PCA ARGUS PROGRAMMING

REFERENCE HANDBOOK

2nd edition 2020

XLI - THE PCA SCRIPT LANGUAGE

History:

The first XLI interpreter came out in1993 as the "XLI-Toolkit" which was a book and a couple of scripts and utilities for the PCA for DOS.

With the windows versions of PCA - Argus a couple of functions and codes were added. This manual has until now been updated with small bits and pieces only.

The present version (2020) has all the new codes and functions included, some few are deprecated, but still included to only give a no-operation, so that older modules are not stopping with a syntax error.

Introduction

The XLI interpreter is built into PCA / ARGUS. The interpreter can handle astrological interpretations as well as other text or graphics outputs and special calculations. It also gives access to a couple of Argus' settings. A number of modules and interpretations are commercially available while other are free. Some of them can be downloaded from the Electric Ephemeris website www.electric-ephemeris.com.

You may also program XLI modules yourself. You can do some simple stuff without being a programmer, but you can also write very sophisticated modules or interpretations with a lot of graphic details, specialized calculations, tables, ephemerides, etc.

You can add as many modules as you like and they can be branched and chained into any structure, and you may add menus to navigate around.

This documentation will give you the information necessary to write your own modules and provide a complete XLI function reference and examples.

You may look into any modules provided with your PCA program, or you may download extra modules and look into them to figure out how they are programmed.

Some modules may be encrypted which will restrict any editing. Argus has a module editor inbuilt (FILE > EDIT MODULE) which will let you change text parts of a module even if it is encrypted.

Using and writing interpretations (entry level)

The XLI interpreter offers an extensive and utmost flexible mechanism for writing computer interpretations. You may define even very complex rules and texts. Even though this manual provides the information necessary, it is not a beginners job to write a complex--rule interpretation.

For simple interpretations, Argus is distributed with ready-made interpretation skeletons and a built-in fool-proof texteditor. You may use these to write your own full blown interpretation, or you may use it as a note-organizer for basic astrological combination rules.

You may have objections about computer interpretations. We would certainly agree that

the computer cannot "understand" the person behind the chart, nor have any psychological insight. But computers are very good at organising facts and freeing you of the systematical (and boring) work. So you may for example use the interpretation skeletons to insert your experiences of certain combinations when you meet them, and let the program scan all new charts to remind you, when the same combination reappears. As a beginner, you may put your notes from books and lectures into the skeletons in the same way.

If you wish to do more complex things, you must refer to the second part of this manual, describing the XLI programming language functions.

INSTALLATION OF NEW MODULES IN ARGUS:

If a module is not prepared for automatic installation, you must use the setup menu in ARGUS to manually create a menu item and connect it to any XLI file on the harddisk.

If the module is prepared for automatic installation, it is usually a packed set of files including a .def file giving a list of files to copy, a name, optionally a name of a subdirectory where to put the files, also optionally an .ico file or a name of one of the standard .ico files Argus provides.

DEF FILE:

Transit-Interpretation

```
TYDT
200
TT1.TXT
TT2.TXT
TT3.TXT
TT4.TXT
TT5.TXT
TT6.TXT
TT7.TXT
TT8.TXT
TT9.TXT
TT10.TXT
```

The first line is the text which will be put on the menu item, which should be the name of the interpretation.

The second line is the name of the subdirectory under ARGUS, in which the interpretation files should be placed.

The third line is a number, telling how many kilobytes the installaion will need, so that the installation program can give a warning if the harddisk is too filled up to hold the files.

The fourth line onwards are the names of the text files. The first of these must be the entry file, i.e. the first in the call chain.

```
Transit Interpretation
```

```
TYDT
200
TT1.TXT
TT2.TXT
TT3.TXT
TT4.TXT
TT5.TXT
&TT6.TXT
TT7.TXT
TT8.TXT
TT9.TXT
TT10.TXT
```

XLI Programming

THE CONCEPT OF XLI MODULES

The PCA astrology program has a built-in option to interpret external add-on files. These files may be interpretation texts, choice menus, position printouts, aspect tables, educational programs, and even completely separate programs as for example text editors.

These add-on modules which we in this book will refer to as XLI-modules may be written by anyone, just using the basic Notepad editor included in Window. How to do this, is what this chapter is about.

Writing your own modules or interpretations will need something between no programming skills at all up to expert level depending on the task. So you can choose your own level and proceed to higher levels if you like.

ABOUT THIS CHAPTER

You will be guided through the XLI mysteries in stages revealing the different features of the language starting at the very basic level and leading towards more sophisticated use.

At each level you will find reference to ready made tested examples that you can try out for yourself.

XLI FILES

The XLI-modules are text files (ASCII-Files). Files are named groups of data, that can be stored on your disc and printed to your printer. A text file as opposed to a program file or binary file can be read on the screen or printed to paper and still make some kind of sense to the reader.

The file system needs that all files have names of letters or numerals. Underscore and stroke is also allowed. Normally a file name has two parts separated by a full-stop. The 8 part is the name, and the last (normally consisting of three letters) after the fullstop are called the extension. The extension is normally used to tell something about the kind and purpose of the file. The XLI-files all have the extension XLI or TXT. The filename and extension are separated by a fullstop e.g. TRANSIT.XLI. The .TXT extension is intended for interpretation texts, while the XLI extension should be used for functional modules doing other things.

The XLI files are made of a mixture of text and coding (program instructions). The text is to be printed (on conditions) or may be just explanatory comments. The coding is instruction to the PCA program to do certain tasks, for example look into charts and check positions

and aspects, change dates, add numbers, store ratings or whatever. The text and coding is separated following certain rules, which are described in this manual.

CREATING OR EDITING XLI-FILES

CHOOSING AN EDITOR

To be able to create or change a textfile you will need a text editor program. There are a lot of such programs available. You may keep it simple and use Windows Notepad. For bigger files you need a more capable editor.

If you cannot find a useful editor amongst your existing software you may use an ordinary wordprocessor (Word/Writer). The problem about this is, that wordprocessors are somewhat too advanced for this job, and you must deliberately ask it to remove all its internal instruction, underlining, different fonts etc from its files. Nearly any wordprocessor has instructions to create ASCII files which does exacly this. But there may still be concern about automatic linefeeds, page shifts and the like. Luckily a lot of text editor programs are free. Personally I use TextPad, but there are plenty

RUNNING THE XLI MODULES FROM ARGUS

The PCA manual describes 4 ways of accessing external modules. Though these methods can be used quite interchangeably, we would recommend:

The XLI-Menu: Just click into File | XLI-Menu and navigate to find the module to run.

A Macro: Press C and enter XOfile,ext. "file" means the first part of the filename, "ext" the extension, which is normally XLI. Note that file and extension are separated by a comma instead of the usual fullstop. Instead a fullstop must be inserted after the extension. For example to call the module EPHGEN.XLI you should enter XOEPHGEN,XLI.

An F-key: Use the makro key definition in the files pulldown menu, and enter a makro string into a free F-key definition field. The macro must be same format as for a single macro. Remember to save the F-key definitions by clicking Preferences | Save-to-disk.

NOTE: A running XLI module may be interrupted by ESC returning to the PCA main menu. This will normally do no harm, but some modules will change the input data or other system values, and by pressing ESC, you will not give them a chance to reset it to its original values.

XLI FILE BASICS

Having found a suitable editor program, you may begin to look at the XLI-files provided. You'll find some of them quite complex, and others quite easy.

An XLI-module or interpretation may be just one single file, but often several files are chained together. If chained, The XLI interpreter will read the complete chain. Chaining files is done at the end of each file by stating the name of the next file. If no name is stated, the file will be considered the last, and the XLI interpreter will put control back to the user and the main menu.

In more complex modules, several next-file names may be given and a menu or state of the program may decide which one to branch to

THE XLI PARAGRAPH

An XLI module is made up of one or more (chained) files, and each file is made up of one or more paragraphs. The paragraph is the basic building block of the XLI module.

Each paragraph must have some program instructions (Coding). These instructions can do a lot of things. For interpretations, they are used to calculate or find out whether the following text applies to the given chart, for example if the Moon is square to Mars. For modules they may setup menus, change characteristics of the PCA main program, control printer, print graphs or figures, control the XLI interpreter etc etc.

After the coding, there may be some text for printout. This text may be of two kinds: Commentary text or heading, and ordinary text.

The following example is a very short file just printing a text if the Sun is in Leo (in the latest Radix chart):

```
$                              $-line
1 PSI 5 =                      ;coding

*Sun in Leo                    Heading/comment (optional)

Dignity, good self-esteem..    Text (optional)
$                              Next paragraph
```

$-LINE: A paragraph must always start with a $-line. This is just a line consisting of one $-sign.

CODING: Next comes the coding part. This may be one or more lines starting just after the $-line. Beware, that no blank line just after the $-line is accepted. In that case, the module may terminate execution at that point.

The coding simply means:

```
1              Sun=1)
PSI            (planet in sign)
5              (Leo=5)
=              ("is").
```

You may find, that the instructions seem "reversed order". The idea is quite simple though and will be explained later. For the moment, you don't need to bother.

HEADING: The Heading/comment part is optional. If used, it must start with an asterisk. There may be just one or several lines. The headings are printed only if the module is called with `OI` (Alternative interpretation). The heading is terminated with a blank line.

TEXT: The text part comes last. It can be any text, and it may be as many lines as you wish. It is possible to insert templates for printing variable numbers or to a limited degree varible text strings, for example to put the person's name into the text. It is also possible to produce bar graphs.

COMMENTS

It is often very useful to put comments into your XLI modules. Comments are notes for the programmer and for others, who like to look into your files. If you after some time get back and want to add or change anyting, you will find, that you do not remember any more, what your coding is doing, so you will have to analyze each code. By inserting comments, you have a way of reminding yourself, what each part of your program is doing. The modules provided with this toolkit have a lot of comments.

Comments have no influence on the module itself. It will work the same with or without comments. You may find a slight reduction in speed if you put a lot of comments, because the XLI interpreter will need to identify and discard all the comment parts. Normally this will be no problem, and we strongly recommend commenting your codes.

To insert a comment, start with a semicolon and put the comment after this. The interpreter will ignore the semicolon and the remainder of that line.

If you need several lines of comments, each comment line must start with a semicolon.

The comments will normally be placed in the coding part of the paragraph to explain the workings of the coding. You may place the comments on the same line as the codes or in a line of its own. Look into the coding examples and see how it is done.

You may also place commentary text in the $-line. Functionally, only the $ sign is needed, so the rest of the line is free to use. This is a good place to put a note, what the paragraph is doing.

If you wish to place comments in the text or header parts of the paragraph, the semicolon must be at the very start of the line. That means, that you may not mix printable text and comments on the same line.

CODING YOUR FILES

The XLI interpreter uses a unique coding language. It uses a notation somewhat like FORTH or the old HP-pocket calculators. Technically it is called "Stack-orientated" or

”reverse polish notation”. You may find it hard to read, but implementing it in Argus was simpler than creating a complete language like JavaScript.

To program, you write a mixture of numbers and codes on one or more lines as you wish. You will find, that breaking your coding into many short lines is easier to read and understand. This also leaves room for adding comments.

To illustrate, the coding below calculates the distribution of the fire element in a chart using a point system:

```
$
14 PSI 0 1 5 9 IN 3 MUL      ;Asc in fire, 3 points
1 PSI 0 1 5 9 IN 3 MUL ADD   ;Sun in fire, 3 points
2 PSI 0 1 5 9 IN 2 MUL ADD   ;Moon in fire, 2 points
5 PHS 0 1 10 IN              ;Mars in 1. or 10. house
1 PHS 0 1 10 IN OR           ;Sun in 1. or 10. house
3 MUL ADD                    ;3 points
3 10 FOR                     ;count planets Mercury to Pluto
1 CNT PSI 0 1 5 9 IN ADD     ;if in fire, add 1 point each
NEXT                         ;end of count loop
11 STO 0                     ;store result in cell 11
```

Don't worry too much about the codes here. Their meaning will be explained later.

Here the codes are arranged so that each line works as a kind of "functional subunit". The comments are very useful as a reminder of what you are doing and will make it easier to read.

You may put up to 1000 characters into the coding field including comments. If you need more coding, you must spread it on more paragraphs. If a paragraph has no text fields, just coding, the interpreter will just continue on the next paragraph.

The stack orientation means, that you must adapt to the first-in first-out way of thinking. Adding 2 and 2 will look like this:

```
2 2 ADD
```

Note, that the numbers come first, then the instruction what to do with the numbers. So ADD will happily add whatever it finds on the stack, in this case the two numbers 2 and 2. These two numbers are removed, and the result, 4 is put back on the stack.

A parallel to this way of thinking is how you operate Argus itself. First you enter birth data, then you call the horoscope calculation. In other programs you may find calling a radix, you will get a dialog box for entering data.

WRITING TEXT LINES

The coding must always be terminated by one blank line. From then off and till the next $-line you may insert any quantity of text, limited only by disc space. We would recommend, however, that you keep your text in small paragraphs which are more manageable.

It is possible to merge a limited number of variable strings and numbers into the text, for instance if you wish to print planetary positions etc from the program.

LINE CONCATENATION

If you for some reason wish to compose a line of two or more parts, you may do so by line concatenation. If a line ends with a backslash (\), the normal carriage return and linefeed will be omitted, so that the following line will just continue at the end of the current (not printing the backslash of course). The second (or more) part(s) may even be added in next paragraph.

THE STACK

It was earlier mentioned, that XLI is stack-orientated, and that this means that you must put values first. Stack operation is beautifully simple and very powerful. However, because we normally write calculations in a different style (alge-braic notation), it takes time getting used to it.

Here is an example of an ordinary calculation task:

```
(5+8)*(8-3)
```

The result should be 65.

The parenthesis are necessary to assure, that the calculations are done in the correct order. In some systems, some of the parenthesis may be omitted, if there exists a definition that gives priority to certain operations (*/+-) over others. Parenthesis and operator hierarchy is in fact very akward and complicated, but we are used to looking at problems this way.

Actually, you may find, that different pocket calculators give different result to the same task due to different operator hiearchy used.

The same problem coded in XLI would look like this:

```
5 8 ADD 8 3 SUB MUL
```

This reflects the way you actually would solve the problem: add 5 and 8, keep the result while subtracting 8 and 3, then multiply the two results. No parentheses are needed, calculations are shown in the same order that they must be executed.

If the parentheses in the above example were moved like this:

```
5+(8*8)-3
```

The result should now be 66.

In XLI notation it would look like this:

```
5 8 8 MUL ADD 3 SUB
```

The instructions rather than the parenthesis are rearranged to reflect the changed execution order.

This brings us back to the stack. To do the above calculation, the system must have some means of keeping the intermediate results. Here five is kept while multiplying 8*8, before the addition is possible. This is done using the stack.

In stack orientated languages like XLI, you must first put the numbers on the stack, then add, multiply etc.

To illustrate the matter, we will look at the above example once more, showing the stack content for each operation:

```
$ coding  ;stack content top bottom
5              ;5
8              ;8 5
8              ;8 8 5
MUL            ;64 5
ADD            ;69
3              ;3 69
SUB            ;66
```

Each time you put a number in the coding, it will be put on top of the stack. The MUL and ADD functions use the numbers on the stack and place the result back on top.

So you do not really need to worry about what to do with the result. Just leave it on the stack till you need it next time.

You cannot overload or empty the stack, it is actually circular. It is your own responsibility, that the stack holds the numbers you need. On the other hand, you do not need to cleanup waste, when you are finished. There is however a limit, how many numbers, you can put on the same time, being circular, at some point, the last entered value will overwrite the first.

The size of the stack is 1024. So you may not put more than 1024 numbers on the stack before the first numbers will be over-written. This will rarely be any problem, but in case, the ISTKZ XLI code may be used to resize the stack to max 32000.

The XLI interpreter has a facility called the DEBUGGER. With the debugger switched on, you may check your coding step by step and check the contents of the stack. The debugger prints somewhatlike the example shown above:

The debugger window will also show values of strings and floating point numbers

MEMORY CELL ARRAY

This is an alternative to storing numbers on the stack. Because the stack is limited in size and is constantly changing, it will often be difficult to remember, how "deep down" the stack, your numbers are placed.

There are 3000 memory cells in XLI numbered 0-2999. There you may save numbers and calculation results for later use. You may however expand the limit using ISTOZ up to 32000.

The two codes STO and RCL are used to store and recall numbers from the memory array.

There are three main uses for the memory cell array:

1) store results, for example planetary positions etc.

2) create tables, for example the rulers of the 12 signs:

```
 5  1 STO     ;Mars     (5)  rules Aries       (cell 1)
 4  2 STO     ;Venus    (4)  rules Taurus      (cell 2)
 3  3 STO     ;Mercury  (3)  rules Gemini      (cell 3)
 2  4 STO     ;Moon     (2)  rules Cancer      (cell 4)
 1  5 STO     ;Sun      (1)  rules Leo         (cell 5)
 3  6 STO     ;Mercury  (3)  rules Virgo       (cell 6)
 4  7 STO     ;Venus    (4)  rules Libra       (cell 7)
10  8 STO     ;Pluto   (10)  rules Scorpio     (cell 8)
 6  9 STO     ;Jupiter  (6)  rules Sagittarius (cell 9)
 7 10 STO     ;Saturn   (7)  rules Capricorn   (cell 10)
 8 11 STO     ;Uranus   (8)  rules Aquarius    (cell 11)
 9 12 STO     ;Neptune  (9)  rules Pisces      (cell 12)
```

3) store parameter constants.

You may program a printout of aspects etc, using certain orb limits and a certain set of planets.

You may of course insert the orb limits etc. directly into the coding where needed. But if you store them in memory cells at the very start of the module, and recall them when needed in your coding, it is very easy to make modifications, because you do not need to search for the insertion place(s), you need just to change one number at the module start.

STO and RCL are used intensively throughout the utility modules provided

INTEGER NUMBERS

Most XLI coding works on integers. A separate set of codes are provided for decimal numbers,

The calculation:

```
5 2 DIV          (divide 5 by 2)
```

will produce 2, not 2.5. This is important to remember.

The integers may range from -32768 to +32767. These limits may seem strange if you do not know about binary numbers. But don't worry, it will rarely be any problem. If you try to add 1 to 32767, you will actually get the result -32768. So when the limit is reached the numbers start happily over from the other end. Of course this result is wrong and you should normally prevent your coding from producing numbers of that size.

With more modern systems, the integer range is actually expanded to double precision. Also in Argus, you may find it possible to pass the integer limitsmentioned above. For example you can create a loop counting 1 to 100000, but you should not rely on double precision for all functions witout testing it yourself.

CALCULATING MIDPOINTS AND ANTISCIONS

- The intang unit

The way integer overflow is handled as described above is actually exploited in PCA. Zodiacal positions are normally 0-360 degrees, and exceeding 360 degrees you will just start over from 0.

The intang unit is a way of holding angles using the full integer range, so that overflow in both directions are auto-matically handled. Such angles may be multiplied (harmonics) added and subtracted, never worrying about the problem of exceeding the circle.

A number of XLI functions use the intang unit, and of course there are also functions to turn them back into a readable format. One intang unit equals approximately 20 seconds of arc (19.77) which is better than the planetary accuracy of Argus.

