ed while the result is passed to the xor operator which also evaluates the c = 0 condition.

Functions

Of course it is possible to write functions in CPBasic. Here is the syntax of a very simple one:

```
func(1,2)

REM Here the function declaration starts.

sub func(a,b)
  print("a = " + a)
  print("b = " + b)
end sub
```

Functions can be defined anywhere in the program code. In CPBasic, no differences are made between functions and procedures. Each function defined with the syntax given above can return a value using the return keyword:

```
r = func(1,2)

print("r is " + r)

REM Here the function declaration starts.

sub func(a,b)
  print("a = " + a)
  print("b = " + b)
  return a + b
end sub
```

The return statement immediately ends function execution and can be used to terminate a function before its last line of code has been reached.

Variables

Variables do not need to be declared explicitly in CPBasic. An assignment, say,

```
i = 0
```

implicitly declares the variable i. By default, all variables are local, which means that they are only known in the function where they were declared.

Variables are case sensitive, which means that the variables i and I are not the same (CPBasic keywords and built-in functions are case insensitive).

Each variable has a type assigned. In the example this type is determined by the right hand side of the assignment. CPBasic knows the primitive types boolean, integer, double, string and object. Here are some more examples for implicit variable declarations:

```
REM A boolean
b = true
b = false

REM An integer
i = 0

REM A double
f = 0.9

REM A string
s = "a string"
```

As we will see later, it can be useful to declare a variable explicitly. This is done with the dim keyword:

```
dim v as integer
dim w as real
```

These variables are created with a defined type but with undefined content. Anyway there are situations when it is useful to declare explicitly, namely when they serve as output parameters for functions and methods.

If you need to work with global variables, for example in quite large interactive program scripts, the keyword global is used. Syntax:

```
gNumberOfItems = 0

global gNumberOfItems
```

The variable gNumberOfItems is global now which means that is is accessible from every part of the program now. Is content is removed from the local function's or method's stack frame and is stored into a special chunk. From now on every stack frame in the program gets a reference to the variable and is able to modify it like a local one.
Note that a global variable should be named uniquely to avoid name clashes.

Parameters

Parameters are input data for functions and methods. They can be accessed like local variables in the function's body. In fact they are treated like local variables in CPBasic's internals but they are references if possible: modifying the contents of function parameters means modifying the contents of the variables they are referencing. This process is transparent, the programmer does not recognize a difference between modifying a variable directly or modifying a reference to this variable.
The following example shows the possibilities of reference mechnisms more clearly:

```
a = 1
b = 2

print("Before: a = " + a + ", b = " + b)
```

```
swap(a,b)

print("After: a = " + a + ", b = " + b)

REM ********************************


sub swap(v1,v2)

  interm = v1
  v1 = v2
  v2 = interm

end sub
```

The result is:

```
Before: a = 1, b = 2
After: a = 2, b = 1
```

This means that the contents of the local variables of the main program a and b are modified from the function swap. This is only possible with references which are used when a function is called with one of the following constructions as parameter: local variables, array elements and member variables. It is not possible to create a reference to a constant. The reference mechanism allows CPBasic to make use of output parameters which are known in CORBA as well.

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Arrays

Arrays are defined with the dim keyword:

```
dim a(5) as integer

a(0) = 1
a(2) = 10
a(3) = 100
.
```

In contrast to most other BASIC dialects, array index counting starts with zero. Also the type of the array elements (the basis type) must be specified explicitly. Arrays can be extended dynamically using the Append() function.

Example:

```
REM An array without any content
dim a(0) as integer

Append(a,0)
Append(a,1)
Append(a,2)

for i=0 to ubound(a)

  print(a(i))
```

next i

To remove an element from an array one can use the function Remove(). Also, it is possible to use dim to resize an array afterwards.

Product Attributes

Product attributes can be accessed similar to variables using the product object and the attribute name. The product object is either a present product that must be resolved (e.g. using the function SearchProduct()) or created (CreateProduct()). If the product object is known an attribute can be accessed using the “.“ operator.

