Unofficial guide to AstroArt’s script commands

Prologue

Since using the CCD camera for my astronomical observation my reference software for the recovery and processing of images has always been AstroArt. Naturally over the years I have found to handle other software of this type, certainly some very good, and although my limited experience, they not allow me to say too much in benchmarking, I think for me the unbeatable immediacy with which the AstroArt user’s are able to use this software almost immediately. A very friendly user interface, combined with the scientific rigor of the algorithms used has made one of the leading software in the field of amateur astronomy. The advent of version 3.0 and higher, with command script has further expanded its potential, so that it is possible (if provided with the necessary hardware) to perform automated tracking and recovery, which greatly facilitate the conduct of the observing sessions.
Anyone who had a minimum of programming experience also knows how difficult and tedious write and especially keep updated the user manual. Therefore it happens that the manuals available is not always up to date with respect to the evolution of software, but also that some features are not always explained sufficiently extensive for the average users.

This guide seeks to illustrate some commands for AstroArt ‘s scripts that, in the user manual are poorly documented or even at all. As the title suggests this is an unofficial guide and although parts of it modeled on the original user manual does not purport to replace it, but rather as an informal integration with regard to the commands of the scripting of this magnificent software.

The commands are present in version 4.0 AstroArt, GUI 3.8 and later. Any inaccuracy and error that was to emerge on this guide are due solely to myself and I ask pardon in advance.

The Scripts

A script is a list of commands executed in sequence. Through the scripts you can automate several observational procedures such for example, automated search of asteroids and supernovae, taken photometric image and very high. This is possible because through AstroArt is possible to control not only the CCD camera, but also the filter wheel and the telescope (if this is pointing to power). The scripting language AstroArt, also known as "ABasic" is a kind of dialect of BASIC with a syntax very similar to that of many types of BASIC, (GWBasic ™, ™ QuickBasic, Visual Basic ™, VBScript ™, etc.).

Example of script (taken from manual AstroArt and testable with the CCD simulator in AstroArt)

```
Camera.Start(10)
Camera.Wait
Image.Save("C:\sample.fit")
```

The first command line starts up an exposure of 10 seconds, the second command line waiting for the end of the exposure before continuing with the script. Finally, the third line of command saves the image you just captured in drive c: \ calling it "sample.fit.

Another example of a script always taken from the manual AstroArt: the shooting of 50 images for the detection of supernovae.

```
for i = 1 to 50
ra = Telescope.List.Ra(i)
de = Telescope.List.Dec(i)
name$ = Telescope.List.Name$(i)
Telescope.Goto(ra,de)
Telescope.Wait
Camera.Start(60)
Camera.Wait
Image.Rename(name$ + ".fit")
Image.save(name$ + ".fit")
next i
```

In this simple script coordinates and name the of the galaxies are recovered directly from the list pre-loaded into the Telescope Window. For each galaxy provides the script to point the telescope, start the exposure and save the scanned image.

Variables and function

The ABasic supports two types of variables, numeric variables and string variables. The first contains a numerical number, while a variable string contains an alphanumeric string.

Numeric variable

They contain a number. This number is internally represented by a floating-point value with double precision (64 bits, 15 digits).

String variable

A variable string contains alphanumeric text. This text may be represented by one or more lines of text. The maximum size of a string variable is 64 MB.

Example:

```
a$ = "Hello"
b$ = a$ + "World"
```

The b$ string variable is now composed by the string "HelloWorld"

A single character of a string can be read using square brackets, thus Using the previous example a$ [1] returns as a result of "H" and a$[2] returns “e” and so on.
If the index number shown in brackets exceeds the length of the string this restarts from the beginning, therefore, $ [6] still returns "H".

A single row of a multi-line string can be read using curly brackets.

Example:

If the variable a$ a contains the following text distributed on three lines

```
“This text
is distributed
on three lines”
```

In this case a${2} returns the string distributed on the second line “is distributed".
The function **count(a$)** returns on the number of lines comprised in a multi-line string.

Reserved word.