The ITOM converts the intang units to a positive angle in minutes. In some cases you may want a positive or negative value of 0 to +-10800 (+-180 degrees). In that case, just use ITOMS instead of ITOM.

For example if you want to calculate an aspect angle, subtrac-ting two planets, for example Sun-Mars:

```
1 PPOS          ;Sun position
5 PPOS SUB      ;Mars-Sun angle (intang units)
ITOMS ABS       ;0-10800 positive
```

So the aspect angle will be the shortest angle between the two planets, and positive.

Also if you calculate signed orbs or declinations and latitudes, they are +- values rather than 0-360.

WRITING SIMPLE INTERPRETATIONS

As we have seen, the XLI paragraph consists of four parts: $-line, coding, heading and text.

When writing interpretations, these parts have the following meaning:

1. $-line paragraph delimiter
2. coding astrological rule
3. heading astrological explanation
4. text interpretation text

The heading is not mandatory, but it provides a means of allowing the end user to choose if he wishes an astrological explanation put into the interpretation text or just want the plain text printed.

Here's an example of a very simple interpretation paragraph:

```
$
8 PHS 2 =

*Uranus in the 2. house

Ups and downs in fortune, unsettled state, gains through
discoveries, mechanism, banks, railways, music and through
electricity.
$
```

The code line is the way to tell the PCA interpreter, that it should inspect the latest calculated chart to see, if Uranus was in the 2. house:

```
8 PHS       ;fetch the actual house position of Uranus
2           ;stack the desired house position
=           ;compare the two numbers (positions)
```

If the two numbers are equal, i.e. if `8 PHS` equals `2`, the result will be `1`. In XLI `1` means "true" and `0` means "false". If the result is true the text part will be printed, if false it won't.

The heading is the way to tell the reader (not the computer), which rules have been used. This will be printed together with the text part. There is an option in the program preferences to include or leave out the headings from an interpretation output.

You may leave out the heading or just put the explanation into the text itself, the interpretation will function anyway, but in that case the end user will not be able to have it printed optionally. The heading may be any number of lines and is terminated by a blank line.

The text part may be any number of lines and is terminated by the $-line starting the next paragraph.

CODING EXAMPLES FOR INTERPRETATIONS:

If you call the module editor (see the chapter on using and writing interpretations) and look into the interpretation skeletons, you'll find a lot of useful examples of coding basic astrological rules. When you choose a rule for editing, you'll see a window at the top of the screen showing the coding for that rule.

Here is a complete list of the codes referencing the stored chart(s):

Simple codes:

```
PSI             planet in sign
PHS             planet in house
RX              planet retrograde
HSI             house in sign
HRU             house ruler
APOW            aspect power
ANUM            aspect number
AORB            aspect orb
PDEG            planet position (degrees)
BDPR            print birth data
NPR             print name
```

Advanced codes

```
AZP             aspect test primitive
PPOS            planet position (intang unit)
HPOS            house position (intang unit)
PV              planet speed (intang unit)
HV              house speed (intang unit)
PLA             calculate single planet (intang unit)
XPLA            retrieve additional planetary information
HOUSE           calculate single house (intang unit)
JD              get julian date
DDIF            find difference between two dates
BHUS            house position (universal)
KUN             get 3 closest Kündig sections
```

Examples using the advanced codes are given later.

Now let us look at some examples of common coding tasks:

```
2 PSI 10 =      ;Moon (2) in Capricorn (10)
6 PHS 11 =      ;Jupiter (6) in 11.th house
3 RX            ;Mercury (3) retrograde
7 RX            ;Saturn (7) retrograde
6 HSI 8 =       ;6.th house cusp in Scorpio (8)
1 HSI 12 =      ;Ascendant(1)in Pisces (12)
14 PSI 12 =     ;Ascendant(14)in Pisces (12)
```

The last two examples are actually testing the same thing. The ascendant is house no. 1 and "planet" number 14. Six of the houses have "planet" numbers, so they can be tested using the planet rule codes.

For a list of the numbers of planets, houses, aspect numbers etc, see the function reference section, where these numbers are listed in tabular form.

Please note that the results above are all YES or NO: Retrograde or not retrograde, equal to or not equal to etc. YES (1) or NO (0) results are used when a text must be printed or not printed. The code 2 PSI produces a result between 1 and 12. To turn it into a YES or NO you must test this result againts a fixed number using = (equals) > (greater than) < (less than) or IN (member of a selection).

More examples:

```
 4 HRU 6 =      ;Jupiter rules 4.th house
 4 HSI RUL 6 =  ;Same result as above
 3 PSI RUL 8 =  ;Uranus dispositor for Mercury
 2 6 ANUM 3 =   ;Moon square Jupiter
 2 6 ANUM 4 =   ;Moon trine Jupiter
 1 7 ANUM 1 =   ;Sun conjunct Saturn
 1 7 AORB ABS
  10 < NOT      ;Sun Saturn aspect orb 1 degree
```

The aspect examples do not care about the order of the aspec-ting planets, so you may write 2 6 ANUM or 6 2 ANUM as you like.

Note that AORB calculate aspect orbs to one tenth of a degree, so 1 degree=10 units, 2 degrees = 20 units etc. AORB is signed, so you must change it to a positive value (using ABS) before the test.

For these aspect tests to work, they must be within the orb limits given in the PCA orb installation and main menu, e.g. pressing key A in Argus to get the aspects, only those within the defined orb limits will be shown.

```
 14 5 APOW 8 < ;ASC in close aspect to Mars
```

The APOW (aspect power) rates an aspect from 0 (just at the orb limit) to 10 (exact). So in this example it must be nine or ten (8 is less than the power) The orb limit is the one, you entered in the PCA menus. Actually, it is similar to the thickness of the aspect lines in the drawn chartwheel.

```
 8 PDEG 154 =  ;Uranus between 4-00 and 4-59 in Virgo
```

This may be used for degree astrology. You may also do some arithmetic and find sign, decanate etc from the PDEG code. It works only in whole degrees, though. Later we will describe the code PPOS, which is more accurate working in intang units.

```
 NPR BDPR              ;Print the natives name and birthdata
```

NPR prints the name (1 line used), and BDPR the data using 5 lines.

To get more control, another code BDATA will let you print parts of the birthdata only.

ARITHMETICS

The above examples are all very basic single astrological testing rules. The XLI toolkit provides a number of math functions, so that you may combine rules, count points, planets etc etc..

COMPARING

>	greater than
<	less than
=	equal
<>	not equal
NOT	not

In the above examples we have often used the = (equals) sign to test for certain house numbers or signs. Another example showed the use of < (less than) to see if a power was above a certain value.

To know if a text paragraph must be printed or not, you need an end result of YES of NO, not a value. Having found the house number of Jupiter, you will need 12 texts, one for each house, and each text paragraph must have the Jupiter house tested if it is 1, 2, 3... etc.

You may ask why >= (greater than or equal) and <= (less than or equal) functions are missing in the above list. Correct, they are missing. But you may construct them from the others. For example to test if Jupiters house position is greater than or equal to 7:

```
6 PHS 7 > NOT ;Jupiters house not less than 7
6 PHS 6 <     ;Jupiters house greater than 6
6 PHS 7 <
6 PHS 7 = OR  ;Jupiters house greater than or equal to 7
```

The code NOT changes YES (1) to NO (0) or vice versa.

COMBINING RULES

Say that you wish to test if either the Sun or the Moon is in the tenth house:

```
1 PHS 10 =    ;Sun (1) in house 10
2 PHS 10 =    ;Moon (2) in house 10
OR            ;either ?
```

The OR code combines the two conditions (YES/NO results) already done. As usual, you must have the two results first before you use the OR code.

You could also have asked if the Sun and Moon are BOTH in the 10. house:

```
1 PHS 10 =     ;Sun (1) in house 10
2 PHS 10 =     ;Moon (2) in house 10
AND            ;both ?
```

The codes OR and AND combines two rules to "either" and "both".

You may expand the number of conditions by adding lines. Say that you want to test if Mars is in a mutable sign:

Version 1

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
OR             ;either
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
OR             ;either
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either
```

Please note the seemingly uneven placement of OR. The first OR tests "either" in Gemini or Virgo, the next "either" tests the first result or Sagittarius, and the last test the second result or Pisces. This must be done, because OR can only test two condi-tions at a time. You may use other setups, which will produce the same result:

Version 2

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
OR             ;either
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either
OR             ;either
```

Version 3

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either Sgr or Psc
OR             ;either above or Vir
OR             ;either above or Gemini
```

Please note that in the last version you put 4 conditions on the stack and do the combining afterwards. Only the last OR will reach down to the bottom and fetch the first (Gemini) condition.

Remember also, that using a lot of `OR`'s will expand the pro-bability of a final YES, and using a lot of `AND`'s, your final result will most often be NO.

COMBINING "AND" AND "OR"

Say that you wish to test if both Sun and Moon is in water signs:

```
1 PSI 4 =      ;Sun in Cancer
1 PSI 8 =      ;Sun in Scorpio
1 PSI 12 =     ;Sun in Pisces
OR OR          ;either of above = Sun in water
2 PSI 4 =      ;Moon in Cancer
2 PSI 8 =      ;Moon in Scorpio
2 PSI 12 =     ;Moon in Pisces
OR OR          ;either of above = Moon in water
AND            ;both Sun in water and Moon in water
```

TESTING ELEMENTS ETC.

Testing for water signs, mutable signs etc are a bit tedious, because you must test each single sign. There are a few tricks you may use:

```
2 PSI 0 4 8 12 IN           ;Moon in water
```

The code `IN` will check your Moonsign against a set of possibi-lities in one go. The result of the `IN` is always YES or NO, so you do not need the equals sign here. There are a few things to remember using `IN`:

1) You must ALWAYS start with a zero before the values to test against. The zero works as a delimiter so that `IN` can know how many values back in the row to use. So you cannot use `IN` to test equal to zero. Luckily signs, houses and planets are numbered 1 upwards.

Please note, that you can not include a zero in the test, the zero being reserved as a data stop. To test if a value is 4 or 0, you must use a workaround, either test zero separately or you may in some cases add one to the value before testing, but that will make the code less readable.

The other trick testing elements is purely mathematical:

```
4 PSI 4 MOD 1 =    ;Venus in fire
4 PSI 4 MOD 2 =    ;Venus in earth
4 PSI 4 MOD 3 =    ;Venus in air
4 PSI 4 MOD 0 =    ;Venus in water
```

The `MOD` (modulus) function calculates the remainder of the division with `4`, which will be `0`,`1`,`2` or `3` then starting over. In fact this will usually be the simplest and most efficient way, but you may find it less obvious to perceive.

COMPARING TWO CHARTS

PCA have two sets of stored positions for charts, one for radix, and one for all other charts.

If you want the positions of both for comparison, you will need a means of choosing which of the two memory blocks to access:

Planet 1-18:	Latest calculated chart (any kind)
Planet 21-38:	Latest calculated radix chart
House 1-12:	Latest calculated chart (any kind)
House 21-32:	Latest calculated radix chart

As you can see, you just add 20 to the planet number to get the radix. So if latest calculation was for example a progressed chart, the progressed Moon will have number `2`, and the radix Moon will have number `22`.

The following examples show which codes this will work for assuming a progressed chart as the latest calculation:

```
34 PSI 5 =     ;Radix ASC (34) in Leo (5)
14 PSI 5 =     ;Progressed ASC (14) in Leo (5)
29 PHS 8 =     ;Radix Neptune in 8. radix house
9 PHS 8 =      ;Progressed Neptune in 8. progressed house
6 HSI 4 =      ;6th progressed house cusp in Cancer
26 HSI 4 =     ;6th radix house cusp in Cancer
11 PDEG 316 =  ;Progressed Moon's node between 16.00-16.59 Aqr
27 RX          ;Radix Saturn retrograde
7 RX           ;Progressed Saturn retrograde
23 HRU 5 =     ;Mars rules radix 3rd house
3 HRU 5 =      ;Mars rules progressed 3rd house
2 33 ANUM 2 =  ;Progressed Moon opp. radix MC
2 13 ANUM 2 =  ;Progressed Moon opp. Progressed MC
1 5 AORB
ABS 20 >       ;Progr. Sun aspect Progr. Mars orb<2 deg.
1 25 AORB
 ABS 20 >      ;Progr. Sun aspect radix Mars orb<2 deg.
21 25 AORB
ABS 20 >       ;radix Sun aspect radix Mars orb<2 deg.
1 21 APOW
10 =           ;Progr. Sun aspect radix Sun exact
```

Further this planet numbering will work on the more advanced codes `PPOS`, `HPOS`, `PV` and `HV`. One example will be given here though. When comparing charts, it is often important

to test which PROGRESSED house a RADIX planet is in and vice versa. So even if we do not explain it in detail, here are some coding examples for this problem:

```
24 PPOS
2 BHUS 4 =    ;radix Venus in progr. 4th house
4 PPOS
1 BHUS 6 =    ;progr. Venus in radix 6th house
31 PPOS
2 BHUS 1 =    ;radix node in progr. 1st house
```

As you can see, you must use two codes: `PPOS` and `BHUS`. The first number is the planet (using the expanded numbering), and the second number between the two codes is `1` for radix house and `2` for progressed house.

This may of course be used for synastry as well, if the latest calculation was a 2.nd radix (option `T` in the PCA main menu). Then the above lines would mean e.g. "John's Venus in Susan's 4th house", "Susan's Venus in John's 6th house" etc.

COUNTING, LOOPING AND BRANCHING

FOR	start of FOR-loop
CNT	get FOR-loop counter
NEXT	end of FOR-loop
XIF	exit FOR-loop prematurely
ENDIF	End of if-clause
IF	Start of if-clause
ELSE	Alternative part of if-clause
NFN	set file branch index
WAIT	withhold heading
CONT	skip withhold

You will often need to repeat the same operation a number of times. For example, you may wish to count the number of planets in water signs. This could of course be done the hard way:

```
1 PSI 0 4 8 12 IN
2 PSI 0 4 8 12 IN ADD
3 PSI 0 4 8 12 IN ADD
4 PSI 0 4 8 12 IN ADD
5 PSI 0 4 8 12 IN ADD
6 PSI 0 4 8 12 IN ADD
7 PSI 0 4 8 12 IN ADD
8 PSI 0 4 8 12 IN ADD
9 PSI 0 4 8 12 IN ADD
10 PSI 0 4 8 12 IN ADD
```

For each line coming out TRUE (`1`) the result will be incremented. Now if you start putting a null result (on the stack) and add the others, you'll get this slightly changed setup:

```
0
```

```
1 PSI 0 4 8 12 IN ADD
2 PSI 0 4 8 12 IN ADD
3 PSI 0 4 8 12 IN ADD
4 PSI 0 4 8 12 IN ADD
5 PSI 0 4 8 12 IN ADD
6 PSI 0 4 8 12 IN ADD
7 PSI 0 4 8 12 IN ADD
8 PSI 0 4 8 12 IN ADD
9 PSI 0 4 8 12 IN ADD
10 PSI 0 4 8 12 IN ADD
```

Now you have ten lines, with nearly the same coding, exept for the first number which increments one for each line, i.e. counts from 1 to 10.

You may setup a COUNTING LOOP to do this job more elegantly:

```
0
1 10 FOR
1 CNT PSI 0 4 8 12 IN ADD
NEXT
```

The 10 lines are now replaced by just one line using a counter for the planet number. To make the counter count, these two lines were added:

```
1 10 FOR          ;starts the counting loop
NEXT              ;loops back until end of count
```

The line(s) in between will go into action 10 times. The `1 CNT` will use the incrementing count value (`1`-`10`). The end result will be exactly the same as in the first example, not using a counter. It will not process faster though, actually rather slightly slower, but it saves you a lot of coding.

Counting loops may count up (`1 10 FOR`) or down (`10 1 FOR`), you may also count using negative values (`-7 8 FOR`). You may easily put the `NEXT` code several paragraphs further on, so that a lot of paragraphs will execute inside the loop. It only has to be in the same file. For example:

```
$
1 3 FOR
1 CNT NUMS

This is line #
$
NEXT

$
```

This will print the following:

```
This is line 1
```

This is line 2
This is line 3

The `NEXT` code is here placed in the following paragraph, so that the text will be printed for each count. Don't worry too much about the `NUMS` code. This is used to print the counter and will be discussed later.

You may setup several (up to 10) counting loops inside each other. For example, if you wish to count the aspects from all planets to all planets, you will need two planet counters: The first should count from Sun to Neptune, the next should count from the current first counter to Pluto. If this is not immedi-ately clear, try to setup a table: Sun-Moon, Sun-Mercury,.... Neptune-Pluto , and check the changing planet numbers.

This example will count the squares in the chart:

```
0             ;no squares
1 9 FOR       ;outer loop
1 CNT 10 FOR  ;inner loop
1 CNT 2 CNT   ;planet 1 and planet 2
ANUM          ;aspect number
3 =           ;is it a square
NEXT          ;end inner loop
NEXT          ;end outer loop
```

Note, that `1 CNT` is the first counter, `2 CNT` is the second counter. They are numbered in the succession they are created, i.e. the outermost counter will be number 1, and in this example `2 CNT` will be the innermost. When using "nested FOR--loops", that is several loops inside each other, be well aware which counters you are using.

Counting loops are used extensively in the XLI user modules.

IF-ENDIF

Sometimes you must have something calculated or written out, only on a certain condition. Say for instance, that you are setting up a counting loop for the 10 planets and the two angles MC and ASC. So counting from 1-14, you wish to skip the Moon's node (11) and the part of fortune (12). The solution is:

```
1 14 FOR      ;setup complete coung
1 CNT
0 11 12 IN
NOT IF        ;check range
....          ;your code here
ENDIF
NEXT
```

The dotted line should be replaced by your aspect tests or whatever. The range check first looks if the loop counter is equal to 11 or 12, then reverses this condition to "not equal" to 11 or 12. If the latter is the case the following lines will be executed, if the counter in fact equals the unwanted values, the lines will be skipped until the ENDIF code.

If you have programmed in other langauges, you may look for an ELSE code, so that you may add other lines of code executing for the node and the part of fortune. There is also an ELSE code in XLI:

```
1 14 FOR      ;setup complete coung
1 CNT
0 11 12 IN
NOT           ;check range
1 DUP IF      ;IF not 11 or 12
....          ;your code here
ELSE
...           ;your code here
ENDIF
NEXT
```

The ENDIF doesn't have to be placed in the same paragraph, but must be within the same file.

You may have several IF-ENDIF constructions within each other (nested):

```
IF
IF
IF
ENDIF
ENDIF
ENDIF
```

Warning: There must be exactly the same number of ENDIF's as IF's. The first IF corresponds to the last ENDIF etc. Sometimes a module seem to stop working somewhere in the middle because of a missing ENDIF.

Warning: Often you will mix FOR-NEXT loops and IF-ENDIF con-structions. But never make them overlap:

```
Okay:         FOR IF ENDIF NEXT
Bad:          FOR IF NEXT ENDIF
Okay:         IF FOR NEXT ENDIF
Bad:          I FOR ENDIF NEXT
```

There is a special code XIF to exit a FOR-NEXT loop prematurely on a certain condition. An example of this is given in the section of creating menus, and in the example of listing graphic transits.

FURTHER ARITHMETIC

Math:

```
  BOO           integer to boolean
  ABS           absolute value
  CHS           change sign
  ADD           add
  SUB           subtract
  MUL           multiply
  MULT          multiply by fraction
  DIV           divide
  MOD           modulus
  DIVR          divide with remainder
  MAX           take maximum of two values
  MIN           take minimum of two values
```

Stack:

```
  DUP           duplicate value on stack
  FETCH         fetch value on stack
  ZZO           set zero offset on stack
  GET           get number on stack (fixed offset)
  PUT           put number to stack (fixed offset)
  XY            exchange numbers on stack
  INC           increment stack pointer
  DEC           decrement stack pointer
```

As soon as you wish to do something more than just simple tests and counts, you will need to perform calculations. We will only comment a few of the math and stack functions here. If you need something, look at the list and refer to the FUNCTION REFERENCE for a specification how to use any of these functions if you think it will serve your needs.

What we will do here is to mention some common problems and how to solve them.

CHANGING INTANG UNITS TO DEGREES MINUTES AND SIGN

A number of the advanced astrological functions produce results in intang units. We will show how to print such values in a readable format:

```
  3 PPOS        ;Mercury position (intang)
  ITOM          ;convert to minutes of arc
  1800 DIVR     ;convert to sign and minutes
  1 ADD         ;renumber signs to start with 1
  XY            ;exchange sign and remaining minutes
  60 DIVR       ;convert minutes to degrees and minutes
  NUMS          ;prepare printout
```

```
Position is degrees: ##  Minutes: ##  Sign no.: ##
```

The NUMS will print the total 3 results produced by the calculation in the text field shown. For each template (##) the last result on the stack will be removed and printed. Therefore the results must be calculated in reverse order (the first number to print must be calculated last).

Now to the arithmetics: The DIVR produces two results: the remainder and the quotient of the division. Say, that you have a position of 9-28 Leo. This is 7768 minutes total. So we will repeat the calculation with this example showing the stack values:

```
3 PPOS          ; 23569
ITOM            ; 7768
1800 DIVR       ; 4 568
1 ADD           ; 5 568
XY              ; 568    5
60 DIVR         ; 9 28   5
```

The result(s) are thus 9 28 5 which can then be printed using the NUMS code.

As you can see from for example 568 60 DIVR, that the remainder (28) is calculated first, then the quotient (9) is placed on top. The sign number remains untouched beneath the following calculations and results.

It may be necessary for you to draw a sketch of the stack as shown above to be sure, that you get it right. You may test your coding on screen by adding the DEBUG code, then calling your test module from PCA. DEBUG will then display you the actions on the stack step by step as shown above.

MULT - SCALING YOUR VALUES:

The integer arithmetic in XLI have certain serious limitations.

Say for instance, that you wish to change minutes of arc back to intang. No function in the XLI provides this facility. What you really need is to divide your minutes with 21600 and multiply by 65536.

But dividing by 21600 will produce 0 instead of a decimal number between 0 and 1. And if you try to multiplying with 65536, you will get a serious overflow problem, because the integers do not range further than 32767. You may reduce the fraction, e.g. using a quarter circle instead: multiply by 16384 and dividing by 5400.

The MULT code lets you multiply and divide in one go, keeping the highest accuracy possible:

```
16384 5400 MULT
```

You cannot write 65536 21600 MULT, because the number 65536 does not exist in the XLI, the allowable range is -32768 to +32767.

You will often need this code if you plot graphs. The XLI can turn the system into graphics mode and draw instead of print. Positioning the drawing pen will often need scaling.

MAX AND MIN

Say that you calculate element strength, and wish the result to be within certain limits, for example -99 to 99, so that the value -99 means -99 or less, and the value 99 means that the value is 99 or more. After having calculated your points, the following coding will do just that:

```
-99 MAX 99 MIN
```

Another use may be to find the strongest of a number of rangings. Say that you have calculated points for the four ele-ments and put them into the storage cells 1-4.

One question may then be: what is the strongest element power. The answer is produced by the following coding:

```
1 RCL 2 RCL MAX 3 RCL MAX 4 RCL MAX
```

Another question may be: Which element is strongest?

```
1             ;assume fire
1 4 FOR       ;setup counting loop
1 CNT RCL     ;get current element
2 DUP RCL     ;get strongest found till yet
< IF          ;if the current is stronger
DEC 1 CNT     ;exchange with current count
ENDIF         ;end condition
NEXT          ;end loop
```

The result will be in the range `1-4` (fire-water). If the strongest point score is shared by more elements, the first of these is chosen by the routine.

Note the code `DEC`. This will dispose the current result. So the construction `DEC 1 CNT` removes the till yet strongest element number and inserts the current instead.

POWERCONTROLLING ARGUS

```
MEGET         get value from main menu
MEPUT         put value to main menu
NPUT          put name to main menu
PSTAT         enable or suppress all output
FUNX          call single PCA main menu option
CML           execute defined macro (destructive)
CALL          call defined macro (non-destructive)
```

One of the strong features of the XLI interpreter is, that PCA may be operated by a file instead of interactively by you.

Even without XLI you have the macros, which can execute complex tasks from one keypress. A macro can also call the XLI interpreter.

But the XLI interpreter can also in turn set up its own macros and run them. You are not allowed though to let the XLI use a macro to call XLI once more, that will not work.

Ultimately you could have another program running, which created a PCA XLI file, started PCA automatically to calculate planets, ephemerides, charts, chartwheels etc, storing the results on disk, and finally returning to the calling program, which could then pick up the files and retreive the results.

There are two codes calling commandlines. The `CALL` is normally the one to go for. It will execute immediately.

The other call is `CML`. This is not as useful as `CALL`. It will in fact replace an existing macro with a new one, and it will not execute until after the XLI module has finished. So say you inserted first a `CML` and then a `CALL` in your XLI file. The `CALL` would (surprise) execute first, leaving `CML` sleeping and undisturbed. When the XLI module finish, the `CML` will then continue. But remember, that `CML` will destroy any previous macro.

The calls are coded like this:

```
$
CALL

RAV
$
```

Note that the macro itself is put into the text field of the paragraph, not in the coding. It has to be the first line in the text field. Normally, you should not put real printable text into such a paragraph.

CONTROLLING OUTPUT

Running complex jobs, you may not always need everything printed on the screen. For example, you may need the positions for a radix and a progressed chart for further calculations and display. This could for example be done like this:

```
$
CALL

R1T.P1R.
$
```

This will run the radix using the current menu data, then get todays date from the internal clock (`1T.`), run the progressed and finally restore the radix time (`1R.`). But say, that you

have some display or print running in your XLI module, that you would not like to be disturbed by the printout of those two charts. The answer is simple:

```
$
CALL

(R)1T.(P)1R.
$
```

Just put parenthesis around the calculations, you do not wish to have displayed or printed.

MANIPULATING MENU INPUT DATA

You may directly access the input data in the PCA main menu. This is done by the `MEGET` and `MEPUT` codes. Every bit of birth data in the input menu has a number, for example:

```
0 MEGET        ;fetch day
1 MEGET        ;fetch month
2 MEGET        ;fetch year
3 MEGET        ;fetch BC flag
```

A complete list is given in the function reference. You may then have the XLI input its own data, run charts etc. Say for instance, that you would like to do the progressed chart for 5 years onwards from the current:

```
$
0 19 FOR       ;save the current menu data
1 CNT MEGET    ;get an item
1 CNT STO      ;store in memory cell
NEXT
CALL           ;call (invisibly) radix, then enter current time

(R)1T.
$
2 MEGET        ;current year
1 DUP 4 ADD    ;end year
FOR            ;count the five years
1 CNT MEPUT    ;insert the year
CALL           ;show progressed chart

P
$
NEXT           ;loop back
0 19 FOR       ;restore the original data
1 CNT RCL      ;fetch from memory cell
1 CNT MEPUT    ;put back to menu
NEXT
```

```
$$$
```

Here we have used another method of saving and restoring the original menu data. Of course, in this case you could as well just have called a macro of `1R`, which would have restored the data of the last radix. But in other cases, for example, if you wish to run other radixes in between, you must use this more safe method. Even this method may not work, however, if you interrupt the XLI prematurely pressing ESC and returns to human control before the module has finished.

Name insertion into the input menu can be done using a macro call:

```
$
CALL

OBEVERLY.
$
```

...to insert the name BEVERLY, but if the name is manipulated using the string handling routines the `NPUT` code can be used for inserting a name from the string array. See the section of string handling and refer to the function reference for the use of `NPUT`.

The sex field is accessed through `7 MEPUT`/`7 MEGET`.

The house system cannot be accessed through `MEGET` and `MEPUT`. You must use the macro method.

GRAPHIC TRANSITS

```
NDATE          convert graphic transits sector to date
GTR            call user defined graphic transit
RTA            reset transit aspect list
NTA            get next transit aspect
```

If you like the graphic transit facility, you may setup other versions with special features using the `GTR` code. You'll find a description of this in the function reference.

Any kind of graphic transits will save the aspects found to memory. These may then be retreived, analysed, displayed or interpreted by an XLI module. The aspects can be retreived one by one in the succession, they appear. For each aspect, the start and end time (approximately), planet numbers and aspect type is available.

This example will print out the aspects:

```
$
2 CARY
```

```
,Sun,Moo,Mer,Ven,Mar,Jup,Sat,Ura,Nep,Plu,Nod,Ptf,MC,ASC
,Cnj,Opp,Sqr,Tri,Sex,ssq,ses,qqx,ssx,
$
1

No    From         To          Aspect:
-----------------------------------------
$
RTA           ;reset aspect retrieve counter
1 10000 FOR   ;loop all aspects
NTA           ;get next aspect
1 DUP NOT XIF ;exit if last one
4 FETCH 20 MOD;get radix planet and adjust planet number
5 FETCH       ;get aspect number
15 ADD        ;adjust to aspect name index above
5 FETCH       ;get transit planet (number no change)
5 FETCH NDATE ;convert end sector to date
7 FETCH NDATE ;convert start sector to date
1 CNT         ;fetch overall counter
NUMS          ;display all values

####  ## ## ####   ## ## ####  @@@ @@@ @@@
$
NEXT
1

-----------------------------------------------
$$$
```

The first lines are defining the planet and aspect names for printout purposes.

The stipulated lines are always printed, so the coding for this is just `1` (YES).

The `1 10000 FOR` is not counting 10000, it is just to be sure to have enough looping power. The XIF will exit at the last aspect.

The `RTA` resets the aspect pointer, so that the next one retrie-ved will be the first.

The `NTA` will get one aspect off the queue. Now a lot of `FETCH` codes are used. That is because the results must appear in the correct succession for printout, and some of them must be adjusted.

Here is a breakdown of the result stack to show, what happens:

```
NTA           sect1 sect2 radx tran asp
4 FETCH       tran sect1 sect2 radx asp
20 MOD        tran20 sect1 sect2 radx asp
```

```
5 FETCH       asp tran20 sect1 sect2 radx
15 ADD        asp15 tran20 sect1 sect2 radx
5 FETCH       radx asp15 tran20 sect1 sect2
5 FETCH       sect2 radx asp15 tran20 sect1
NDATE         d2 m2 y2 radx asp15 tran20 sect1
7 FETCH       sect1 d2 m2 y2 radx asp15 tran20
NDATE         d1 m1 y1 d2 m2 y2 radx asp15 tran20
1 CNT         cnt d1 m1 y1 d2 m2 y2 radx asp15 tran20
```

The names for the result values are quite abbreviated. If you are in doubt what they mean, look at the explanation of the coding above, and if needed refer to the definitions in the function reference part.

This works also with the collective transits, so you may use the above to setup a collective transit list.

Aspects, that repeat during the period will be listed several times. The transit interpretation skeleton has a mechanism for avoiding repetition. The method is quite complex, so don't try to analyze the code, unless you need a real challenge.

SETTING UP YOUR OWN MENUS

The following simple menu is a leftover from DOS days where you had only keyboard, no mouse.

```
MENU          setup screen menu
OPT           wait for keypress from defined set
WKEY          wait for keypress
XIF           exit loop
```

The ability to setup choice menus means, that you could expand the PCA into a really huge program with a limitless number of choices. The toolkit itself adds the two menus: The user-module menu (`XS`, file TTUS.XLI) and the interpretation brancher menu (`OI`, file TTFLET.TXT).

You may continue this menu-branching as much as you wish, and all the branching will be programmable using the macros.

NOTE: There is one limitation: You cannot use the `CALL` code to make XLI call other XLI modules. This would overload the XLI interpreter. But you may use the `CML` code, which executes the macro only AFTER the current module has finished.

Here is a very simple example of a menu choosing between two further XLI modules, A and B:

```
$
MENU

A  select module A
```

```
B  select module B
X  back to PCA main menu
$
0 88 66 65 OPT NFN

$$$
A.XLI
B.XLI
```

The MENU code opens a dialog in the middle of the screen. You may write anything in the text part, it works only as a reminder of the possible keypresses.

Only the keypresses defined in the list before the `OPT` code are accepted. The `0` value in the start of the line MUST be present, and just works as a delimiter telling where the list starts. The following values until the `OPT` code are the numbers of the valid keys. `A` has number `65`, `B` has number `66` and `X` has number `88` (the ASCII numbers of the key symbols).

The branching itself is obtained by the `NFN` code. The result of `OPT` is `1`,`2`,`3`,... etc depending on the key codes' position in the list. `NFN` will preset the branch file pointer to 1st, 2nd, 3rd... filename after the `$$$` line.

Please note that there is no filename for choice `X`. This should have been inserted after B.XLI, but is left blank. When XLI finds a blank filename to branch to, it returns to the main menu.

You do not necessarily need file branching to select different things. You could also have checked the `OPT` result with `IF-ENDIF` constructions.

```
$
0 66 65 OPT
1 DUP          ;duplicate choice result
1 = IF
1 3 CONFX
ENDIF          ;If A pressed, set degrees ON
2 = IF
0 3 CONFX
ENDIF          ;If B pressed, set degrees OFF

$$$
```

The option result is duplicated for two tests, result=1 or result=2. If you need to expand the number of tests, you must add one (`1 DUP`) duplication before each test line except the last, to keep one copy of the result available.

NOTE: This is a very primitive menu developed for DOS. With Argus, you can use MENUX to create more interesting menus with buttons and mouse clicks.

DISPLAYING CHOICES IN THE MENU

The MENU mechanism is not elaborate enough to let you enter values or strings, but you may enter choices and have them displayed.

The following examples demonstrates the use of MENU

```
$
1 CARY          ;define the strings to display in the menu

,>,
$
0 5 CONFX
1 DUP 5 CONFX ;get the system variable
0 STO
1 10000 FOR   ;setup "endless" loop
0 RCL 2 =     ;National
0 RCL 1 =     ;Latin
0 RCL 0 =     ;Symbols
NUMS          ;display strings
MENU          ;start the menu

@S    Symbols
@L    Latin abbreviations
@N    National abbreviations
X    SAVE and EXIT

$
0 27 78 76 83 13 88 ;get option from keyboard ESC S L N CR/x X
OPT
1 DUP
0 1 2 6 IN
XIF           ;exit loop if key 3 pressed
3 SUB 0 STO   ;save selected option (0-3)
NEXT          ;loop back
0 RCL 5 CONFX ;set the system variable to chosen value
$$$
```

The @ fields in the menu will display one of the strings defined by the CARY code in the start, These strings are simply:

string 0: a space” ”
string 1: an marker ”>”

So in the menu displayed, the chosen option will have a ”>” before the letter.

Pressing one of the three keys S, L or N on the keyboard will change the choice and loop back showing the menu once more with the new choice.

Two more options are included in OPT, that is 13 and 88 whic are the keys ENTER and X, which will both exit the loop and save the new system variable.

Clicking the x in the top-right corner of the menu windows with the mouse will make OPT return the value 2, which in this setup equals the ENTER key choice

This is another example using the same technique to input a number. The `MENU` mechanism is not really suited for that job, but as you'll see, even if it is clumsy, it works:

```
$
0 0 STO        ;reset input number to 0

$
1 10000 FOR    ;setup endless loop
0 RCL          ;current number to display
NUMS           ;prepare number display
MENU           ;setup as menu

ENTER NUMBER: #########
$
0 13           ;accept ENTER
57 56 55 54 53 52 51 50 49 48    ;and digit keys
OPT            ;get key
1 SUB          ;set range to 0-9 (ENTER=10)
1 DUP 10 = XIF;exit if ENTER pressed
0 RCL
3275 > IF      ;avoid number overflow
0 RCL 10 MUL   ;shift one digit left
ADD            ;add new digit
0 STO
ENDIF          ;store new value back
NEXT           ;end loop

$
0 RCL          ;display end result
NUMS


RESULT: #########
$$$
```

You could prune this a bit by adding a test for the backspace key to remove one digit, the minus key to change sign (in which case you should expand the range test to avoide

negative overflow) etc. It should in theory also be possible to change the coding to input strings using the string array and string handling functions.

Setting up menus with frames and things will make your XLI module look much more professional, even if it does not produce smart mouse-driven pop-up and pull-down menus in colours. The examples and modules in this toolkit are kept very basic, to make them easier to understand.

PRINTING VARIABLE STRINGS AND NUMBERS

NUMS display numbers and strings

We have often in the foregoing seen the `NUMS` code used to print values or names of planets, aspects etc. inserted in the text field.

The text field of the paragraph will normally be fixed, that is, you decide exactly, how it is going to appear, and one printout of the same paragraph will look identical to the next. Used for ordinary interpretation, the text fields are varied from chart to chart only by their succession.

The exception is the variable insertion. A variable is a number or bit of text (a string) which may change depending on for instance the chart. You may for example have printout of the planetary positions. The planet names will be fixed, but the degrees, minutes and sign name will vary.

```
Jupiter ## ## @@@

Saturn  ## ## @@@
```

The letters will appear as they are, but the `##` and `@@@` fields are TEMPLATES, that will change according to the current results on the stack.

Templates are of two kinds: numeric templates: `####` and string (text) templates: `@@@@`. You enter a template as a continuous row of `#`- or `@`- characters.

You may think of the template as a window or "hole" in the screen or paper, where you can see the current value of changing numbers or text strings.

To use the templates, you must place the code `NUMS` somewhere in the coding for that paragraph. The coding itself must provide the numbers or strings you wish to display. If the coding produces the result `3`, then a `####` template in the text will display the number `3`, and alternatively `@@@@` will display string number 3. Text strings have numbers, which will be explained later.

You may display up till 64 values/strings in one paragraph. You must calculate as many numbers, as you put templates into the text field.

Your coding must provide the numbers you need in reverse order, the first template will display the last value put on the stack

A very simple example:

```
$
2 MEGET        ;fetch PCA main menu year
1 MEGET        ;month
0 MEGET        ;and date

Day: ##  Month: ##  Year: ####
$
```

If the menu date is 23th sep 1944, this paragraph would display:

```
Day: 23  Month:  9  Year: 1944
```

The size of the templates must be big enough to hold all the digits possible. If there are less digits, blanks will be inserted (september is displayed blank-9),

If there are more digits than the template can hold, you will only see the last ones. So if you prefer having 1-digit months displayed with a leading zero, you could add 100 (1 MEGET 100 ADD) and keep the template to ##, which will cut of the leading ”1”.

We will repeat the same example with the string template used for the month, so that month will appear by name, not by number:

```
$
1 CARY

,jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
$
2 MEGET          ;fetch PCA main menu year
1 MEGET          ;month
0 MEGET          ;and date

Day: ##  Month: @@@  Year: ####
$
```

The first paragraph defines a numbered row (ARRAY) of text strings, so that we can use the usual month numbers. The comma in the start means, that there is no month number zero.

You may place the three templates in the above example close together:

```
Date: ##@@@####
```

producing something like `23sep1944`.

Of course you cannot do the same with the numbered month version unless you insert spaces, slashes or dots between the three templates

EXAMPLE - EPHEMERIS GENERATOR

(A complete example of printing planetary positions)

The following example is the ephemeris generator included in the Argus distribution, here with comments:

The first line defines the month abbreviations. Signs and planets are predefined strings in Argus. But we will also need the weekay names to make the output more readable.

The PTMP code makes the module tidy up when finished. It manipulates the input data on the go, and without the PTMP code, you will find the input date changed showing the last date in the printout. You would probably prefer it left as it was.