Example:

REM Assume that in the product tree the product with name “MyProductName“

REM is present and has a double attribute named “diameter“:

myprod = SearchProduct(“MyProductName“)

d = myprod.diameter

print(“d = “ + d)

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

User Reference

Built-in

print(string1 [,lang2, string2 [,lang3, string3]])

Prints string1 to the shopper view. Html tags are possible to e.g. set a certain color or font style. For language localisation it is possible to specify translated strings prefixed by the language string. For example, the print command

print(“Good morning“, “de“,“Guten Morgen“)

will print a German version if the Browser is set to German language. Otherwise the first string will be printed as default value.

abs(a)

Returns the absolute value of the integer or double value a.

acos(a)

Returns the radiant arc cosine of the value a.

append(array,v)

Appends the value v to the array array.

asin(a)

Returns the radiant arc sine of the value a.

atn(a)

Returns the radiant arc tangent of the value a.

cint(a)
clng(a)

Returns the rounded value of number a.

cos(a)

Returns the cosine of the randiant angle a.

csng(integer)

Explicitly cast integer to a double.

split(string, delimiter, column)

Analyzes a string by understanding it as a line, consisting of a finite number of columns, separated by one or more delimiting characters. For example, the string "a b c d" is a four column string supposed its delimiter is the space character " ". If its delimiter is, for example, "c", the string would include two columns. By the ,,column" paramter the column is picked, beginning with column no. 0.

exit()

Terminates the program explicitly

exp(double)

The exponential function e^x.

fix(double)

Casts a double number to the next smaller integer.

hex(integer)

Yields the hexadecimal stringified representation of a positive integer.

int(double)

See fix().

isnull(object)

Indicates if a variable is Null, meaning that this variable eigther has no valid content (e.g. was not initialized) or is not even typed. Frequently used when checking for valid return object values, e.g. SearchProduct().

language()

Returns an identifier string of the current language (example: “de“ for German)

len(string)

Returns the length of a string.

log(double)

Returns the natural logarithm ln(x).

oct(integer)

Yields a stringified representation of a positive integer in octal form.

pi()

Yields pi

rsplit(string, delimiter, column)

See split(), but with inverse column counting, i.e. column 0 is the last column.

remove(array, index)

Removes the entry from the array which is given by the index integer. If index is -1 the array is cleared.

sgn(double)

Returns -1 / 0 / +1 for arguments < 0 / = 0 / > 0.

sin(a)

Returns the sine of the randiant angle a.

sqr(double)

Returns the square root of the double argument value.

str(number)

Explicitly turns a numeric type into a string.

substr(string, begin, length)

Extracts a sub string from a string. This sub string starts at begin and is ought to have length characters.

tan(a)

Returns the tengent of the randiant angle a.

uid()

Returns a string with a unique identifier (ID) e.g. for a filename

val(s)

Parses the string s; extracts and returns a double value if possible

ubound(arr)

Returns the highest permitted index of the array arr.

chr(i)

Returns the ith ascii character

typename(v)

Returns the type name of variable v

Data base related

SessionID()

The current session ID of this data base

UserPermissionLevel()

The permission level of the users that owns the current data base session

RootSession()

Returns a flag if this is a root session, i.e. the root user owns this session

Push()

Performs a push, i.e. makes all changes in this session available to other sessions in this data base. Also stores the changes and increments product veriosn numbers if needed. Normally a push is not needed explicitly since when the wizard invokes a new plugin a push is done automatically.
Pull()

Reverse operation of Push(); loads every changes from the neutral data base list to this session. Normally this is not needed explicitly since a pull is done each time before a macro is started.

GetAvailableLabels()

Returns a string list of all available labels in this data base kernel

SaveLabel(label_name)

Saves a label, i.e. a snapshot of the current product version state under the given name. Normally this is not needed explicitly since when the wizard invokes a new plugin a label is saved automatically.

LoadLabel(label_name)

Loads a label, i.e. the snapshot of the current product version state with this name.

TheUserID()

Returns the current user‘s integer ID

GetProductsInSession()

Returns a list of all products in this session in their actual version

GetRootProducts()

Returns a list of all root products in this session in their actual version. A root product is a product that is not assembled elsewhere in another product, i.e. it is top level normally.

SearchProduct(product_name)

Returns the product of this name if it is present in the session. Else return null pointer.

CreateProduct(product_name)

Creates and returns a new product with name product_name. The product_name must not be in use elsewhere in this data base kernel since a product name must be unique. If the product cannot be created a null pointer is returned.