These words are part of the ABasic language, so normally they must not be used as an argument in string variables. They are:

IF	MOD	WHILE	CLS
THEN	REM	ENDWHILE	
ELSE	FOR	GOTO	
ENDIF	NEXT	GOSUB	
OR	STEP	PRINT	
AND	BREAK	INPUT	
NOT	CONTINUE	END	

Let us now describe in more detail.

Cyclical function: FOR, NEXT, STEP, BREAK, CONTINUE, WHILE, ENDWHILE.

The ABasic supports two types of loop instruction**:** **For**-**Next** function and **While**-**Endwhile** function. The complete syntax for the FOR-NEXT loop function is as follows:

FOR <variable> = <expression> TO <expression> [STEP <numeric constant>]
...
...
NEXT <variable>

For-**Next** instruction trigger a loop for a number of times of the instructions between the two commands: **For** (cycle start) and **Next** (end cycle). A simple example is print to the screen the numbers from 1 to 10, "a" is the control variable.

```
for a = 1 to 10
print a
next a
```

The break command exits from a loop, in this example the cycle is interrupted when the value of control variable 'a' becomes larger than 5.

```
for a = 1 to 10
print a
if a>5 then break
next a
```

The **CONTINUE** command is used inside a **For-Next** loop and acts in a manner analogous to **NEXT** command, so automatically start a new iteration. For example:

```
for a = 1 to 10
print a
if a > 5 then continue
print "Test"
next a
```

Finally, the **STEP** command is used at the beginning of a **For**-**Next** loop to determine the progression of the control variable. For example:

```
for a = 1 to 10 step 2
print a
next a
```

In this way the control variable 'a' skip all even numbers from 1 to 10. **STEP** function may also to have negative values, for example:

```
for a = 10 to 1 step -1
print a
next a
```

Instead, the **WHILE-ENDWHILE** command evaluates the condition at the beginning of the cycle. If the condition is false then the cycle is interrupted and execution continues after the **ENDWHILE** command.

For example:

```
a = 1
while a <= 10
print a
a = a+1
endwhile
print “cycle finished”
```

because the **WHILE** command evaluates the condition at the beginning of the cycle the instructions inside the loop could also never be executed.
The **BREAK** and **CONTINUE** commands can be used in a loop **WHILE-ENDWHILE** in a completely similar to what has been seen for the FOR-NEXT loop.

Conditional Function: IF, THEN, ELSE, ENDIF, OR, AND, NOT

The IF-THEN-ENDIF commands evaluates a logical expression and determines the program flow according to the result of that expression.

Some examples of logical expressions:

```
a > 5 and b$ = "astro"
a >= 3 or not (b = 5)
```

The logical and mathematical operators used in logical expressions have their own scale of priorities when they have to be written in the list of instructions, the following is the scale of precedence of these operators going from highest priority to lowest priority.

Operator priority:

Highest priority ( ), < , > , <= , >= , <> , = , NOT, AND, OR Lowest priority

Extended syntax of the command **IF-THEN-ENDIF**

IF <logical expression> **THEN**
...
...
[**ELSE**]
...
...
ENDIF

Example of command **IF-THEN-ENDIF**

```
for i = 10 to -10 step -1
if i>0 then
print "positive value"
endif
if i=0 then print "zero"
if i<0 then
print "negative value"
endif
```

Compact sintax of the command **IF-THEN**

IF <logical expression> **THEN** <instruction> [**ELSE**]< instruction >

Example of Compact sintax of the command **IF-THEN**

```
for a = 1 to 10
if a <= 5 then print "-" else print "+"
next a
```

Other function:

Function	Details	Example
REM	Any text preceded by **REM** command is ignored during program execution. This feature allows the inclusion of records in the list commands. The inclusion of a superscript "’" works in the same way than **REM**.	**REM notes** **‘notes** **Output:**
GOTO n	Skip the program execution to line **n** ignoring all that lies between **GOTO n** command and line **n**.	**for i = 1 to 10** **if i = 5 then goto 10** **print i** **next i** **10 print " I jumped on line 10"** **Output:** **1** **2** **3** **4** **I jumped on line 10**
GOSUB n **RETURN**	Skip the program execution to line **n** ignoring all that lies between **GOTO n** command and line **n**, but once met with the **RETURN** command Back to the line immediately below the command **GOSUB n.**	**for i = 1 to 10** **if i = 5 then gosub 10** **print i** **next i** **END** **10 print " I jumped on line 10"** **print " but now back where I started "** **RETURN** **Output:** **1** **2** **3** **4** **I jumped on line 10** **but now back where I started** **5** **6** **7** **8** **9** **10**

PRINT “s”	Print to the screen a text “**s**”, numeric variable **n** or alphanumeric string **s$**. Text and variables may also be linked on the same command line.	a=4 b$="Version" c$="execute with" print "script AstroArt " print c$+" Astroart "+b$;a Output: script AstroArt execute whit Astroart Versione 4
INPUT	Allows input from the user's numerical variables or string variables.	input "insert a number: ",n input "insert a string: ",s$ print n print s$ Output: The program shows two successive windows for insert the data. Written data that appear in the Output window.
END	ends execution of a script.	print "program expired at row 3" print "row 1" print "row 2" print "row 3" END print "row 4" Output: row 1 row 2 row 3
CLS	Clear the output windows.	for i = 1 to 10 print "abcdefghilmo" next i message ("press 'OK' for clearing the output window") CLS Output:

Numeric functions.

Function	Details	Example
pi()	returns the value of pi greek	**Print pi()** **Output:** **3.141592654**
sin(n)	Calculate the sine of the angle **n** in radians. If the angle **n** is expressed in degrees instead of radians, use the following procedures: **sin(n*pi()/180)** otherwise **sin(degtorad(n))**	**Print sin(90)** **Output:** **0.8939966636**
cos(n)	Calculate the cosine of the angle **n** in radians. If the angle **n** is expressed in degrees instead of radians, use the following procedures: **cos(n*pi()/180)** otherwise **cos(degtorad(n))**	**Print Cos(90)** **Output:** **-0.4480736161**
tan(n)	Calculate the tangent of the angle **n** in radians. If the angle **n** is expressed in degrees instead of radians, use the following procedures: **tan(n*pi()/180)** otherwise **tan(degtorad(n))**	**Print tan(50)** **Output:** **-0.271900612**
exp(n)	Calculate the value of Napier's number raised to the **n**, or e^n	**Print exp(10)** **Output:** **22026.46579**
ln(n)	Calculate the value of the logarithm to the base e (natural logarithm) of **n**	**Print ln(10)** **Output:** **22026.46579**
log10(n)	Calculate the value of the logarithm to the base 10 of **n**	**Print log10(100)** **Output:** **2**
log2(n)	Calculate the value of the logarithm to the base 2 of **n**	**Print log2(50)** **Output:** **5.64385619**
sqr(n)	Calculate the square root of number **n**	**Print sqr(16)** **Output:** **4**
abs(n)	Calculate the absolute value of number **n**	**Print abs(15)** **Print abs(-15)** **Output:** **15** **15**