```
$
PTMP
2 CARY

ar,ta,ge,cn,le,vi,li,sc,sg,ca,aq,ps
Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Okt,Nov,Dec
$
```

The next codes enter time into the menu, so that you will get the positions for Noon. The zone in the Argus module is left as is, so you can print an ephemeris for noon for any country. Here the lines setting zone to zero will force the output to GMT noon.

```
$
12 4 MEPUT     ;hours=12
0 5 MEPUT      ;minutes=0
0 6 MEPUT      ;seconds=0
0 8 MEPUT      ;zone hours =0
0 9 MEPUT      ;zone minutes =0

------------EPHEMERIS of planetary positions--------------
```

The counting loop below counts only one month. You could set it to 12 months as shown in the second line which here is “commented out” i.e. the leading semicolon makes it a non-functional comment.

```
$
1 MEGET
1 MEGET
FOR            ;simple one-month version
;12 ADD FOR    ;12 months variant
2 MEGET        ;year
1 CNT          ;month
NUMS           ;display

@@@ ####   Su  Mo Mer Ven Mar Jup Sat Ura Nep Plu
$
80 TABLN
```

```
NUMS            ;80 wide horizontal line


$
1 PAD           ;make string templates widths fixed as given
1 31 FOR        ;day-in-month counter (inner counter)
1 CNT 1 MEPUT   ;set the input month (outer counter)
2 CNT 0 STO
0 MEPUT         ;set the input date after storing it
DNORM           ;this will handle month overflow:
```

The DNORM is nice, so you can avoid invalid dates, for example apr 31th or feb 30th. In such cases the month is incremented and the day changed accordingly, e.g. april 31th to May 1st

If you would like the ephemeris to stop at the last day of month instead of always printing 31 days, you would need to check if DNORM changed the month with something like:

```
1 MEGET DNORM 1 MEGET <> XIF
```

to exit the day counting loop in case of new month.

```
$ Get day of week

JD              ;Julian Day 10000ths ones 7 and 4digit fraction
DIVR DEC        ;modulus of 10000ths
4 MUL
XY ADD
XY 4999 < ADD ;if fraction >0.5 add one
1 ADD 7 MOD
1 STO           ;final weekday result
13 ADD          ;start of weekday names table
0 MEGET         ;day of month
NUMS            ;display with a continuation line mark (\)

##@@           \
$

1 11 FOR        ;innermost loop to count planets Sun-Node

3 CNT
12 MOD PLA      ;calculate planet from current input values
XY DEC          ;remove velocity from result
ITOM            ;convert intang position to minutes of arc
1800 DIVR       ;divide into sign (0-11) and remaining minutes
1 ADD SNAME     ;change sign to (1-12) and get sign name index
XY 60 DIVR      ;divide remainder into degrees and minutes
```

```
XY 100 ADD XY ;Trick to display single digit with leading zero
NUMS          ;display degrees minutes and sign

##\
##@  \
$
NEXT          ;next planet
1             ;terminate with a linefeed (blank line)


$
1 RCL 0 = IF  ;if sunday, output as horizontal line
80 TABLN


$
ENDIF
NEXT          ;next day
1
80 TABLN      ;end of month horizontal line

$
NEXT          ;next month
80 TABLN      ;end of table horizontal line
1


$$$
```

You may change this structure to your personal needs: a Moonphase ephemeris, Hindu positions, local noon positions etc.

STRING MANIPULATION

CARY	define string array
NAME	get current name into string array
PLACE	get current place into string array
STDEF	define one string in string array
STCAT	concatenate strings
STCMP	compare strings
STCUT	cut slice of string
STLEN	return string length
STPOS	find position of substrin
STCHR	manipulate single character in strin

The XLI interpreter has a couple of string handling functions, even if it's not a full blown string handling system. You cannot write your own wordprocessor using XLI!

Strings are numbered from zero upwards. You may create many thousands of strings only limited by computer memory.

To define strings, you may use CARY to create a whole list from zero and up, or you may use STDEF to create a single numbered string.

The string(s) are entered in the text part of a paragraph. You write `1 CARY` if you have one line of string definitions, `2 CARY` if you have 2 lines etc. You decide for yourself how many lines you wish to use. You could just as well have defined the months like this:

```
$
5 CARY

,jan,feb
mar,apr,may
jun,jul,aug
sep,oct,nov
dec

$
```

The above defines strings number `0-12`. String zero ist just empty. All earlier defined strings are erased by `CARY`.

If you wish to define or redefine for example string `12`, use the code `STDEF` istead of `CARY`:

```
$
12 STDEF

dez
$
```

This will redefine string `12` from `dec` to `dez`.

The `CARY` code is used in many of the utility modules in Argus..

There are a number of other string handling: you may cut, join or compare strings. You'll find more information in the function reference part.

GRAPHICS

GRON	switch to graphics mode
GROFF	switch back to text mode

GRCOL	set graphics colour
PENUP	lift pen
DRAW	draw line
DFAT	draw fat line
DRSYM	draw defined symbol

Graphics look pretty, and a lot of modern software puts its heavy sales arguments on graphic features.

To understand this chapter, you must know, that output may be produced in two quite different modes: text or graphic.

Text output is using a very limited number of letters, digits and other symbols: the character set, which has 256 members. This allows for compact data and fast screen, printer and file handling. An ordinary book may be stored in text mode in say 400 KB of memory. If it was to be stored in graphics in laser print quality, the 400 KB would not even hold a single page.

The Argus output pages can hold a mixture of text and graphics. Graphics are defined as a frame floating with the text.

The Argus graphics (normally used for just the chartwheel) will normally fill out 60-90 % of the page width depending on the setting in the print-layout preferences.Further it will not fit into a text line, but always use a number of lines by itself. So even if your preferences say 20% of page width for chartwheel, any graphics, you create will not accept text to the left or right of the graph but take up a couple of lines by itself.

Still, you will be able to manually move graphics around, resize them by width and height and move them to fit into a text line or place several graphs horizontally side by side. But that cannot be scripted in XLI.

The most interesting module is the graphic ephemeris. It has a number of other interesting XLI features, so we will go through it in detail:

THE GRAPHEPH MODULE:

```
;XLI Graphic Ephemeris utility module
```

First the month names are set up as strings. All text drawn in graphics mode must be defined as text strings:

```
$
1 CARY

,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
$
```

Saving the menu values have been discussed earlier. To calculate planets, we need only to change the date and time. So only the first 8 positions in the menu are saved. The

memory cells used for this are cell 100-107. Alternatively, you may use the PTMP code in the start, which will also restore the input menu contents.

Writing modules, you should keep a good record of the use of memory cells, so you do not overwrite important information, and so that you can easily find the information, you need.

```
$
0 7 FOR          ;save menu values
1 CNT MEGET
1 CNT 100 ADD STO
NEXT

$
```

Next part inserts time. You could also instead have done a `CALL` with a macro of 212. (mimicing press 2 for time, then 12 for noon and a full stop for data end)

```
  $
  12 4 MEPUT    ;init time to noon
  0 5 MEPUT
  0 6 MEPUT

  $


  ;SET USER CONSTANTS
  ;------------------
```

If you later on wish to change the appearance of the graph, it is a very good idea to have the values saved in memory cells like this instead of putting the values directly into all the different places, where you need them. Here they can be easily found and changed. You could even later expand the module with a menu giving you different choices:

```
  $
  1024 10 STO   ;set frame width (max 1024)
  600  11 STO   ;set frame height (max 974)
  1 12 STO      ;first planet to display (1=Sun)
  11 13 STO     ;last planet to display (11=Node)
  2 15 STO      ;horizontal grid option 0,1 or 2
  1 16 STO      ;vertical grid option 0 or 1

  $
```

Now use `GRON` to change the output to graphics mode. The `3 GRCOL` chooses the colour for the following draw. Colour `3` is the graph foreground chosen in the PCA colour installation menu, which is normally used to print degrees and circles in the chartwheel.

```
  $
  GRON          ;open a graphics fram
  3 GRCOL
```

```
$
```

The next part will draw the outer frame using the stored width and height values. The coordinates are negative (`CHS`) or positive, because the origin is in the middle of the screen or print area.
The `DFAT` code draws not just one line, but a series of parallel lines. `3 2 DFAT` means: draw 3 lines with a mutual distance of 2 dot units. You could also use the code PENW to draw solid thick lines

```
10 RCL CHS 11 RCL CHS 10 RCL CHS 11 RCL 3 2 DFAT
10 RCL CHS 11 RCL 10 RCL 11 RCL 3 2 DFAT
10 RCL 11 RCL 10 RCL 11 RCL CHS 3 2 DFAT
10 RCL 11 RCL CHS 10 RCL CHS 11 RCL CHS 3 2 DFAT

$
```

The next three paragraphs draw the grid. The colours are the ones chosen for angles and houses in the PCA color preferences.

```
$
15 RCL 0 < IF
4 GRCOL        ;draw horz. 5 degree lines
11 RCL 15 DIV
0 STO          ;heigth of one degree
-2 2 FOR       ;5 lines total
10 RCL CHS     ;horizontal start (x1)
1 CNT 5 MUL
0 RCL MUL      ;vertical start (y1)
2 DUP CHS      ;horizontal end (x2=-x1)
2 DUP          ;vertical end (y2=y1)
1 2 DFAT       ;draw one line
NEXT           ;repeat for all 5 lines
ENDIF

$
15 RCL 1 < IF
5 GRCOL        ;draw horz. 1 degree lines
-14 14 FOR     ;29 lines total
1 CNT ABS
5 MOD 0 =      ;is it a 5-degree line
NOT IF         ;draw if not
10 RCL CHS     ;horizontal start (x1)
1 CNT
0 RCL MUL      ;vertical start (y1)
2 DUP CHS      ;horizontal end (x2=-x1)
2 DUP          ;vertical end (y2=y1)
1 2 DFAT       ;draw
ENDIF
```

```
NEXT
ENDIF

$
16 RCL
0 < IF
4 GRCOL        ;draw vert. month lines
10 RCL 6 DIV
0 STO          ;width of one month
-5 5 FOR       ;11 lines total
1 CNT
0 RCL MUL      ;horizontal start (x1)
11 RCL CHS     ;vertical start (y1)
2 DUP          ;horizontal end (x2=x1)
2 DUP CHS      ;vertical end (y2=-y1)
1 2 DFAT       ;draw
NEXT
ENDIF

$
```

Now the names of the months defined earlier with the `CARY` code are drawn. To do this, the code `DRTXT` is used.

```
$
1 12 FOR       ;Draw month names
1 CNT          ;string number
0              ;rotation=0=vertical
50             ;size
1 CNT 7 SUB    ;x-position -6 to 5
0 RCL MUL      ;use stored month line scaling
0 RCL 4 DIV
ADD            ;move a little right
11 RCL 50 ADD ;place text 50 units above frame
DRTXT          ;draw month name
NEXT

$
```

Next comes the planet movement curves. Included are the 10 planets and the Moon's node, if you did not change the constants in cell 12 and 13.

```
; DRAW PLANETARY MOVEMENTS
;-------------------------
$
10 RCL 30 DIV
0 STO          ;x-step=width/30
12 RCL 13 RCL
```

```
FOR             ;count planets
1 CNT 2 =
NOT IF          ;exclude Moon (too fast)

$
```

This sets the colour of the planet as defined in the colour installation menu. Planet colours start at no. `17`.

```
$
1 CNT 17 ADD
GRCOL           ;set planet colour

$
```

Counting the year goes in steps of two days. The counter moves from 0 to 180 in steps of one, so this must be multiplied by two and have add one to get the date. This result is entered into the input menu using `MEPUT` even if you may have a silly date like 302nd of january, the `DNORM` function will convert this to the correct day and month.

```
$
0 180 FOR       ;date loop, 2 days step
102 RCL
2 MEPUT         ;get year from saved menu values
2 CNT 2 MUL     ;get date step
1 ADD           ;add to the 1st of january
0 MEPUT         ;insert day in menu
1 1 MEPUT       ;insert month=1
DNORM           ;normalize the date

$
```

The `ZZO` code used below is not really necessary here, but shown as an example. If you have a lot of calculations going, it may be difficult to remember how deep down the result stack, a given value resides. `ZZO` puts a marker letting you use `GET` and `PUT` to access results placed AFTER the `ZZO` code. `0 GET` will get the first result, `1 GET` the second etc.

```
$
ZZO             ;set origin in result stack

$
```

The PLA code calculates single planets using the menu date and time. It calculates both longitude and speed, but here you need only the longitude. To remove the speed, the two results are swapped, and the DEC code cancels the (now topmost) speed. A bit of arithmetics converts the longitude from intang to minutes and sign.

```
$
1 CNT PLA       ;calculate planet using menu values
```

```
XY DEC         ;get longitude (discard speed)
ITOM           ;convert to minutes of arc
1800 DIVR      ;convert to sign & minutes

$
```

Now the result stack holds: SIGN MINUTES
The graph superimposes all the signs (30 degree system), so to draw the movement, only the date (x-value) and position in sign (minutes) are used. The sign number itself is used only to check if the planet moves past a sign limit.

```
$
2 CNT 90 SUB   ;calculate horizontal position
0 RCL 3 MULT   ;horizontal position
4 STO          ;save x
0 GET 900 SUB  ;vertical position=minutes of arc
11 RCL
900 MULT       ;scaled  to +-frame heigth (cell 11)
5 STO          ;save y
1 GET 6 STO    ;save sign

$
```

Each time a position is calculated it is saved in cell 4 (x), cell 5 (y) and cell 6 (sign). After the line drawing, these values are moved to cell 1-3 as "old" values. For each step you will then be able to draw a line from the "old" x,y to the "new" x,y. The only exception is the first step, where no old values yet exist. Therefore the first line below excludes drawing in that case.
Also excluded are sign shifts, which must be drawn differently.

```
$
2 CNT IF       ;if not first position
6 RCL
3 RCL = IF     ;if same sign as last position
4 RCL 5 RCL    ;start x,y
1 RCL 2 RCL    ;end x,y
1 2 DFAT       ;draw line of movement
ENDIF

$
```

If sign shift is detected, the line of planetary movement must be split in two: The first part, which moves from "old" x,y to the sign limit (the grid frame), and the second part moving from the other sign limit to the "new" x,y. The calculation is split in two parts: One direct and one retrograde.

The calculation needs some insight into triangle geometry. A further complication is, that using integer arithmetic, it is necessary to use the `MULT` code to be able to scale the

values properly. MULT combines a multiplication and a division avoiding round-off and overflow errors.

```
$
6 RCL
3 RCL SUB      ;if sign changed to next
12 MOD 1 = IF ;calculate when
5 RCL 2 RCL
SUB            ;y difference (minutes only)
11 RCL
2 MUL ADD
7 STO          ;real y diffence (including sign)
11 RCL
2 RCL SUB      ;remaining y movement in old sign
1 DUP          ;use twice
4 RCL
7 RCL MULT     ;scale new x proportionally
XY             ;remaining y movement again
1 RCL
7 RCL MULT
SUB            ;scale old x and subtract
1 RCL ADD      ;add to old x
7 STO          ;crossing 0 degrees at this x
1 RCL 2 RCL
7 RCL 11 RCL
1 2 DFAT       ;first part drawn
7 RCL 11 RCL
CHS
4 RCL 5 RCL
1 2 DFAT       ;second part drawn
ENDIF

$
6 RCL 3 RCL
SUB            ;if sign changed to prev
12 MOD
11 = IF        ;calculate crossing point
5 RCL
2 RCL SUB      ;y difference (minutes only)
11 RCL 2 MUL
SUB 7 STO      ;real y diffence (including sign)
11 RCL CHS
2 RCL SUB      ;remaining movement in old sign
1 DUP          ;use twice
4 RCL
7 RCL MULT     ;scale new x proportionally
XY                              ;remaining y again
1 RCL
```

```
7 RCL MULT
SUB           ;scale old x and subtract
1 RCL ADD     ;add to old x
7 STO         ;crossing at this x
1 RCL 2 RCL
7 RCL 11 RCL
CHS
1 2 DFAT      ;first part drawn
7 RCL 11 RCL
4 RCL 5 RCL
1 2 DFAT      ;second part drawn
ENDIF

$
```

Now after drawing a line bit, the x,y and sign information is moved from "new" (cell 4,5,6) to "old" (cell 1,2,3), and the daycount and planet count loops continue. Note the `ENDIF` placement between the two `NEXT`'s. You must always keep these constructions either entirely overlapping or not at all. Partial overlap e.g. `FOR IF NEXT ENDIF` will produce chaos, and should be avoided.

```
$
ENDIF
4 RCL 1 STO   ;keep last x
5 RCL 2 STO   ;and y
6 RCL 3 STO   ;and sign
NEXT
ENDIF
NEXT

$
```

Finally, the saved menu values are put back to the input menu, so that it will look the same as before the call, provided, that you did not interrupt the printout pressing ESC.

```
$
0 7 FOR       ;restore menu values
1 CNT
100 ADD RCL
1 CNT MEPUT
NEXT
WKEY
GROFF

$$$
```

This concludes the GRAPHEPH module.

GRAPHIC EXAMPLE - DRAWING A CHARTWHEEL

This example can be used as a template for drawing a chartwheel to your own specification. It also presents a couple of graphic techniques, e.g. pushing and popping graphic states and use of rotating and scaling. You'll find rich commenting in between which should help your understanding. The coding itself is actually only 66 lines.

```
$
1000 1400 SCALE GRON 0
```

All graphics output must start by switching graphics on with code GRON and terminated by switching it off with code GROFF. The code between GRON and GROFF defines the graphics output.

Graphics appear within a frame with the default coordinates +-1024 both for x and y. With SCALE, you can define a different plot area. As the only graphics command, the SCALE code must appear before GRON. So default is 2048 2048 SCALE.

When displayed in the output window, the width of the graph is determined by the preferences "wheel size". Using a different plotarea width just means, that this width is divided in finer or coarser steps.

But using a non-square plotarea lets you use more or less paper height. So if you wish to make a full page graph, you would define a plotarea with more height than width.