CopyProduct(copied_product, new_product_name)

Copies and returns a new product which is a clone of copied_product but has a new name. The product_name must not be in use elsewhere in this data base kernel since a product name must be unique. If the product cannot be copied a null pointer is returned.

DeleteProduct(product)

Deletes a product in this session. Note that physically a product is never really deleted and can be recovered if needed. Returns true on success.

Clear()

Deletes all products in this session. Again all deleted products can be recovered.

GetDeletedProductsInSession()

Returns a list of all deleted products of this session

SearchDeletedProduct(product_name)

Returns the product of this name if it was present in the session and has been deleted. Return null pointer if no such product could be found.

DiscardChanges()

Discard all changes that have been made in this session since the last push or the start of this session

ListTemplates()

Returns a list of all template product names

SaveTemplate(product)

Saves this product and all its sub products to the template storage. Here they are available to all data base kernels. Normally this is done for a generic product configuration (a template) which is loaded later in the configuration process and adjusted to the current session‘s needs.

LoadTemplate(product_name)

Loads a product identified by its unique product_name from the template storage. If the product is not available a null pointer is returned.

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Plugin sequence related

Barrier(exit_flag)

Prevents the shopper to switch to the next plugin, i.e. turns off the next-button. If exit_flag is true, the macro will also be ended. Normally it is not explicitly needed to call this function since the next-button is turned off by default.

Permit(exit_flag)

Enables the shopper to switch to the next plugin, i.e. turns on the next-button. If exit_flag is true, the macro will also be ended. It is explicitly needed to call this function since the next-button is turned off by default.

NextPluginIndex(next_plugin_index, incremental_flag)

Assigns which plugin will be invoked when pressing the next-button. The plugin is identified by its index next_plugin_index. If the incremental_flag is set to true this index is treated incrementally, i.e. added to the current index. The call NextPluginIndex(1, true) will cause a jump to the sequentially next plugin when next is pressed. This function overrides the settings which can be made directly in the web interface.

GoNext()

Immediately switch to the next plugin in the sequence without waiting for the user to click the next-button. This behaviour is disabled when the macro is edited, i.e. in admin mode.

CAD functions

Functions to create, modify and display 3D CAD shapes in BREP representation. You can handle STEP files for interaction with CAD desktop programs. These files can be up- and downloaded if needed.

To enable the functions described in this chapter you must check the “CAD, FEM” check box in the Options panel.

Base functionality

CADShapeSurfaceMesh(shape | shape array, deflection)

Creates surface meshes from one or more shapes with a defined defletion (e.g. 1 for 1 mm deflection from analytical surfaces. Meshes are needed for shape display. Returns either the single mesh or an array of meshes.

CADReadStep(dirname, filename)

Read a STEP file from a sandbox directory and returns the shape.

CADWriteStep(shape, dirname, filename)

Write a shape to a STEP file in a sandbox directory.

CADShapeSetProperty(shape, property name, property value)

Set shape properties or various kinds.

Boolean operations. Combine two or more shapes in a geometric operation.

CADComposeShapes(shape 1 [, shape 2 [,…]] | shape array)

Combine two or more shapes (either supplied by argument list or by array) in a single assembly. This assembly is returned by the function.

CADBoolean(boolean name, shape 1, shape 2)

Compine two shapes in a boolean operation and return the result. Possible operations (boolean name):

“add” or “+”: Add two shapes to form a new single shape

“cut” or “sub” or “-”: Subtract shape 2 from shape 1 and return the result

“common” or “&”: Return the common volume of shape 1 and shape 2 in the resulting shape

Primitives: Create a primitive shape

CADPrimitiveShape(primitive name, parameter 1 [, parameter 2 [,…]])

Make and return a primitive shape. Possible primitives:

“box” or “boxshell”: Creates and returns a box (or the shell of a box) made up by the three input parameters dx,dy,dz.

“cylinder” or “cylindershell”: Creates and returns a cylinder geometry. The cylinder axis initially corresponds to the global z axis. Parameters: radius and height.

“torus” or “torusshell”: Creates and returns a cylinder geometry. The first parameter is the torus axis radius, the second the minor radius.

“prism” or “extrusion”: Extrudes a profile (first parameter) along a vector (parameters 2,3,4).