rnd(n)	Return a random number between **0** and **n**	**For i = 1 to 5** **Print rnd(10)** **Next i** **Output:** **0.9364372841** **6.289201556** **2.800253921** **8.77184656** **1.612342733**
sgn(n)	Return the sign of a number **n** according to the scheme: **sgn(n) = -1 if n < 0,** **sgn(n) = 0 if n = 0,** **sgn(n) = 1 if n > 0.**	**Print sgn(-12.345)** **Print sgn(0)** **Print sgn(12.345)** **Output:** **-1** **0** **1**
fix(n)	Return the integer part of a number **n**	**Print fix(8.771845)** **Output:** **8**
int(n)	Return the integer part of a number **n**	**Print int(8.771845)** **Output:** **8**
round(n[,n1])	Rounds a number **n** to **n1** th decimal place. NOTE: if **n1** are omitted n are rounded for zero decimal place.	**Print round(8.771845,3)** **Print round(8.771845)** **Output:** **8.772** **9**
frac(n)	Return the fractional part of **n** number.	**Print frac(10.45678)** **Output:** **0.45678**
asin(n)	Calculates the arcsine in radians of a number **n**. Function valid in the range (1, -1) For values outside this range is returned the null value "NAN". For convert from radians to grade: **asin(n)*180/pi()** otherwise **radtodeg(asin(n))**	**Print asin(1),”radians”** **Print asin(n)*180/pi(),”Grade”** **Output:** **1.570796327 radians** **90 Grade**
acos(n)	Calculates the arccosine in radians of a number **n**. Function valid in the range (1, -1) For values outside this range is returned the null value "NAN". For convert from radians to grade: **acos(n)*180/pi()** otherwise **radtodeg(acos(n))**	**Print acos(1),"radians"** **Print acos(n)*180/pi(),"Grade"** **Output:** **0 radians** **0 Grade**
atan(n)	Calculates the arctangent in radians of a number **n.** For convert from radians to grade: **atan(n)*180/pi()** otherwise **radtodeg(atan(n))**	**Print atan(1),"radians"** **Print atan(1)*180/pi(),"Grade"** **Output:** **0.7853981634 radians** **45 Grade**

atan2(nx,ny)	Calculates the arctangent2 in radians of a point with coordinates (**nx, ny**). For convert from radians to grade: **atan2(nx,ny)*180/pi()** otherwise **radtodeg(atan2(nx,ny))**	**print atan2(40,50)** **Output:** **0.6747409422**
sinh(n)	Calculates the hyperbolic sine of a number **n** in radians. For convert from radians to grade: **sinh(n*pi()/180)** otherwise **sinh(degtorad(n))**	**print sinh(10)** **Output:** **11013.23287**
cosh(n)	Calculates the hyperbolic cosine of a number **n** in radians. For convert from radians to grade: **cosh(n*pi()/180)** otherwise **cosh(degtorad(n))**	**print cosh(10)** **Output:** **11013.23292**
tanh(n)	Calculates the hyperbolic tangent of a number **n** in radians. For convert from radians to grade: **tanh(n*pi()/180)** otherwise **tanh(degtorad(n))**	**print tanh(10)** **Output:** **0.9999999959**
asinh(n)	Calculates the hyperbolic arcsine in radians of a number **n**. For convert from radians to grade: **asinh(n)*180/pi()** otherwise **radtodeg(asinh(n))**	**print asinh(100)** **Output:** **5.298342366**
acosh(n)	Calculates the hyperbolic arccosine in radians of a number **n**. For convert from radians to grade: **acosh(n)*180/pi()** otherwise **radtodeg(acosh(n))**	**print acosh(10)** **Output:** **2.993222846**
atanh(n)	Calculates the hyperbolic arctangent in radians of a number **n**. Function valid in the range (1, -1) For values outside this range is returned the infinite value "INF". For convert from radians to grade: **atanh(n)*180/pi()** otherwise **radtodeg(atanh(n))**	**print atanh(0.5)** **Output:** **0.5493061443**
degtorad(n)	Convert a number **n** from grade to radians.	**n=57.29577951** **print "Grade value: ";n** **print "Radian equivalent: "; degtorad(n)** **Output:** **Grade value: 57.29577951** **Radian equivalent:0.9999999999**
radtodeg(n)	Convert a number **n** from radians to grade. .	**n=1** **print "Radians value: ";n** **print "Grade equivalent: "; radtodeg(n)** **Output:** **Radian value: 1** **Grade equivalent: 57.29577951**
modulo(n1,n2)	Calculate the expression **radq**((**n1**^2)+(**n2**^2)).	**print modulo(1,5)** **Output:** **5.099019514**

len(s)	Returns the number of characters in a string **s** (also counted the spaces between words).	**a=len("viva AstroArt!")** **print "the string contains ";a;" characters "** **Output:** **the string contains 14 characters**
val(s)	Convert a string contains numeric characters in the corresponding number.	**a$="1"** **print val(a$)+2** **Output:** **3**
asc(s)	Return the ANSI code of the leftmost character of the string.	**print asc("AstroArt")** **Output:** **65**
pause(n)	Pauses program execution for **n** number of seconds.	**pause(30)** **Output:**
n1 mod n2	Return the rest of division **n1/n2**	**print 14 mod 4** **Output:** **2**
count(s)	Return the number of row in a multi line string **s**.	**a$="bye"+crlf$()+"bye"** **print a$** **print crlf$()** **print "the number of rows in the variable string is: ";count(a$)** **Output:** **bye** **bye** **the number of rows in the variable string is: 2**
counter(n)	This function is mentioned in the AstroArt manual but it does not exist in ABasic	