```
$
-1000 -1000 DRAW
1000 -1000 DRAW
1000  1000 DRAW
-1000  1000 DRAW
-1000 -1000 DRAW
PENUP
```

To draw a line, use code x y DRAW. The first DRAW will not draw, just place the pen. Succedent DRAW commands will draw from the last point to the next. Use code PENUP after the last draw. The first draw after PENUP will be a non-drawing move.

```
$
GPUSH
0 11 FOR
1 CNT 30 MUL 8192 45 MULT ROTAT
```

```
  0 970 DRAW
  0 1000 DRAW
  PENUP
  NEXT
  GPOP
```

It is essential in a graphics system to be able to transform graphics, meaning to displace, rotate and scale graphics elements. For this XLI has the codes ORGIN ROTAT and RSIZE. When using temporary rotates, scales etc. you will need to get back to the original coordinate system afterwards without having to reverse each displace etc.

The commands GPUSH and GPOP will do exactly this. Put a GPUSH code before your transformation codes and a GPOP after drawing the transformed elements.

As an example we will draw 12 piechart-type division lines from the centre and 1000 plot units in the directions 30, 60, 90 120... degrees.

v ROTAT will rotate the coordinate systm, so that the following graphics will be rotated v intang units around the current origin (centre). Intang units are 65536 units equalling 360 degrees. To convert degrees to intang multiply with 8192/45.

```
  $
  32768 34 PPOS SUB ROTAT
```

To have the ascendant appear horizontal, the whole drawing must be rotated accordingly. If the ascendant is in 0 Aries, the chart will need to be rotated 180 degrees to have the ascendant point left, as the normal coordinate system starts its angles from the X-axis which points to the right.

If the ascendant is in 30 degrees (0 Taurus), the drawing should be rotated only 150 degrees. So the general rule is, that the rotation is calculated as 180°-ascendant or in terms of intang values: 32768 14 PPOS SUB.

```
  $
  800 200 200 0
  DRSGN          ;radius width height rotation DRSGN
```

To draw the zodiac, you can use codes for drawing single symbols rotating them on the fly as above. But

it is easier to use the DRSGN code, which will draw
all twelve signs rotated correctly and in the colors
defined in the Argus preferences. The parameters for
DRSIGN are:

Radius to the centre of the sign glyps.
width of each glyph
height of each glyph
rotation: angle of zero Aries in intang

```
$
900 5 2  -100 10 -45 7 -30 5 -18 0  DGREE
700 5 2  100 10 45 7 30 5 18 0  DGREE
500 5 2  -100 10 -45 7 -30 5 -18 0  DGREE
```

To draw a degree scale, the DGREE code provides a lot
of control. The arguments are:

radius: radius of the base circle
wcirc: linewidth of base circle
wdcirc: linewidth of extra circle (set to 0 for no extra circle)
l30 :length of 30 degrees divisions
w30 :width of 30 degree divisions
l10 :length of 10 degree divisions
w10 :width of 10 degree divisions
l5 :length of 5 degree divisions
w5 :width of 5 degree divisions
l1 :length of 1 degree divisions
w1 :width of 1 degree divisions

Please note, that all parameters other than radius are
measured in promilles of radius, so that you can easily
scale up and down just by changing the radius and keeping
the other parameters.

Note also, that you can use negative values for the lengths
of the degree markings, which will make them turn inwards.

```
$
1004 PENC
15 PENW
1 1 1 -900 1000 100 DRHOU
1  10 10 -900 1000 100 DRHOU
1005 PENC
5 PENW
1  2  3 100 700 50 DRHOU
1  5  6 100 700 50 DRHOU
1  8  9 100 700 50 DRHOU
1 11 12 100 700 50 DRHOU
```

To draw houses, use the DRHOU code. The parameters are:

chart : as for planets
from house number
To house number
radius of start of house division line
radius of end of house division line
length of arrow

It is possible to use a negative radius for example
to make the house line pass through the chart centre.

In the above example, there are codes for pen width
(PENW) and pen color (PENC). The color numbers can either
be the 000-888 RGB system used by Argus, or color numbers
1000-1037 used in the Argus color preferences. Just add
1000 to the number shown in the preferences color table.

```
$
1  1  1  10  500  3  DRASP
```

The aspect drawing DRASP has the following parameters:
chart :as with planets
spread :1=endpoints at spreaded positions 0=true positions
first planet
last planet
radius for endpoints
linewidth

```
$
1  1  10  10  SPRCH
1  1  10  10  500  600  700  DRPLA
```

To draw the planet positions, use the DRPLA code. To avoid
overlapping, they should be spread first using the SPRCH code.

The parameters for the SPRCH are

chart	0=radix 1=current 2=radix 3=auxchart1 4=auxchart2 5=present 6=auxchart3
lo planet	lowest planet number
hi planet	highest planet number
spreadsize	spread value, should equal planet size (see DRPLA)

The parameters for DRPLA are:

chart	:same as for spread
lo planet	:lowest planet number
hi planet	:highest planet number

planet size ;degrees as seen from chart centre
rinner ;radius of inner circle to where positions marks point
rplanet ;radius where planets should appear
router ;radiuo of outer circle to where position marks point

please note: rinner and router should normally be equal to the radius of a degree scale. If you need only one set of position markers, make rinner =router.

Even if you may find it tempting to draw planets first and aspects and houses later, there is a reason for doing it in this order. The planet drawing makes an invisible (background colored) dropshadow, to make them stand out more clearly, if drawn on top of other graphic elements.

```
$
1 1 10 10 SPRCH
1 1 10 10 500 600 700 DRPLA

$
2 1 10 4 SPRCH
2 1 10 4 900 950 900 DRPLA
1 2 0 1 10 1 10 900 700 1 DRXAS
```

To show how to draw a second set of aspects for the current chart (until now it has been radix), the same SPRC and DRPLA codes are used with slightly different parameter values.

To draw the cross-aspects between the two, a second aspectdrawing code is used: DRXAS which has the following parameters:

chart0 first chart number
chart1 second chart number
spread 1=aspect line ends at spreaded position, 0=true position
lopla0 first planet in chart 0
hipla0 last planet in chart 0
lopla1 first planet in chart 1
hipla1 last planet in chart 1
r0 radius for aspect endpoints chart0
r1 radius for aspect endpoints chart1
linewidth use values between 0 and 4 to get reasonable results

```
$
GROFF

$$$
```

This concludes the Chartwheel example

CHANGING PCA CONFIGURATION

PCA has a number of installable settings. You can change these using the systemvariables edit facility in the preferences menu. However they may also be changed using the SYSTR function.

Further there are some special functions to change system settings.

PTMP: Placing this code in the start of your module will assure, that your changes will be temporary during the module run and reset when the module finishes. Without PTMP the values will change for the rest of your Argus session.

CONFX: There are a couple of these settings available, which decide a number of program options. Some of them are values others are flags, which can be only 1 (true) or 0 (false). A full list of the system variables including the CONFX values are given at the end of this document. Many CONFX values are unused for now.

SYMIX is used by the SYMBOL to change the glyphs for Uranus and Pluto

AYA sets the ayanamsha used by sidereal astrologers. Ayanamsha is a zodiac origin offset, meaning, that the sign Aries starts somewhere displaced from the vernal equinox. The setting refers to the displacement at 1.1 1900 A.D. Argus will then adjust the offset to the current time calculating the precession. Precession happens, because the stars and constellations moves relative to the vernal equinox by approx 50" of arc per year.

COL customizes the colours and can actually do no more than you can do using the colour installation menu. It is the only way though, that you can change the colours from XLI, because you cannot create a macro manipulating the colour installation, which uses cursor keys mostly.

Further information on these codes are given in the function reference.

RUNNING ANOTHER PROGRAM FROM WITHIN PCA

EXEC execute external program

AUTO.XLI If exist, AUTO.XLI will be executed automatically, when Argus is loaded.

BROWS will open a browser window in your default internet browser

Calling external programs without having to stop PCA means, that on return, you will find PCA in the state you left it with the same data and jobs, you were running before the call. For example, if you just need to lookup something on your harddisk, edit an XLI file for testing or send a fax. This however, could be easier done, if you have a multitasking or taskswitching operating system.

More interesting is, that positions, aspects or whatever calculated by PCA may be transferred to the program you call, and values calculated by the external program can be transferred back to PCA.

The AUTO.XLI is not a code but an XLI file that, if present, will execute as soon as PCA starts without showing the menu first. The AUTO.XLI may be coded also to quit PCA automatically when done, by putting a macro at the end consisting of a Q (quit). Ultimately, any number of astrology programs (with similar facilities), databases, statistics programs, communication programs sending and receiving letters and faxes may cooperate in one big session.

EXECUTING an external program

You may execute any .EXE program or .BAT from an XLI file. Here are some examples of using EXEC:

```
$
EXEC

minityd.txt
$$$
```

The coding field should just have the code EXEC. The first line of the text field must hold the complete path and program name including .EXE.

You could also put a name of a file having an extension with a default application to run it in Windows. For example try replace the minityd.txt with argus.bmp. This should open some graphics viewer, for example Windows Photo Viewer showing the argus.bmp image.

If you want to call Notepad.exe you probably will get nothing, if you just write notepad.exe, as Argus will not be able to find it. Still as a CMD window will know where to find it, you could write a BAT file called notepad.bat and put it in the Argus folder, then pu notepad.bat under the EXEC code. The notepad.bat file should be just a one-line textfile of "notepad.exe".

You could put arguments after the program name, for example.

$

EXEC

PCA42.EXE 1T.XL

```
$
```

This will open another instance of Argus printing a one month collective transits graph for the current month. 1T.XL is a macro setting current time and calling the graph.

You may find a lot of things possible using EXEC. Feel free to experiment.

MISCELLANEOUS TIPS AND TRICKS

GRAF	display bar graph
INFIL	read 1 line of data file

Printing Bar graphs

You may print horizontal bar graphs to visualise the size of numbers. This works in simple text mode using the IBM-OEM characters, for example a block character (220). An example of this is given in the ELGRAF.XLI which follows here:

```
;ELGRAF printing bar graphs for tripli- and quadruplicities
;1 STO  ;Fire
;2 STO  ;earth
;3 STO  ;air
;4 STO  ;water
;5 STO
;6 STO
;7 STO

$
PTMP                                ;set module to clean up when finished
NPR 1                          ;Print name

Number of planets including ASC in
$
1 OEM                               ;use DOS OEM character set

0   1   2   3   4   5   6   7   8   9  10  11
--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM
0 1 7 FOR 1 CNT STO NEXT                    ;Rest STO vars
1 14 FOR                                ;Count planets
1 CNT 11 > 1 CNT 0 14 IN OR IF          ;and ASC
1 CNT PSI                               ;Signs
```

```
1 SUB 4 MOD 1 ADD                       ;Elements
1 DUP RCL 1 ADD XY STO
1 CNT PSI
1 SUB 3 MOD 5 ADD                       ;Quadruplities
1 DUP RCL 1 ADD XY STO
ENDIF
NEXT
7 1 FOR 1 CNT RCL 4 MUL NEXT 182 GRAF

Element Fire                #
Element Earth                #
Element Air                 #
Element Water                #
$
1 OEM

--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM 182 GRAF

Cardinal signs              #
Fixed signs                 #
Movable signs                #
$
1 OEM

--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM

$$$
```

File input/output

File output for example for data export is done setting up a print file. You can then run one or any number of paragraphs with text or with inserted templates to print out number and strings. Putting a backslash at the end of a line allows you to put bits of lines together to longer lines.

To get input from a file, you use the code `INFIL`. `INFIL` reads a line of a textfile. It is possible just to read one line as is, or you may read special bits, numbers or strings. A detailed explanation is given in the function reference part.

The following example will read a line of the file XXFILE.TXT into string `0`. It could then be printed, inserted as a name in the namefile, compared to other strings etc.

```
$
1 INFIL

XXFILE.TXT
@
```

```
$
```

To print a complete textfile on screen , the following code will do the job:

```
$
3 PAD                   ;output expand mode (single @ template)
1 32000 FOR             ;setup maxnumber of lines
1 INFIL                 ;read one line from file to string 0
-1 = XIF                ;break at end of file

XXFILE.TXT
@
$
0 NUMS                  ;output string 0 to screen

@
$
NEXT

$$$
```

To extract 6 numbers from an input line:

```
$
1 INFIL                 ;read one one line and parse numbers

XXFILE.TXT
######
$
```

The `@` and `#` templates can be mixed with search characters to setup quite complicated input tasks.

For example in the NAME2TXT module the following `INFIL` is used:

```
$
1 INFIL

NAMEFILE.CSV
"@"###, @1,#####, @2,##, @3,##, @4, @5,
$
```

The module will read a line like this:

```
"Electric Ephemeris  ",12,12,1925,AD, 8,39, 1, 1, 0,E,55,42,N, 12,35,E,M  :
```

The above input spec string means:

`"` look for the first quote character

`@"` read the following characters until but not including the next quote character into string 0

`###` read three numbers separated by any character(s)

`,` look for next comma

read any number of (leading) spaces (if any)

`@1,` read the following characters until but not including the next comma into string 1

`#####` read five numbers separated by any character(s)

`,` look for next comma

read any number of (leading) spaces (if any)

`@2,` read the following characters until but not including the next comma into string 2

`##` read two numbers separated by any character(s)

`,` look for next comma

read any number of (leading) spaces (if any)

`@3,` read the following characters until but not including the next comma into string 3

`##` read two numbers separated by any character(s)

`,` look for next comma

read any number of (leading) spaces (if any)

`@4,` read the following characters until but not including the next comma into string 4

read any number of (leading) spaces (if any)

`@5,` read the following characters until but not including the next comma into string 5

`,` look for final comma

You may find a number of other uses and combinations if you go through the function reference on INFIL.

There is a similar code called UTFIL for writing a text file. See the function reference.

LOW LEVEL ASPECT CALCULATION (AZP)

The code `AZP` is a primitive aspect routine which needs more programming from your side. The advantage is, that it is inde-pendant of the PCA orb settings and that it is completely flexible. Below is an example of its use:

```
2 PPOS 8 PPOS SUB AZP 4 = XY ITOMS ABS 60 > AND     ;Moo Tri Ura
```

Here follows a breakdown of the above coding:

```
2 PPOS 8 PPOS SUB  ;calculate Moon-Uranus angle (intang)
AZP                ;calculate aspect number and orb
4 =                ;check that it is a trine
XY                 ;fetch the orb (intang)
ITOMS              ;convert intang to minutes of arc
ABS                ;remove orb sign
60 >               ;check within one degree
AND                ;combine aspect number and orb checks
```

The `XY` fetches the orb one level down the stack, and leaves the first condition (the trine test) just below. When the orb testing is done (`60 >`), you will have the result of the orb test topmost, and the trine test below. The `AND` will combine the two and the result will be `1` (true) if both the two conditions were met.

The `AZP` function produces two results, the aspect number (placed topmost on the stack) and the actual orb (next). The actual orb is signed, so you may care about the order in which you subtract the two planets, so you can use the orb sign to find out if the aspect is applying or separating (if you know which one is the fastest).

A more advanced coding example is given here to calculate the orb speed. Here the actual planet speeds are used, so you do not need to care about which planet is the fastest, and whether they are retrograde etc.:

```
$
3 PPOS 4 PPOS SUB AZP       ;aspect no. orb
XY 1 DUP 12 MUL             ;12 * orb
3 PV 4 PV SUB               ;orbspeed (signed)
1 DUP NOT IF                ;check div by zero
DEC DEC 999 1 ENDIF         ;if so replace by 99
DIV                         ;orbspeed (months)
XY ITOMS                    ;orb (signed minutes)
NUMS                        ;print results

Venus aspect Mercury

orb:    ######
```

```
speed:  ######
aspect: ######
$$$
```

The orb will be a signed value. The sign does not tell if the aspect is applying or separating, but if the angle is less than or bigger than the precise aspect.

The orbspeed will be a number of months, if the chart is a progressed one. That is the reason for multiplying by `12` in line 2. This number will also have a sign which in fact tells if the aspect is applying or separating. So -7 means, that in a progressed chart, the aspect was exact 7 months ago. This calculation assumes, that the planets are moving at constant speed. Planets around their station will not fit this ideal case. To get the exact time of perfecting the aspect, you will need to do repeated planetary calculations to adjust for speed nonlinearity.

PLEASE NOTE Argus has an improved version of AZP called AZE. Please refer to the function reference

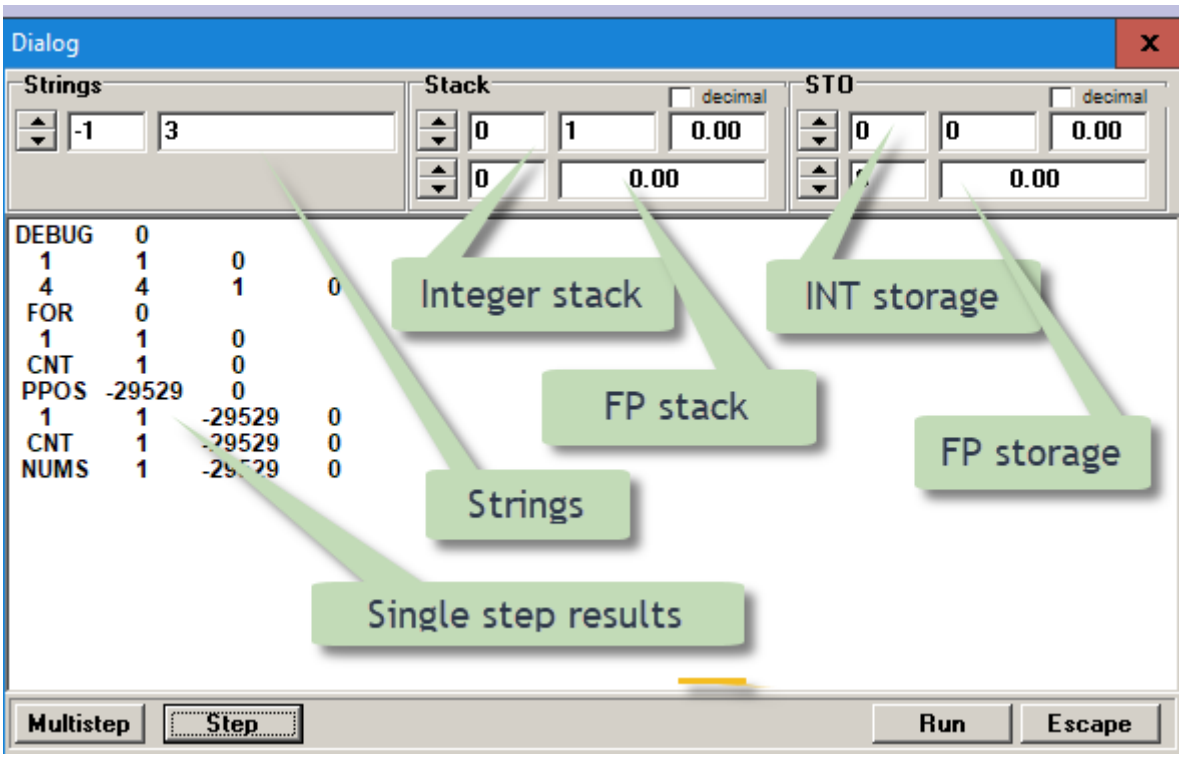
THE XLI DEBUGGER

Debugging programs may be a very hard job. You think, that you have analyzed your problem well, coded it correctly, and still even simple tasks may seem to behave differently from the expected. Your first thought may be: faulty toolkit, faulty PCA, faulty computer/harddisk, virus attack or whatever. The point is, that you cannot exclude any explanation until you really find out, what's actually going on. Very often, when you finally find the reason, what you first thought impossible, suddenly will appear obvious.

To find out what's going on, you need a magnifying glass to follow the action of your code step by step. This is what the debugger does. The execution of XLI modules may run in two modes: normal mode and debug mode.

In debug mode, you'll have to click the STEP button for each code, so it will run very slowly, even if you hold the MULTISTEP button down instead for repeat. But the advantage is, that for each code executed, you will see the topmost numbers on the result stack. Values already on the stack when the DEBUG code starts the debutter will not be shown, only the topmost.

The debug window



Activating the debugger:

The debugger (or debug mode) may be activated at any point in your module. Just put the code `DEBUG`, where you want the debugging to start. When the debugger has started, you may single step clicking the STEP button. Holding down the MULTISTEP will execute several steps in repeat-mode. To leave the debugger, click the RUN button. Finally you may quit the module prematurely clicking the ESCAPE button.

If you put `DEBUG` inside a loop, clicking RUN will leave debug mode only until it loops back to `DEBUG` next time. You may then by repeatedly clicking RUN follow the result at just one point in the loop, not having to tediously step through all the single codes inside the loop.

If you put `DEBUG` inside an `IF-ENDIF` construction, it will only be activated if the `IF` statement is true. So if you have complex code with a large loop, for example 0 600 FOR, and you want to debug an see what happens, when you reach count 472, you can insert a line coded