Profiles

CADProfile(profile name, parameter 1 [, parameter 2 [,…]])

Creates and returns a flat profile face that e.g. can be extruded. Possible profile names:

“polygon”: A polygon from a set of lines, Parameters 1,2,3: Double arrays with respective x,y,z-coordinates of the polygon points. The last point must match the first point so the polygon is closed.

Transformations: Create transformations used to transform shapes, e.g. translation, rotation.

CADCreateTransformation(transformation name, parameter 1 [, parameter 2 [,…]])

Create and returns a transformation. Possible transformation names:

“new”: new empty transformation
“translation”: Translation. Parameters 1,2,3: Translation vector X,Y,Z.

“rotation”: Rotation. Parameters 1,2,3: rotation axis, Parameter 4: rotation angle (rad).

“scale”: Scale. Parameters 1,2,3: scaling center, parameter 4: factor of scaling.

CADChainTransformation(transformation 1, transformation 2)

Chains two transfromations and returns a transformation which is equivalent to first apply transformation 1 and then transformation 2.

CADTransformShape(shape, transformation)

Transforms a shape.

CADTransformedShape(shape, transformation)

Copies a shape and transform (and return) the copy.

Meshes

FVShowMeshes(mesh array, options string)

Displays an array of meshes. A canvas is created in the shopper view frontend where 3D drawing is done. To shown shapes see function CADShapeSurfaceMesh().

The options string is used to supply optional parameters for display, each separated by a “&” character. Not that CPBasic can’t handle empty strings so if you need no parameters please supply a whitespace. Possible options:

backgrColor

relWidth

aspectRatio

planeShow

planeColor

planePosY

Other

DownloadLink(filename, caption [, lang_string, cation_lang [, …]])

Creates a hyperlink for file download from the tmp directory of the web server. Useful if you have created a file there and want to offer it for a download. The caption can be supplied in various translations like in the print() function.

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Product member functions

Product member functions are always bound to a “this-“ product object variable, e.g. p, and invoked with this syntax: p.SomeMethod() (e.g. n = p.ProductName())

ProductName()

Returns the product name of this product

CreatedBy()

Returns the ID of the user who created this product

IsModified()

Returns a flag if this product has been modified since the last Pull(), Push() or start of this session

ModificationTime()

Returns the epoch time of the last modification (Seconds since 1.1.1970)

Version()

Returns the current product version

RestoreVersion(version)

Makes the version version the currrent one of this product

GetAvailableVersions()

Returns an array of versions which are available for this product

GetIsRemoved()

Returns a flag if this product has been removed (deleted)

SetIsRemoved(flag)

Set remove (delete) flag by hand. Same as DeleteProduct(p).

SetWritePermissionLevel(new_level)

Sets a new level of writing permissions. Only users with this or a higher permission level can write to this product.

SetReadPermissionLevel(new_level)

Sets a new level of reading permissions. Only users with this or a higher permission level can read this product.

GetWritePermissionLevel()

Returns the level of writing permissions

GetReadPermissionLevel()

Returns the level of reading permissions

IsReadable()

Flag if this product is currently readable

IsWritable()

Flag if this product is currently writable

GetSubProducts()

Returns an array of sub products which are currently assembled to this product

Assemble(other_product)

Assembles other_product to this product. The other product is a sub product then. A product can be assembled several times at verious placed so a complete product structure can be assembled.

Disassemble(sub_product_index)

allpccloud GmbH

Reichswaldallee 23a
40472 Düsseldorf

www.allpccloud.com

Disassembles a product, identified by its assembly index, from this product.

GetProductAttributes()

Returns an array of attribute names of this product

AddProductAttribute(attr_name)

Adds an attribute with a specified name to this product

DeleteProductAttribute(attr_name)

Deletes the attribute with the specified name from this product

File sandbox

All functions that read or write files either write to a default directory or the directory name can be yielded in a prepended string. The following directories are available:

“TmpDir”: The web server’s temporary directory. A file that is stored there can be downloaded by everyone e.g. via DownloadLink().

“AdditionalDir”: The directory where uploaded additional files are stored. This directory is read only.

“SessionDir”: A directory specific to a kernel session. Here files can be stored that are interchangable between plugins. The files in this directory can be downloaded by clients only.