String function

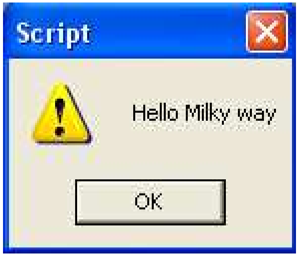
Function	Details	Example
ucase$(s)	Converts all characters in a string **s** in uppercase.	**print ucase$(“astroart”)** **Output:** **ASTROART**
lcase$(s)	Converts all characters in a string **s** in lowercase.	**print lcase$(“ASTROART”)** **Output:** **astroart**
ltrim$(s)	It removes the empty spaces to the left of a string.	**print "without ltrim$: "+" astroart"** **print "with ltrim$: "+ltrim$(" astroart")** **Output:** **without ltrim$: astroart** **with ltrim$: astroart**
rtrim$(s)	It removes the empty spaces to the right of a string.	**print "astroart "+" without rtrim$:"** **print rtrim$("astroart ")+" with rtrim$:"** **Output:** **astroart without rtrim$:** **astroart with rtrim$:**
chr$(n)	Returns the character corresponding to ASCII code number **n**.	**print chr$(64)** **Output:** @
str$(n)	Convert a number **n** from numeric value to characters string.	**a=234** **a$=str$(a)** **print "a = ";a;" is a number"** **print "a$ = "+a$+" is a string"** **Output:** **a = 234 is a number** **a$ = 234 is a string**
mid$(s,n1,n2)	Return a substring of **s** string that is cut on the left starting from characters **n1** and **n2** is number characters long.	**print mid$("abcdefghilmnopqrstuvz",2,5)** **Output:** **bcdef**
hex$(n)	Converts a decimal number **n** in the string that represents the value in hexadecimal format.	**print hex$(1000)** **Output:** **3E8**
left$(s,n)	Return a substring of string **s** that is cut on the left from the first character and **n** is number characters long.	**print left$("AstroArt",5)** **Output:** **Astro**
right$(s,n)	Return a substring of string **s** that is cut on the right from the first character and **n** is number characters long.	**print right$("AstroArt",3)** **Output:** **Art**

ltab$(s,n)	Shift a string to the right for **n**-len(**s**) characters than a string **s**. This command working only if **n**-len(**s**)>0	**a$="Astro"** **print "the work'";a$;"' it’s long ";len(a$);" characters"** **for n=0 to 10** **print ltab$(a$,n)+str$(n)** **next n** **Output:** **the work ' Astro ' it’s long 5 characters** **Astro0** **Astro1** **Astro2** **Astro3** **Astro4** **Astro5** **Astro 6** **Astro 7** **Astro 8** **Astro 9** **Astro 10**
rtab$(s,n)	Shift the string **s** to the right of **n**-len(**s**) characters of the start line. This command working only if **n**-len(**s**)>0	**a$="Astro"** **print "the work'";a$;"' it’s long ";len(a$);" characters"** **for n=0 to 10** **print rtab$(a$,n)+str$(n)** **next n** **Output:** **the work ' Astro ' it’s long 5 characters** **Astro0** **Astro1** **Astro2** **Astro3** **Astro4** **Astro5** **Astro6** **Astro7** **Astro8** **Astro9** **Astro10**
format$(n,s)	Replaces the character 0 (zero) in the string **s** with digits of the numeric value **n**. The replacement takes place from right to left. If the number of 0's is present in less than the number of digits **n** of the remaining digits will be shown to the left of the last 0. If the number of 0 in **s** is greater than the number of digits of **n** 0 to the left of the last digit of **n** will appear as zero.	**print date$(); " today’s date"** **aaaammgg$=left$(date$(),4)+mid$(date$(),6,2)+right$(date$(),2)** **print format$(val(aaaammgg$),"Or: year 0000 month 00 day 00")** **Output:** **2010 10 06 today’s date** **Or: year 2010 month 10 day 06**
time$()	Return a string with actual value of time in (hh mm ss) format.	**print time$()** **Output:** **09 06 36**
date$()	Return a string with actual value of date in (aaaa mm gg) format.	**print date$()** **Output:** **2010 10 06**