```
1 CNT 472 = IF DEBUG ENDIF
```

The loop will then run until count 472, then start the debugger. Of course, you can insert any condition, starting the debugger at e.g. a certain value gets negative or exceeds some limit.

At the top panel of the debugger, to the left you see a panel showing the values of the string array. You may step up and down or enter the desired index.

The top panel middle shows all the stack values. you may step up/down or enter an offset, so you can also see discarded values. The "decimal" field shows the angle in degrees if the stack value is an intang. Intang values are not exactly user friendly. If you check the decimal box, the digits after the point are true decimals, if unchecked, they ar minutes of arc.

The bottom field shows the FP stack-orientated

Top right, is the STO cells and the FP STO cells (NSTO~).

In practice:

You may experience a number of problems writing XLI modules or interpretations. The debugger does not solve all of these, but will enlighten most coding errors.

Find the paragraph, which produces false output, or which fails to produce the expected output, insert the code `DEBUG` as the first code in the paragraph and run the module. When the program goes into debug mode, click STEP a number of times letting the debugger single step. Check the intermediate results and compare them to what you expected.

If your module consist of a series of files calling the next, and it somwhat seems not doing its job, it may be due to a link fail (check the next-filename) at the end of each module, or that some paragraph has a blank line after the $-line, or that you have an unfinished IF ENDIF construction. Debugging is a creative job!

There is also a code called XLOG which instead of letting your single step your module writes the whole lot to disk. Please note, that this may slow down program execution to the degree, that you may think, the program crashed..

ENCRYPTING YOUR TEXTS AND MODULES

Interpretations and modules for PCA are quite vulnerable to piracy copying. Real programs (EXE and COM files) may be copy protected, but interpreted textfiles like XLI-files cannot. Even if the main PCA program is not copyprotected either, it has the user name unerasable inserted.

To add a minimum of safety from illegal copying of your modules, you may use the program SCRAM to encrypt your interpretations and modules, and at the same time prevent, that they are used with more than one users program. The encryption uses the PCA user number as encryptation key.

To encrypt a module, use the following command-line command:

```
SCRAM filename user number (RETURN)

for example SCRAM TT1.TXT 13260
```

Of course TT1.TXT must be present. SCRAM will produce a new file TT1.SCM. This will run with PCA, but the .SCM extension means, that PCA will decrypt it checking it against its user number. If the user number does not match, only rubbish will be output.

If your interpretation or module consist of several files, you should encrypt all of them. You

You do not need to change the file extensions to .SCM, neither in the installation menu or in the file chains themselves. If PCA cannot find a file, it will automatically try to look for a file with the same name, but with the extension .SCM. As soon as an SCM file is found, PCA automatically assumes, that the remainder of the file chain is .SCM files.

The SCRAM program is available for developers on request from Electric Ephemeris.

COMPLETE XLI FUNCTION REFERENCE

The following complete alphabetical function code reference explains the details of each function.

For each function a stack specification is given. The topmost figures taken off the stack are called X,Y,Z,P,Q,R etc.

For example:

```
X Y DIVR → X mod Y  X/Y
```

means that X and Y are removed from the stack and that first X mod Y, then X/Y are put back on the stack. So after the operation X/Y will be on top of the stack.

```
X IF →
```

means that X is removed, and nothing is put back.

```
X STO → X
```

means that X is used, but remains unchanged on the stack.

```
ENDIF →
```

means that the code does nothing with the stack.

Integer and Floating point stack

Argus actually handles two stacks, one for integer numbers and a second one for floating point (decimal) numbers. FP numbers provide a higher accuracy.
For instance when you input time and date, the menu data is held as 7 integer numbers: day, month, year, BC/AD flag, hours, minutes and seconds. There is two FP codes: GETTI and SETTI, which takes all seven numbers and converts them to just one FP value or back. A single GMT date/time value is often convenient when you need to add or subract time spans.

Also planetary positions held in integer, even as intang values have a precision of approx 20 seconds of arc, which is normally okay, but having the Swiss Ephemeris, you can have much higher accuracy, which is provided with a couple of FP XLI codes.
Floating point XLI functions are marked with FP, so you can know, that they operate the FP stack.

XLI MACROS

A great way to run or test small XLI samples is embedding them in a macro.

To do this, create a macro starting with a $-sign.Now you can insert XLI codes in much the same way as if it was a file. The difference is, that you have only one line. You may still have more than one paragraph, just separate them with additional $-signs.
To add a text part to a paragraph put each line in parenthesis:
The text line may contain templates, so you can output values, for example results of your coding.
You can also include a DEBUG code, so you can single step through the coding.
Here is some very basic examples of an XLI macro

```
$  1 (HELLO WORLD)
$1 5 FOR   1 (HELLO WORLD) $NEXT
$1 5 FOR   1 CNT NUMS  (HELLO WORLD no. ##) $NEXT
```

Please note, that even if you enter some lower case letters, the Macro interpreter **always** changes it to uppercase.
The following will print the birth data of number 3 to 10 in the namefile.

```
$3 PAD 3 10 FOR 1 CNT NFI  15 1 BDATA 1 NUMS (@) $ NEXT
```

PART TWO - FUNCTION REFERENCE

Data:	**MEGET**	fetch value from PCA input menu
	MEPUT	insert value into PCA input menu
	NFI	fetch namefile entry no. n
	NPNT	read/write namefilepointer
	NPUT	insert name into PCA input menu
	BDPR	write birth data
	NPR	write name
	DDIF	find distance between to dates
	NDATE	convert transit-timeslice to date
	JD	find julian date
	INFIL	read one line from textfile
	UTFIL	print to file, equivalent to INFIL
	MOVCH	move chart data around
	SWDAT	swaps input data sets
	CNTRY	get country name from current data
	AREA	get timezone area code from current data
	MENAM	insert name in string n
	BDSEL	select input dataset
	AGE~	FP age in years currentdata-radixdata
	BDATA	birth data to string or print
	DNORM	Normalise input menu values to valid date
	GETGL	Get 1 line from atlas
	GETTI	Get date/GMT from input data as FP number
	GETZO	Get zone from input data as FP number
	KM	Distance in km between current and radix position
	MAC*	Overwrite namefile position
	MAC+	Insert current data in namefile
	NFN~	Select file in multiple file-end branch
	NOTE	Get one note line from currently selected namefile entry
	PFILE	show file content in output window
	SETTI	Set date and time input values from time (days)
	SETZO	Set input zone value from zone (days)
	VICI	Output size n list of closest cities
	ZNORM	Get correct zone from zonetable
Astrology:	**PSI**	planet in sign
	PHS	planet in house
	RX	planet retrograde
	HSI	house in sign

HRU	house ruler
APOW	aspect power
ANUM	aspect number
AORB	aspect orb
AZP	aspect test primitive
PDEG	planet position (degrees)
PPOS	planet position (intang units)
HPOS	house position (intang units)
PV	planet speed (intang units)
HV	house speed (intang units)
PLA	calculate one planet (intang units)
XPLA	find auxiliary planet information
HOUSE	calculate one house (intang units)
BHUS	house position (universal)
KUN	fetch the 3 closest Kündig sections
GTR	call userdefined graphic transit
RTA	reset transit aspect list
NTA	fetch next transit aspect
RECH	fetch latest horoskope type
ZODOF	Change zodiac origin to x
AZE	aspect primitive
PNAME	get system string index for planet name
SNAME	get system string index for sign name
REF	switch interpretation reference mode on/off
ANAME	get system string index for aspect name
FPPLA	FP planet/house calculation including velo, lat, ra, decl etc
HITS	Aspectarium find aspects, sign shifts etc.
HSET	Change calculated house position for a chart
NTAX	Get next stored aspect from graphic transit/progression
PSET	Change calculated planet position for a chart
SPRCH	Spread planets of given chart
XPOS	Get extra position info from chart
YPLA	Calculate other planet or body

Mathematics:	**>**	bigger than
	<	less than
	=	equal
	<>	not equal
	IN	group membership test
	AND	bitwise AND
	OR	bitwise OR
	CPL	bitwise complement
	NOT	logical negation

BOO	integer to boolean
ABS	absolute value
CHS	change sign
ADD	add
SUB	subtract
MUL	multiply
MULT	multiply by fraction
DIV	divide
MOD	modulus
DIVR	divide with remainder
MAX	find maksimum of two values
MIN	find minimum of two values
ITOM	intang units to minutes of arc (unsigned)
FPON	turn floating point mode on
FPOFF	turn floating point mode off
FTOI	convert floating point value to integer
XOR	exclusive or
ITOF	convert integer value to floating point
PTR	convert coordinates from polar to rectangular
SPRED	spread conjuncted positions
ITOMS	intang enheder to mintes of arc (signed)
<>~	FP not equal
<~	FP less than
=~	FP equal
>~	FP greater than
ABS~	FP absolute
ACOS	FP arccosinus (degrees)
ADD~	FP addition
AND~	FP logical AND round(a) AND round(b)
ARTOE	intang ARTOE for point on ecliptic
ASIN	FP arcsin (degrees)
ATAN	FP arctan (degrees)
BOO~	FP not zero (or close to 0.001)
CHS~	FP change sign
COS	FP Cosine (degrees)
CPL~	FP complement convert to int and reverse bits
DECL	Convert intang ekliptic position to intang declination
DGREE	Draw circle with 360 degree markings
DIV~	FP division
ETOAR	Intang ecliptic point to AR
FLOOR	FP Floor function
FPMOD	FP modulus
FPPTR	FP polar to rectangular coordinates

	FPRTP	FP rectangular to polar coordinates
	INT~	FP remove decimals
	IN~	Same as IN but using FP stack
	MAX~	FP pick largest of two values
	MIN~	FP pick lowest of two values
	MOD~	FP modulus obsolete version
	MUL~	FP multiply
	NOT~	FP NOT: (1.0 if round(x)=0 else 0.0)
	OR~	FP OR: (1.0 if round(x) or round(y) <>0)
	RE2SP	Rectangular to spherical coordinates
	SIN	FP sine (degrees)
	SP2RE	Spherical to rectangular coordinates
	SUB~	FP subract
	TAN	FP tan
String:	**CARY**	define string variables
	NAME	fetch current name into the streng array
	PLACE	fetch current city into the string array
	STDEF	define one string in the string array
	STCAT	concatenate strings
	STCMP	compare strings
	STCUT	cutout string
	STLEN	find length of string
	NTOS	convert number to string
	STPOS	find substring in string
	ANTOI	convert number field of comma-separated line to
	STCHR	Manipulate single char in string integer
Stack:	**DUP**	duplicate value in stack
	FETCH	fetch value in stack
	ZZO	place origin on stack
	GET	fetch number from stack
	PUT	put number into stack
	XY	exchanges numbers on top of stack
	INC	add one to the stack pointer
	DEC	subtract 1 from stack pointer
	DEC~	FP Remove topmost value from stack
	DUP~	FP duplicate topmost stack item
	FSTKZ	resize FP stack
	INC~	FP stack pointer moved one up (opposite DEC~)
	ISTKZ	Resize integer stack
	SORT~	Sort max 15 FP values
	XY~	FP swap two topmost stack items

Memory:	**STO**	save in memory cell
	RCL	fetch from memory cell
	FSTOZ	resize FP STO array
	ISTOZ	Resize integer STO array
	NRCL~	FP RCL get FP store cell content
	NSTO~	FP STO store value in FP store cell
	RCL~	FP RCL get stored value
	STO~	FP STO store value
Flow-control:	**ENDIF**	end of if-construction
	IF	start of if-construction
	FOR	start of FOR-loop
	CNT	get FOR-loop counter
	NEXT	end of FOR-loop
	XIF	quit FOR-loop conditionally
	NFN	set filebranch index
	WAIT	withhold heading
	CONT	release withheld heading
	PSTAT	change printer status or suppress all print
	EXEC	execute an external program
	DELAY	wait for n milliseconds
	ELSE	conditioned branch
	PROC	define subroutine
	RETN	return from subroutine
	SUBR	call subroutine
	DEBUG	enter debug mode
	BROWS	Open url in default browser
	CLRIF	Clear all IF conditions
	CONT	Continue from WAIT and skip the waiting text
	LOGX	Stop logfile output
	M2XLI	Execute macro sting as XLI code
	MAC<	Select radix data
	MAC=	Fetch currently pointed data from namefile into input menu
	MAC>	Select current data
	MACA	Output aspects
	MACB	Output solar arc chart
	MACD	Output day chart
	MACE	Output Tertiary chart
	MACF	Output Minor progressed chart
	MACG	Output Composite chart
	MACH	Output Relationship chart
	MACL	Output Lunar return chart

	MACP	Output secondary progressed chart
	MACQ	Quit Argus
	MACR	Output radix chart
	MACRG	Call program registration box
	MACS	Output Solar return chart
	MACT	Output transit chart
	MACU	Start the clock chart
	MACV	Draw chartwheel
	MACW	Draw bi-wheel
	MACv	Draw no-house chartwheel
	MACw	Draw no-house bi-wheel
	MOVIE	Start live chart
	XLOG	Start logfile output
Interactive:	**MENU**	create menu
	OPT	wait for defined keypresses
	WKEY	wait any keypress
	MENUX	windows type dialog box
	KEY	simulate a special keypress
Layout:	**FEED**	conditional formfeed (obsolete)
	GRAF	show bar graph
	NUMS	print numbers or strings
	ZMODE	text wordwrap mode
	MONS	set monospaced font mode
	OEM	set OEM font on/off
	FONTS	define fonts
	FONT	select font definition
	XFEED	conditional formfeed (obsolete)
	CFEED	conditional formfeed (obsolete)
	TXCOL	change text color
	TAB	tabulate
	PAD	set string templates padding mode
	CENT	set centered text justification
	DFCOL	Define color from RGB values
	ENCOD	Encode text string between ANSI OEM and UTF-8
	FEED~	FP version of FEED (obsolete)
	SKZ	Skip this paragraph if zero (stack untouched)
	SKZ~	same as SKZ
	TABLN	Print horizontal line
	VBTIM	Dont translate character set in output
Graphics:	**GRON**	enter graphics mode

	GROFF	leave graphics mode
	GRCOL	set graphics colour
	PENUP	lift the pen
	DRAW	draw line
	DFAT	draw fat line
	DRSYM	draw defined symbol
	DRTXT	draw text string
	PENC	set pen color
	PENW	set pen width
	BRCOL	set fill color
	POLY	draw polygon
	RSIZE	set graphics scaling
	ORGIN	set graphics origin
	BMP	import and diplay bitmap (BMP file)
	GMODE	change currently open graphic window mode
	SCALE	define graphic frame
	WHEEL	draw chartwheel
	CIRC	Draw circle
	DRASP	customized chartwheel draw aspects
	DRHOU	customized chartwheel draw house divisions
	DRPLA	customized chartwheel draw planets
	DRSGN	customized chartwheel draw signs
	DRWTT	Draw text in graphics mode
	DRXAS	customized chartwheel draw X-aspects
	GCLR	clear graphics stack
	GPOP	Pop graphics stack
	GPUSH	Push graphics stack
	ROT	Set graphics stack rotate
	ROTAT	Rotate coordinates 2D
	ROTX	Rotate polar coordinates 3D around X axis
	ROTY	Rotate polar coordinates 3D around Y axis
	ROTZ	Rotate polar coordinates 3D around Z axis
	SHAPE	Draw shape from values in string
Macro:	**CML**	execute defined macro (destruktive)
	CALL	call defined macro (non-destruktive)
	FUNX	call single PCA function
	MACn	call single key macro
	MAC.	Clear window
	MACOF	Suppress all output
	MACON	restore output
Configure	**COL**	insert colour definition

	CHROT	set chartwheel rotation
	CONFX	set special configuration value
	SYMIX	create symbol choice table
	PXL	setup user defined printer translation table
	AYA	set ayanamsha value (minutes of arc)
	PTMP	backup system settings to restore at module exit
	TIDY	restore system settings before module exit
	SYSTR	Set or read system string
	SYSAV	save selected part of system configuration to disk
	GAZ	set atlas to ACS or simplified
	PMIS	lookup if given string is contained in features
	REVNO	Argus Revision number
	SWFLG	Set extra Swiss Ephemeris flags
Misc:	**SYN**	initialise synastry
	NAX	auxiliary namefile access
	PPATH	get Argus program path
	XPATH	get current XLI module path
	PROFL	Profiling - set/reset timer
	PLIN	not used
	SNO	Argus serial number AND 7FFF (15 bits - obsolete)
	USR	Retrieve USR code from interpretation
Security;	**PASSW**	encrypt password string
	SCM	Read simple encryption key

FUNCTION DETAILS

<
<~

Stack: X Y < → Y < X

"Less than" checks if Y is less than X. If so it places a 1 on the stack, else a 0.
The FP version (<~) works on the FP stack and will return 0.0 or 1.0

< >
<>~

Stack: X Y < > → Y < > X

Not equal: Checks if Y is different from X. If so it places a 1 on the stack, else a 0.
The FP version (<>~) works on the FP stack and will return 0.0 or 1.0

=
=~

Stack: X Y = → Y = X

"Equals" checks if Y is equal to X. If so it places a 1 on the stack, else a 0.
The FP version (=~) works on the FP stack and will return 0.0 or 1.0

>
>~

Stack: X Y > → Y>X

"Greater than" checks if Y is greater than X. If so it places a 1 on the stack, else a 0.
The FP version (>~) works on the FP stack and will return 0.0 or 1.0

ABS (absolute)
ABS~

Stack: X ABS → abs(X)

Absolute value means changing to positive number, if X is negative.

The FP version (ABS~) works on the FP stack

ACOS

FP Stack: x ACOS → arccos(x)

arccosinus (degrees)

See also SIN COS TAN ASIN ACOS ATAN

ADD
ADD~

Stack: X Y ADD → X+Y

Adds two numbers
The FP version (ADD~) operates on the floating point stack

See also DIV SUB MUL MOD DIVR MULT

AGE~

FPStack: AGE~ → age in years

Computer clock - birthtime in the radix data input (not the current). The birthtime must either be entered with MEPUT radix (20-31) or by running the radix calculation e.g. MACR which will copy the current data to the radix data input.

ANAME

Converts an aspect number to a negative system string number for use with NUMS. ANAME is the equivalent of writing: 618 ADD CHS

The CFG lines holding the planet, sign and aspect names are ASCII and do not hold symbols. To be able to display either abbreviations or symbols depending on the program setting, the NUMS mechanism is intercepted, so that accessing these strings will be translated to symbols. For example -542 NUMS should normally print "Moon" in the @@@@ template, but if the Argus is set for symbols it will write the Moon symbol instead. Because it is difficult to memorise and calculate the cfg indexes each time you want to display a planet or sign, the codes PNAME, ANAME and SNAME can be used to fetch the index. So 1 ANAME NUMS will display a conjunction symbol or abbreviation, 3 PNAME NUMS a Mercury etc.

see also PNAME, SNAME

AND
AND~

Stack: X Y AND → X AND Y

Checks if two conditions are both true. This means that:

1 1 AND → 1
1 0 AND → 0
0 1 AND → 0
0 0 AND → 0

1 represents "true" and 0 "false".

Technical note: if AND, OR is used on other numbers than 1 and 0 the result will be a "bitwise" comparison. To be able to analyse the result, you will have to convert the numbers to binary, i.e. each 16 ones and zeroes, and compare each set of bits separately. This may be used creatively if you are experienced in this field. If not, better be sure, that you use the function on zeroes and ones only.

The floating point version (AND~) will round the two values and return AND converted back to floating point.

ANTOI

p n ANTOI → (integer)

Fetch item p in a commaseparated list in stringarray[n] and convert it integer.
P must be a number between 1 and the last number in the list. If p is 0, the return value is also 0. If p is larger, the last value in the list will be returned.
If an item contains other that digits, the first group of digits will be used
If an item has no digits it is invalid and the next one will be used. This should be avoided
This code is used to retrieve integer values from a commaseparated data string, which could look something like
the string defined below. The result in the example below will be 3980.