crlf$()	Function equivalent to the carriage return, insert a blank line in the output window.	**print "astroart "+"astronomical "+"software"** **print crlf$()** **print "astroart "+crlf$()+"astronomical "+crlf$()+"software"** **Output:** **astroart astronomical software** **astroart** **astronomical** **software**
opentext$(s)	Open a text file named **s**. Please note that in the **s** string must also appear the file extension and the path. If path is omitted the file will be searched only in the current directory.	**file$=opentext$("C:\WINDOWS\system32\rsvpcnts.h")** **print file$** **Output:** **/*++** **Copyright (c) 1996 Microsoft Corporation** **#define RSVPOBJ 0** **#define RSVP_INTERFACES 2** **#define RSVP_NET_SOCKETS 4** **#define RSVP_TIMERS 6** **#define API_SESSIONS 8** **#define API_CLIENTS 10** **Etc etc etc…….**
savetext$(s1,s2)	Write the string **s1** in a text file named **s2**. Please note that in the **s2** string must also appear the file extension and the path. If path is omitted the file will be searched only in the current directory. Warning: This command overwrites any other file with the same name in the same folder.	**print savetext$("AstroArt, the best software for astronomical imaging","c:\AA.txt")** **print "in the directory c:\** should have appeared **" +crlf$()+"a text file named 'AA.txt' " +crlf$()+"contenent inside the sentence:" +crlf$()+"'AstroArt, the best software for" +crlf$()+"astronomical imaging'"** **Output:** **in the directory c:\ should have appeared** **a text file named 'AA.txt'** **contenent inside the sentence:** **'AstroArt, the best software for astronomical imaging'**
copytext$(s)	Copy the string **s** in the clipboard.	**print copytext$("AstroArt")** **a$=pastetext$()** **print a$** **Output:** **AstroArt**
pastetext$()	Paste the contents of the clipboard on the output windows or in a variable.	**print copytext$("AstroArt")** **print pastetext$()** **a$=pastetext$()** **print a$** **Output:** **AstroArt** **AstroArt**

finddir$(s1,s2)	Search a directory named **s2** in a path **s1**.	input "Path directory ",path$ input "directory name to find",dir$ a$=finddir$(path$,dir$) print "Search: "+a$ al$=lcase$(a$) dirl$=lcase$(dir$) if al$=dirl$ then print "directory found" else print "directory NOT found" endif Output directory found (if exist) oppure: directory NON found (if not exist)

findfile$(s1,s2)	Search a file named **s2** in a path **s1**.	pathltp$="c:\"
		b$="AA.txt"
		c$= pathltp$+b$
		print savetext$("AstroArt, the best software for astronomical imaging",c$)
		input "file name? ",obj_name$
		findf$=findfile$(pathltp$,obj_name$+".txt")
		if findf$=(obj_name$+".txt") then
		 print "File FOUND!"
		endif
		if findf$<>(obj_name$+".txt") then
		print "File NOT FOUND!"
		endif
		
		Output:
		File FOUND! (if you typed in the input the uppercase text 'AA')
		File NOT FOUND! (if you not typed in the input the uppercase text 'AA')

message(s)	Show a message on the screen contenent the **s** string.	Message(“Hello Milky way”) Output:  Hello Milky way
ra$(n)	Convert the RA (Right Ascension) value expressed as a decimal number from **n** to a string indicating the value of right ascension expressed in hh mm ss.s	alpha=05.345678 print "Right Ascension: "+ra$(alpha) Output: Right Ascension: 05 20 44.4

dec$(n)	Convert the DEC (Declination) value expressed as a decimal number from **n** to a string indicating the value of Declination expressed in +/-gg pp ss.s	**delta=-12.345678** **print "Declination: "+dec$(delta)** **Output:** **Declination: -12 20 44**
createdir(s)	Create a directory with the name and path specified by the string **s**. If you do not specify a drive and / or the directory’s path will be created in the current directory.	**createdir("c:\images")** **Output:**

Function for CCD , Filter wheel and Telescope.

Funzione	Dettagli	Esempi
Camera.Start(n[,n1])	Take an exposure of **n** seconds. Set **n1** to zero to take a dark frame.	**Camera.Start(60,0)**
Camera.Wait	Waits until the end of the exposure.	
Camera.Exposing	Returns “1” if a exposure is in progress, otherwise “0”.	
Camera.Binning(n)	Sets the binning mode.n is a index to the binning list in the “Settings” page of the CCD panel.	**Camera.Binning(2)**

Camera.SelectDarkFrame
 	
Camera.EnableDarkFrame(n)
	Selects the current image as dark frame and automatically enables the correction for the following images.
	
	Enables or disables the dark frame correction.
	n = 1 correction enable
	n = 0 correction disable
		Camera.SelectDarkFrame()
		
		Camera.EnableDarkFrame(0)

Camera.Stop	Stops the current exposures.	
Guider.Stop	Stops the current guiding session.	
Guider.Close	Close the guiding window.	
Guider.Select(n)	Selects which CCD should be used for autoguide: 1 = main ccd, 2 = guide ccd, 3= secondary camera.	**Guider.Select(2)**

Guider.MoveReference([dx, dy])	Changes the x and y coordinates of the reference star, to perform the “dithered guide”. If dx and dy are not specified then the shift will be pseudo-random.	**Guider.MoveReference()**
		GuiderMoveReference(-0.3, 0.7)

Camera.Connect([driver]) **Camera.Disconnect**	Connects the CCD driver from Astroart. Disconnects the CCD driver from Astroart.	**Camera.Connect(“Simulator”)**

Camera.StartAutoguide([x, y])	Starts and autoguide session. If x and y parameters (the coordinates of the guide star) are not given then this command takes a sample image and selects automatically the best star.	**Camera.StartAutoguide()**
		x = Image.GetPointX()
		y = Image.GetPointY()
		Camera.StartAutoguide(x,y)

Camera.StopAutoguide()	Stop autoguide session.	

Camera.Autofocus([x,y])	Starts an autofocus session (requires the Ascom autofocus plugin). If x and y parameters (the coordinates of the focus star) are not given then this command selects automatically the best star from the current image.	**Camera.Autofocus()**
		x = Image.GetPointX()
		y = Image.GetPointY()
		Camera.Autofocus(x,y)

Focuser.GotoRelative(n)	Moves the focuser up or down by a specified amount **n**.	**Focuser.GotoRelative(-50)**
Focuser.GotoAbsolute(n)	Move the focuser to a given coordinate.	**Focuser.GotoAbsolute(1000)**
Telescope.Goto(ra,dec)	Moves the telescope to the equatorial coordinates ra (0..24),dec. (-90..+90)	**Telescope.Goto(23.45, 44.12)**
Telescope.Wait	Waits until the telescope has completed a Goto.	
Telescope.Stop	Stops the telescope.	
Telescope.Ra **Telescope.Dec**	Returns the current position of the telescope.	**x = Telescope.Ra** **y = Telescope.Dec**
Telescope.Pulse(dir$ [,time])	Moves the telescope for **time** seconds towards the **dir$** direction (”N”,”S”, ”E”,”W”). If **time** is negative then the direction is inverted. If time is omitted, it moves until the **Telescope. Stop** command.	**Telescope.Pulse(“N”, 0.5)**

Telescope.Speed(n)	Sets the speed for Pulse motion. 1=guide 2=center 3=find 4=slew	**Telescope.Speed(4)**
Telescope.List.Open(file$)	Opens a text file which contains objects and coordinates. See chapter 6.1.	**Telescope.List.Open(“c:\data\galaxies.txt”)**
Telescope.List.Count	Returns how many objects are listed in the Telescope Window.	**n = Telescope.List.Count**
Telescope.List.Clear	Clears the list of object in the Telescope Window.	

Telescope.List.Ra(n)
Telescope.List.Dec(n)
	Return the coordinates of the **n** th object of the list.	**x = Telescope.List.Ra(25)**

Telescope.List.Name$(n)	Returns the name of the **n** th object of the list.	**a$ =Telescope.List.Name$(42)**
Telescope.Send(s)	Sends a string **s** to the telescope via the serial port.	**Telescope.Send(“#Hc#”)**
Wheel.Filters	Returns the number of filters of the filter wheel.	**n = Wheel.Filters**

Wheel.Goto(n)
Wheel.Goto(s)
	Moves the filter wheel to the given filter express.
	expressed by its filters number **n** or its filters name **s**.
		Wheel.Goto(4)
		Wheel.Goto(“R”)

Image.Save(filename$)	Saves the current image with path and namefile specified by **filename$**. If the path are omitted the image are saved in the active directory.	**Image.Save(“C:\images\saturn.fit”)**
Image.Rename(name$)	Renames the current image.	**Image.Rename(“jupiter.fit”)**
Image.Open(filename$)	Opens an image from disk with path and namefile specified by **filename$**. If the path are omitted the image are saved in the active directory.	**Image.Open(“C:\moon.tif”)**
Image.GetKey$(key$) **Image.GetKey(key$)**	Reads string values from the FITS header named **key$**. Reads numeric values from the FITS header named **key$**.	**a =Image.GetKey(“NAXIS”)** **a=Image.GetKey("EXPOSURE")**

Image.SetKey(key$,value)	Write in the header a parameter named **key$** with the value **value**.	**Image.SetKey(“COMMENT”,” Bad seeing”)**
		Image.SetKey(“JD”,34234)

Image.FlipH	Flips the current image horizontally.This feature requires AstroArt 4.0 + Service Pack 1.	

Image.FlipV	Flips the current image vertically. This feature requires AstroArt 4.0 + Service Pack 1.	
Image.Resize(x,y)	Resize an image to the size horizontal **x** and vertical **y**. This feature requires AstroArt 4.0 + Service Pack 1.	**Image.Resize(320,240)**
Image.BlinkAlign	Aligns the current image with the next one inside the Astroart Desktop and blinks them. This feature requires the Service Pack1.	**Image.BlinkAlign**
Image.Close	Closes the current image.	

Image.GetPointX()
Image.GetPointY()
	Return the coordinate of the selected point (or star, or rectange) on the current image.	**x = Image.GetPointX()**

Image.DSS(ra,dec,name$)	Creates a new image from the ‘Digital Sky Survey’ atlas. The DSS images. This image will be centered on the coordinates **ra** and **dec** and will be named **name$**. Needs the DSS plugin.	**Image.DSS(12.034,45.213,”asteroid.fit”)**
Output.Save(filename$)	Saves the output panel to disk with path and namefile specified by **filename$**. If the path are omitted the image are saved in the active directory.	**Output.Save(“C:\Log.txt”)**
Output.Copy	Copies the output panel to the Clipboard.	
System.Execute(filename$)	Executes a external program. with path and namefile specified by **filename$**. If the path are omitted the program is searched in the active directory.	**System.Execute(“C:\Windows\Notepad.exe myfile.txt”)**
System.Broadcast(message$, **wparam,lparam)**	Sends a Windows Message to all windows. This can be used to control other programs. The function is equivalent to: **h = RegisterWindowMessage(message$)** **SendNotifyMessage(HWND_BROADCAST,h,wparam,lparam)**.	

Function not documented.

Function	Details	Example
system.shutdown	Close AstroArt and power off the computer. Irreversible function, use carefully.	