```
$
1 STDEF

274,993,3287,11281,2,435,882,3980,9999,10039,4,0,34,.8123
$
8 1 ANTOI NUMS

value 8= #######
$$$
```

ANUM (aspect number)

Stack: X Y ANUM → (number)

This code tells the kind of aspect between X and Y.

No aspect	0
Conjunction	1
Opposition	2
Square	3
Trine	4
Sextile	5
Semisquare	6
Sesquisquare	7
Inconjunct	8
Semisextile	9

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

AORB

Stack: X Y AORB → (orb)

This code tells the orb of the aspect between X and Y. The result is in tenths of a degree. The orb value may be positive or negative:

If the angle measured from X to Y is less than the ideal aspect angle, the orb is positive, if it is more than the ideal angle, it is negative.

For example if:

Angle from Sun to Mars 89 degrees :	1 5 AORB → 10
Angle from Mars to Sun 271 degrees:	5 1 AORB → -10

The result 10 means one degree (ten tenths). It may help to imagine a decimal point.

Even if the aspect is out of the defined orb, this function will still return a nonzero value. This value will apply to the closest aspect. For instance if the angle between X and Y is 131 degrees, you will get the result 40 because it is 4 degrees from a sesquisquare. This works even if you cancelled out sesquisquares by defining their orb to zero in the preferences.

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

APOW (aspect power)

Stack: X Y APOW → (power)

The aspect power is a number between 1 and 10 dependant on the actual orb of the aspect. If the orb is zero (exact aspect), the power will be 10, if it is just on the orb limit the power will be zero. So the power will depend on the maximum orb you defined in the menu and in the aspect orb installation.

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

AREA

n AREA

Loads the timezone area code in the latest calculated chart into string array n.

See also CNTRY

ARTOE

X ARTOE → Ecliptic value

Converts Right Ascension for a point on the ecliptic to its ecliptic longitude. Input and result values are Intang.

ASIN

X ASIN → arcsin(x)

Input and results on the FP stack

ATAN

X ATAN → arctan(x)

Input and results on the FP stack

AYA (Ayanamsha)

Stack: X AYA →

Changes the Argus Zodiac configuration to user defined ayanamsha X (Intang).

Some astrologers use the siderial zodiac instead of the tropical one used by most western astrologers. Hindu astrology refers always to the siderial zodiac which rely on the fixed star constellations rather than on the equinoxes. There are however disputes, which fixed stars should define the siderial zodiac starting point.

The AYA code lets you define an "ayanamsha", the difference between the two zodiacs, and automatically subtract this from all calculations. This will apply as long as you are working from the XLI interpreter. As soon as you return to the main program, PCA will revert to the tropical zodiac. If you want the main program also to use your defined ayanamsha, you may use the CONFX code, see this. Position printouts with ayanamsha, will have the ayanamsha angle written under the date, time, etc.
To enter your favourite ayanamsha, you must know its size at the 1.st of january 1900, measured in minutes of arc. For example to insert LAHIRI ayanamsha, code:

```
$
1348 AYA

$
```

BEWARE The inserted value is lost when Argus is terminated. Modules using the PTMP code will also cancel any ayanamsha set with the AYA code or in the preferences. This is actually an unwanted side-effect which will be fixed in Argus 4.2 rev 25.

AZP (aspect primitive)

Stack: X AZP → (orb) (aspect no 1-9)

This is an aspect routine giving just raw data without using the orb limits inserted in the main and installation menus. Also, the values are intang units. The function produces both the aspect number and the actual orb.

The X is the angle between the two planets measured in intang units. This may be calculated using the PPOS code to get the planets positions and then subtracting them to get the angle. For example:

```
1 PPOS 4 PPOS SUB AZP
```

Aspects numbers returned are:

0 - 20 degrees:	Conjunction
20 - 40 degrees:	Semi-sextile
40 - 50 degrees:	Semi-square
50 - 70 degrees:	Sextile
70 - 110 degrees:	Square
110 - 130 degrees:	Trine
130 - 140 degrees:	Sesquisquare
140 - 160 degrees:	Inconjunct (Quincunx)
160 - 180 degrees:	Opposition

The orb is a signed number giving the distance from the ideal angle. The sign is calculated the same way as in the AORB code (see this), but the orb is given in intang units.

AZE

Stack: v AZE → (orb) (aspectno)

This is a variant version of AZP (see AZP) including all valid aspects. The aspect has to be within the current orb definition for that aspect, i.e. if aspect scheme R is selected all aspects are checked against the aspect orb in scheme R before they are accepted.

BDATA

Stack: N Q BDATA →

Output selected parts of birth data from current input menu to string Q

N bit 0 : name
bit 1 date:
bit 2 time:
bit 3 zone:
bit 4 latitude:
bit 5 longitude:
The bits can be combined, if e.g. just bit 0 and bit 4 is set, N will be 17, and the ouput will be name+latitude.

Q: string number to recieve the data

Q may also have negative values in which case:
-1: output data to screen
-2: output data to screen witn linefeed

BDPR (birth data print)

Stack: BDPR →

Print date, time, longitude and latitude on four lines. This code is useful together with the code NPR in the start of an interpretation to print birth details.

BDSEL

Stack: n BDSEL →

Select input menu data card: 0=radix 1=current 2=Present. So for example, to make a radix chart for the time now, you could code 2 BDSEL MACR to make the radix calculation take the input data from the present data menu with the horary data.

Index 0: Currentdata
1: Radixdata
2: Currentdata
3: Temprdata
4: Tempcdata
5: Presentdata

This does not affect the actual card selection in the user interface.

Index 3 and 4 (temprdata and tempcdata) lets you make the selectedpointer point to the data saved by PTMP. This means, if you use PTMP and then change the radixdata and/or currentdata, you can get access to the original data using 3 BDSEL or 4 BDSEL. You can then use SWDAT to swap these data around. Be aware though, that this could change the original data you tried to save with PTMP.

BHUS

Stack: X Y BHUS → (housenumber 1-12)

This code will find the house position of any point using the houses of either the latest radix or the latest non-radix.

X: the position to test in intang units

Y: 1 for latest radix houses
2 for latest non-radix houses

For example having calculated radix and solar return, to find the position of the radix Moon in the Solar return houses, use this coding:

22 PPOS 2 BHUS

The planet numbers allowed are listed under PSI.

BMP

x0 y0 x1 y1 BMP (Draw bitmap)
You can load a bitmap from a BMP file into your graphic.
A graphic must have been initializes with the code GRON creating a square area 2048 x 2048 pixels with an graphics origin in the centre. So coordinate X can be -1024 (left) to +1024 (right) and Y can also be -1024 (bottom) to 1024 (top).
x0 and y0 is the coordinates of the top left corner. If you set x1=x0 and y1=y0 then the bitmap will have its normal size defined by the bitmap itself.

You will probably prefer to stretch it to fit your needs. In that case you should set x1 and y1 to where you want the bottom right corner. For example if you set the four coordinates to
$
GRON -1024 1024 1024 -1024 BMP GROFF

yourimage.bmp
$
In this example, the bitmap in file yourimage.bmp will fill your square completely.
If x0>x1 the bitmap will be mirrored left-right and if y0>y1 then the bitmap will be mirrored upside down.

The bitmap file can be a .BMP .EMF .JPG or .ICO, the latter however does not seem to be scaleable.

BOO (Boolean)
BOO~

Stack: X BOO → (1 or 0)

Checks that a number is nonzero. This code is used, when you wish to combine numbers with AND and OR, and these numbers may be more than just 0 or zero. Having calculated for example two sets of points, then to check, that they are both nonzero, you would code

X BOO Y BOO AND

Just writing X Y AND will do a bitwise AND and this will probably not be what you are looking for.

The floating point version uses the FP stack

BRCOL (fill color)

000 BRCOL changes brush color to black, 080 BRCOL changes the brush to green etc. The brush is the colour used for filling polygons.

BROWS

BROWS Open url in default browser. The URL in question must be placed on the first line in the text part of the paragraph.

You may use the code NUMS before BROWS to pass data to the url:

$
1030 NUMS

BROWS

laurids.com/mswheel.html?bdate=1912####
$

CALL

Stack: CALL →

Execute a specified macro. The macro must be given as the first line in the text field. CALL reads a line in the text field and executes it. The example below will insert the current date and time and calculate a radix chart.

$
CALL

1T.R
$$$

See also CML

CARY

Stack: X CARY →

Reads lines of strings into the string array. For explanation of the string array see the chapter on this.

The strings must be separated by commas and will read into array index 0, 1, 2 etc.

For example:

$
2 CARY

ari,tau,gem,cnc,leo,vir,lib,sco,sgr,cap,aqr,psc
Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
$

This will read the sign and month names into the string array. The sign names will get the indexes 0-11 and the months 12-23.

Note that there is no comma at the end of the line, the line shift will act as a comma. If you put a comma in front of the first line, the signs will get indexes 1-12 and the monthes 13-24.

CARY is used for mass loading of the string array. To load a single entry, use the code STDEF.

WARNING: CARY will whipe all earlier definitions of the string array, e.g. NAME.

CENT

Use this code to center text output. 1 CENT will make the following text adjust centered. 0 CENT will turn this off.

CENT does not work on graphics output

CHROT (chart rotation)

Stack: X CHROT →

The PCA chartwheel presentation have a couple of rotational variants, you may set with this code. The possible values for X are:

0: ascendant horizontal (default)
1: MC vertical
2: radix ascendant horizontal
3: radix MC vertical
4: aries to the right
5: aries up
6: aries to the left
7: aries down

The meaning of 2 and 3 is that the signs will have the same orientation as in the previous calculated radix chart, so it will be easy to realise how much progressed angles have moved.

CHS
CHS~

Stack: X CHS → -X

Change sign means changing to positive number if X is negative and vice versa.
The FP version works on the FP stack

CIRC

Stack: X Y R CIRC →

Draw circle with centre in X,Y and radius C. Graphics mode must be set to on)using the GRON code). You may use code PENW and PENC to set pen width and color first.

CLRIF

Stack: →

Clear all IF conditions. Use this code if you do not know if code interpretation is suspended due to a previous IF statement, and you wish to force all previous IF conditions cancelled and the code to execute anyway.

See also IF ENDIF XIF

CML (commandline/Queue)

Stack: CML → 0

This code will setup a macro, which will execute as soon as the XLI module has finished. It will overwrite any queue pending. This code may be used for example in an AUTO.XLI module, which will force PCA to do specialized tasks as soon as it starts, before it leaves control to you showing the main menu.

The macro must be given in the text field. CML reads a line in the text field and executes it. The example below will call a radix and aspects using the menu values.

```
$
CML

RA
$
```

A more flexible macro handler is the code CALL (see this).

CNT (count)

Stack: X CNT → (counter value)

This code is used inside a FOR NEXT counting loop to get the current value of its counter.

If only one loop is started, X must be one. If more than one loop is running inside each other, X must be 1 for the outermost loop (the first one started), 2 for the next and so on. Up till 10 loops may be running simultaneously. For an example, see code FOR.

See also FOR NEXT XIF

CNTRY

n CNTRY →

Loads the country name from the "current" latest calculated chart into string array n.
So it is not enough to get the country into the input menu, you must calculate the chart first befor using CNTRY

See also AREA

COL (colour)

Stack: X Y Z COL →

Set the colour for one colour item (a planet, a sign, an aspect or panel background). This code allows to customise the individual PCA colours. It has the same effect as using the colour setup in the installation menu.

X is obsolete, was 1 or 2 in the DOS version to select screen or printer colors differently. In Argus it is needed but unused, any value will do.

Y is the colour item number (0-43). Each item, that you may assign a separate colour has an item number. For a list of these numbers, see GRCOL.

Z is the colour number (0-888), where each digit represents the quantity of red, green and blue respectively. So bright red has number 800, bright green 080 and blue number 008.

You may use the COL just for the current module, starting with PTMP, the original colors will be restored at module exit. See PTMP how to do this.

CONFX (configuration extras)

Stack: X Y CONFX → X

A couple of setttings for Argus are saved in the confix array, which you can access with CONFX. You may also access and modify it in the preferences, "edit system variables", it consists (currently) of 64 integers, some of which have different use for single bits.

For example confix[0]

1 0 CONFX : (bit 0) Chiron included
2 0 CONFX : (bit 1) midpoint trees aspect type and orb suppressed
4 0 CONFX : (bit 2) - unused
8 0 CONFX : (bit 3) - No header
16 0 CONFX :(bit 4) Huber age point Jan Romander (slightly different speed)
32 0 CONFX :(bit 5) Print jobdate
64 0 CONFX :(bit 6) page numbers suppress
128 0 CONFX (bit 7) extended page number format "page xx off qq)

Other confix settings are
confix 3 0 No degrees and minutes on chartwheel
1 degrees on chartwheel
2 degrees and minutes on chartwheel
confix5: 0 use symbols in printouts
1 use latin abbreviations in printouts
2 use national abbreviations in printouts
confix 6: Tabline colour 000-888 or -1 for text color;
confix 7: Reserved for black use of Argus (no menus, auto-quit)
confix 8: bit 0=1 No orbspeeds, aspects printed in 4 columns
bit 1=1 unused
bit 2=1 No aspects on bi-wheel
bit 3=1 Short radix houses on bi-wheel
bit 4=1 include bonatti sections
bit 5=1 suppress chartwheel aspect to angles
bit 6=1 include PtFt on chartwheel
bit 7=1 horary clock graph includes all positions to the left
bit 8=1 center chartwheel on page
confix10: bit 1=1 suppress progressed moon in secondary and tertiary
confix 11: bit 1=1 sets orbcombine to minimum for planet pairs with a zeroorb planet.

Notes:

The true equation for Huber AP uses the Koch house equations rather than the interpolation which is used by the Huber school. Using the true equation seems more "mathematical correct", and makes the movement of the AP more smooth.
Page number suppressing is normally set in the print layout preferences
Return value: The CONFX code returns the old value. This allows you to test, for example just getting the value of CONFX 0 without changing it could be done like this:
0 0 CONFX 1 DUP 0 CONFX DEC
!!If you want to set or reset one bit in CONFX 0 the following codes will do just this:
0 0 CONFX 128 OR 0 CONFX DEC ;sæt bit 7 i CONFX 0
0 0 CONFX 64 CPL AND 0 CONFX DEC ;sæt bit 6 i CONFX 0

CONT

Continue from WAIT and skip the waiting text

See also: WAIT

COS

FPstack: X COS → cos(x)

FP Cosine (X degrees)

See also SIN TAN ATAN ACOS ASIN

CPL
CPL~

Stack: X CPL → not(X)

Reverse bits

Bitwise complement. This is a highly technical code changing all zero bits to ones and vice versa. To analyse the result, you will have to convert X to binary, i.e. to 16 ones or zeroes.

The FP version rounds the argument on the FP stack, reverses the bits, converts back to FP and pushes it back on the FP stack

---

DDIF (Date difference)

Stack: D1 M1 Y1 D2 M2 Y2 DDIF → (difference)

Calculates the distance in days between two dates:

Y1 = date1 - years
M1 = date1 - months
D1 = date1 - days
Y2 = date2 - years
M2 = date2 - months
D2 = date2 - days

The difference is date1-date2, for example:

1950 3 6 1950 4 6 DDIF → -31 (days)

The difference may be up to +- 32767 days. If the distance is greater, you will get the value 32767. This code is used in the biorythms module to find the age in days.

To calculate greater differences, you must put the dates on the input menu and use FP functions GETTI and SETTI to convert the dates to FP.:

```
D2 0 MEPUT
```

```
M2 1 MEPUT
Y2 2 MEPUT
GETTI FPSUB
D1 0 MEPUT
M1 1 MEPUT
Y1 2 MEPUT
GETTI
SUB~
```

The result will be on the FP stack.
See also GETTI SETTI JD

DECL

Stack: X DECL → declination

Convert intang ecliptic position to intang declination

See also: ARTOE ETOAR

DEBUG

Stack: DEBUG →

This code is used by the XLI programmer to check the effect of his coding. If you are in doubt whether a bit of coding is doing what you intend, you may temporarily insert this code just before the coding to examine.

As soon as the interpreter reaches DEBUG it switches to single step mode. This means, that it will execute just one instruction code at a time, then waiting for click on the step or the multistep button.

In single step mode, the screen will for each instruction display the instruction name and the topmost values on the stack. So you may follow step by step what your coding is doing.

To leave single step mode, click the Run button. just press ESC, and your module will execute on at normal speed. If within a loop, it may stop again next time the DEBUG code is met.

To cancel out the module, click the Escape button

See also: the chapter on debugging and codes XLOG and LOGX

DEC (decrement stack pointer)
DEC~

Stack: X Y DEC → X
Decrement code will remove the upper number on the stack. This is useful if some superfluous stuff must be disposed of. For example:

X Y Z STO → X Y
X Y Z STO DEC → X

The first line will store Y in memory cell Z, but Y will still be left back on the stack. If you do not need Y any more but the X beneath, the second line shows how DEC will dispose of Y.

The FP version works on the FP stack

DELAY

Stack: X DELAY →

Pause for approximately X milliseconds. However, for output, the screen will normally wait until the module is finished before showing anything, so the value is limited.

Please note that the milliseconds measure is not exact, but depends on your computer. 486's seem to have the double speed, so for instance 2000 DELAY causes a pause of one second instead of two.

DFCOL

Stack: R G B N DFCOL →

Argus has a table of "simple colors" using values from 0-888 defining mixing red (first digit), green and blue (last digit) in proportions. So e.g, 800 is pure red 80 pure green, 0 is black. Values having digits 9 are undefined, but can be assigned any color with DFCOL.

The simplified colors represent 729 different colors, whereas DFCOL has better resolution (0-255) for each component giving more than 16 million diffent colors. With DFCOL you can mix any of these and assign the result to one of the 271 unused color numbers in the palette. For example:

255 84 120 900 DFCOL

will assign a warm red to the color number 900.

If you now later use color 900 e.g. to set the Argus panel color: 0 0 900 COL, you will get a warm red background on the Argus icon panel. (You will need to right click the icon panel, then cancel to have the panel color materialize) The panel and other Argus colors are normally defined in the preferences menu.

You may also use DFCOL the same way to redefine the first 888 colors in the color palette. This will give you a 999 different definable colors palette to play with. But the color numbers will then be less recognizable as red,green,blue values.
See also COL PENC BRCOL

DFAT (Draw fat line)

Stack: X1 Y1 X2 Y2 N D DFAT →

Draw a multiple line (like the ones used for exact aspects) from X1,Y1 to X2,Y2, consisting of N parallel lines D units apart. D should be only a few units to simulate one thick line.

This code is rather obsolete. It was used in the DOS days, where only thin lines were availabel. With Argus, you would prefer printing a single line using the DRAW code and defining the stroke width with code PENW

See also: PENW PENC DRAW PENUP POLY

DGREE

Stack: radius wcirc wdcirc l30 w30 l10 w10 l5 w5 l1 w1 DGREE

Draw circle with 360 degree markings

To draw a degree scale, the DGREE code provides a lot of control. The arguments are:

radius: radius of the base circle
wcirc: linewidth of base circle
wdcirc: linewidth of extra circle (set to 0 for no extra circle)
l30 :length of 30 degrees divisions
w30 :width of 30 degree divisions
l10 :length of 10 degree divisions
w10 :width of 10 degree divisions
l5 :length of 5 degree divisions
w5 :width of 5 degree divisions
l1 :length of 1 degree divisions
w1 :width of 1 degree divisions

example:
500 5 2 -100 10 -45 7 -30 5 -18 0 DGREE

Please note, that all parameters other than radius are measured in promilles of radius, so that you can easily scale up and down just by changing the radius and keeping the other parameters.

Note also, that you can use negative values for the lengths of the degree markings, which will make them turn inwards.

DIV
DIV~

Stack: X Y DIV → X/Y

Divides X by Y. If Y is zero, you will get an error message.
FP version operates on the FP stack
See also ADD SUB MUL MOD DIVR MULT

DIVR (divide with remainder)

Stack: X Y DIVR → X mod Y X/Y

This is a combined division and modulus function. It is equivalent to X Y 2 DUP 2 DUP MOD 3 FETCH 3 FETCH DIV.

If Y is zero, you will get an error message.

DNORM

Stack: DNORM →

Normalize date input menu values. It is perfectly possible to use MEPUT to enter out-of-range day, month or year values, then you could use the DNORM to convert to a valid date.

You may also use this to check if an entered day is valid:

```
$
0 MEGET 1 MEGET 2 MEGET DNORM
2 MEGET =
XY 1 MEGET = AND
XY 0 MEGET = AND

Date is valid
$
NOT

Date is invalid
$
```

See also: MEPUT SETTI

DRASP

Stack: chart spread p1 p2 r lw DRASP →

customized chartwheel draw aspects

The aspects drawing DRASP has the following parameters:

chart	:stored chart (radix, transit etc) as with planets
spread	:if 1:endpoints at stored spreaded positions. if 0: true positions
p1:	lowest planet (e.g. 1=Sun)
p2	highest planet (e.g. 10=Pluto)
r	radius where endpoints should go
lw	linewidth

To use spreaded positions, you must first calculate these using the code SPRCH. The chart memory has a separate array for these positions, which are undefined until SPRCH is executed.

See also DRXAS SPRCH SPRED DRAW DRPLA DGREE PENC PENW DRHOU

DRAW

Stack: X Y DRAW →

Move the pen or draw a line to position X,Y. If it is the first DRAW instruction at all or the first after a PENUP instruction the pen will move to X,Y without drawing, else it will draw a line to X,Y from where the last DRAW instruction left the pen.

Example: To draw two rectangles you may code:

```
$
GRON
3 GRCOL
-800 -600 DRAW
-200 -600 DRAW
-200  600 DRAW
-800  600 DRAW
-800 -600 DRAW
PENUP
800 -600 DRAW
200 -600 DRAW
200  600 DRAW
800  600 DRAW
800 -600 DRAW
PENUP
WKEY
GROFF
```

$
See also: POLY PENUP PENC PENW

DRHOU

Stack: chart h1 h2 r0 r1 arlen DRHOU →

To draw houses, use the DRHOU code. The parameters are:

chart: chart number as for planets
h1 first house number
h2 last house number
r0 radius of start of house division line
r1 radius of end of house division line
arlen length of arrow

See also: DRXAS SPRCH SPRED DRAW DRPLA DGREE PENC PENW DRHOU

DRPLA

Stack: chart loplanet hiplanet planetsize rinner rplanet router DRPLA

customized chartwheel draw planets

The parameters for DRPLA are:

chart :chart number
lo planet :lowest planet number
hi planet :highest planet number
planet size ;degrees as seen from chart centre
rinner ;radius of inner circle to where positions marks point
rplanet ;radius where planets should appear
router ;radiuo of outer circle to where position marks point

See also: DRXAS SPRCH SPRED DRAW DRPLA DGREE PENC PENW DRHOU

DRSGN

Stack: rcen gwid ghei rot DRSGN

Customized chartwheel draw sign symbols

The parameters for DRSGN are:

rcen Radius to the centre of the sign glyps.
gwid width of each glyph

ghei height of each glyph
rot rotation: angle of zero Aries in intang

See also: DRXAS SPRCH SPRED DRAW DRPLA DGREE PENC PENW DRHOU

DRSYM (Draw symbol)

Stack: Z R S X Y DRSYM →

Draw a symbol from the PCA symbol table:

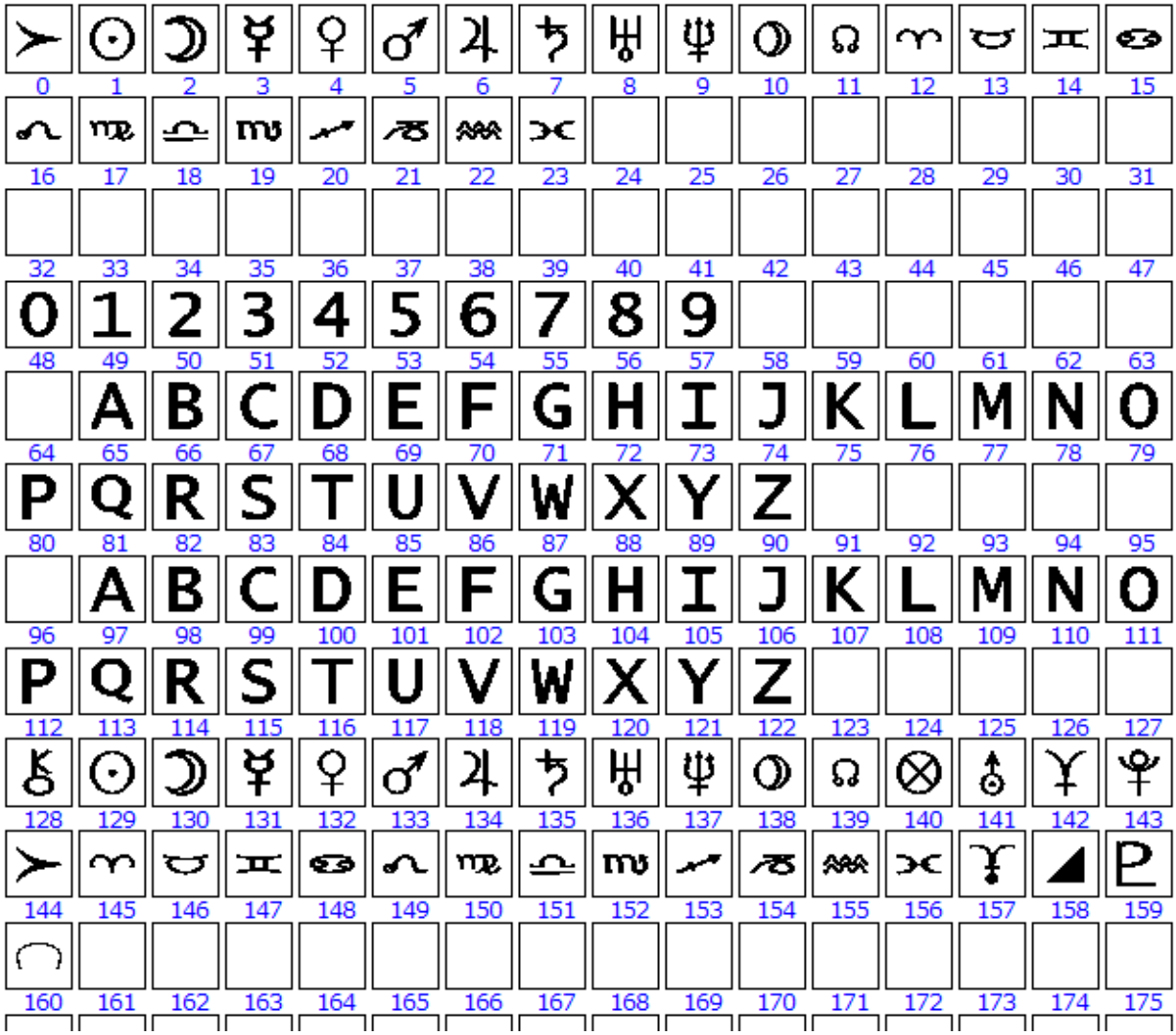
Z is the symbol number
R is the rotation
S is the size
X and Y is the coordinates where the symbol center should be

DRSYM uses the current pen color and width set with PENW and PENC

The following code will print all the symbols available:

```
;DRSYMS XLI MODULE
$
1800 2100 SCALE
GRON
4 PENW
0 15 FOR
0 15 FOR
1 CNT 2 CNT 16 MUL ADD          ;sym no
0                                ;rotation
180 0 STO DEC 120                    ;siz
-1500 1 CNT 200 MUL ADD 1 STO  ;x
1900 2 CNT 250 MUL SUB 2 STO   ;Y
5 DUP 5 DUP 5 DUP 5 DUP 5 DUP
DRSYM
DEC DEC DEC DEC DEC
1 RCL 0 RCL 2 DIV SUB
2 RCL 0 RCL 2 DIV SUB
2 DUP 2 DUP DRAW
2 DUP 0 RCL ADD 2 DUP DRAW
2 DUP 0 RCL ADD 2 DUP 0 RCL ADD DRAW
2 DUP 2 DUP 0 RCL ADD DRAW
2 DUP 2 DUP DRAW PENUP
1 CNT 2 CNT 16 MUL ADD 1 NTOS
6 8 70 0 6 DUP 100 ADD 6 DUP 1 DRWTT
NEXT
NEXT
GROFF
```

$$$



DRTXT (Draw text string)

Stack: Z R S X Y DRTXT →

Draw string array index Z:

Z is string number

R is the rotation

S is the size

X and Y is the coordinates where the first symbol center should be

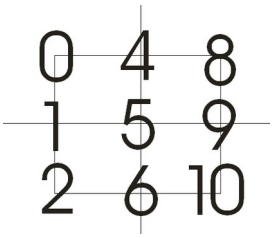
---

DRWTT

M C Z T X Y S DRWTT ;draw text using tt-font

This code inserts text into a graphic. You can also use DRTXT to do that, but in that case you have only the plotted symbols whose shapes are defined in PCA.CFG. This code uses the current font, but text cannot be rotated.

M is the mode. The mode defines how the text is justified.
0 1 2 3: Horizontally right adjusted to y-axis
4 5 6 7: Horizontally centered around y-axis
8 9 10 11: Horizontally left adjusted to y-axis
12 13 14 15: Horizontally window centered
0 4 8 12: Writing above x axis
1 5 9 13 Writing on X axis (vertically centered)
2 6 10 14: Writing below x axis
3 7 11 15 Writing at bottom of screen



C is the color using the 000-888 RGB color system of Argus.
Z is the font size, not points but the Argus graphic units, so that 2047 is the full window heigth;
T is the font style, 0=normal 1=bold, 2=italic, 4=underline. The figures may be added to create compound values, e.g. 1+2=3 means bold-italic.
X and Y define the text placement in the graphic coordinate system, i.e. 0,0 is window center, and +-1024 are the screen edges.
S is the string index holding the text to draw

DRXAS

Stack: chart0 chart1 spread lopla0 hipla0 lopla1 hipla1 r0 r1 DRASP

customized chartwheel draw cross aspects between two charts

The aspects drawing DRASP has the following parameters:

chart0:	stored chart (radix, transit etc) as with planets
chart1:	second stored chart
spreead :	endpoints at stored spreaded positions. if 0: true positions
lopla0:	lowest planet (e.g. 1=Sun) in chart 0
hipla0:	highest planet (e.g. 10=Pluto) in chart 0
lopla1:	lowest planet (e.g. 1=Sun) in chart 1
hipla1:	highest planet (e.g. 10=Pluto) in chart 1
r0:	radius where endpoints should go chart 1
r1	radius where endpoints should go chart2:

To use spreaded positions, you must first calculate these using the code SPRCH. The chart memory has a separate array for these positions, which are undefined until SPRCH is executed.
See also DRXAS SPRCH SPRED DRAW DRPLA DGREE PENC PENW DRHOU

DUP
DUP~

Stack: X Y DUP → X (duplicate)

Duplicates one of the numbers of the stack. To duplicate the uppermost, use 1 DUP, to duplicate the next use 2 DUP etc.

Examples:

X Y Z 1 DUP → X Y Z Z
X Y Z 2 DUP → X Y Z Y
X Y Z 3 DUP → X Y Z X
X Y 2 DUP 2 DUP → X Y X Y

The FP version works on the FP stack

ELSE

The ELSE code can be used in IF..ENDIF constructions.

See also: IF ENDIF

ENCOD

Stack: b ix ENCOD →

Encode text string in string[ix] between ANSI OEM and UTF-8 and URL

ix: string index

b: 0: ANSI to OEM
1: OEM to ANSI
2: ANSI to UTF-8
3: ANSI to URL encode
4: URL encode to ANSI

ENDIF

Stack: ENDIF →

Description: Ends the current IF condition. You must always have a matching number of IF's and ENDIF's. See code IF for further explanation.

See also IF ELSE

ETOAR

Stack: ecl ETOAR → ar

ecl: ecliptic position in intang units

ar: right ascension

The function assumes no ecliptic latitude.

See also: RE2SP, SP2RE, RTP, PTR, FPRTP, FPPTR

EXEC

Stack: EXEC →

Call external program. This may be any EXE program or file with an extension, which windows knows how to open.

```
$
EXEC


argus.bmp
$$$
```

You may put an application and add arguments for the application to open.

FEED

FEED~

- obsolete

FETCH

Stack: X Y FETCH → X (fetched)

Reorders the stack by fetching the value Y steps down the stack and putting it on the top.

Examples:

X Y Z 2 FETCH → X Z Y

X Y Z 3 FETCH → Y Z X

1 FETCH has no meaning, as it will just fetch the uppermost, and put it at the same place.

FLOOR

FP-Stack: X FLOOR → x

FLOOR FP Floor function returns the nearest integer below X. .

See also INT~ ITOF FTOI

FONT

Select font number defined with FONTS. 0 FONT selects the first font defined, 3 FONT select font 3, 13 FONT selects font 13 and so on. The font is set for text output. If graphics selected with the GRON code, it is set both for the text screen and for the graphics screen where it can be used by the DRWTT command.

The FONT definitions include size, style, color and font name, even if the DRWTT use only the name overrideing both size, style and color.

See also FONTS

FONTS

Stack: n FONTS

n is the number of font definition lines, which must be placed in the text part of the paragraph.

This command lets you define up till 15 fonts to use in your interpretation. A font is defined by its size, style, color and name.

As soon as they are defined, you can switch between them in you text. The FONT code sets the font for an entire paragraph, but you may also switch between font 0-7 inline by inserting special characters (184-191)

If you just want to change between colors, you can define a number of fonts with the same name, style and size and give them different colors. Here is an example of defining 3 fonts. You must put the number of fonts you want before the FONTS code and provide the same number of lines in the text below holding the definitions:

3 FONTS

9 0 0 Arial
9 1 0 Arial
10 0 800 Argus

Don't get tempted to put comments in the definition lines, The font name must be the last item in the definition line.

This defines font 0 as Arial with size 9 points normal style and color black.
Font 1 is defined as Arial size 9, Bold and color black.
Font 2 is defined as Argus symbol font size 10 normal style and color red.

When defined, you can choose font 0-7 inside the text using the characters number 184-191. If you have defined more than 8, fonts 8-15 can only be selected using the code FONT, which works for a whole paragraph.

You may leave out the font name in the definition lines. In that case the default font typeface is used as setup in the Argus preferences.

See also: FONT DRWTT

FOR

Stack: X Y FOR →

This is a looping instruction, so that you may have your code execution repeated a number of times. It works together with the code NEXT.

The loop is running a number of times determined by X and Y. A counter starts at X and ends at Y. The counting may go either up or down so both 1 8 FOR and 8 1 FOR are valid. The loop will always run at least 1 time, so for example 2 2 FOR will run 1 time, 2 3 FOR two times etc.

Example: 0 10 FOR 0 1 CNT STO NEXT

will fill the value zero into memory cells 0-10 inclusive.

You may have several loops inside each other. For example, this coding will calculate the number of planetary aspects between two charts:

```
$
0                                   ;start value
1 10 FOR                            ;radix2 planet count
20 30 FOR                          ;radix planet count
1 CNT 2 CNT ANUM                ;test aspect number
BOO ADD                           ;increment if not zero
NEXT                                ;end radix count
NEXT                                ;end radix2 count
```

Do not rely on for-next structures working across files or function calls.

FPMOD

Stack: x m FPMOD → x mod m

FP modulus showing the remainder when dividing x with m. In Argus modulus of negative numbers is always positive. For example, if you have an angle of -35 ° modulus 360, for astrological needs, you want the result 325°, not -35, which would be the traditional result.

See also: MOD

FPPLA

Stack: pla n FPPLA → X

Calculate planetary info from data in the data input menu.

Pla: planet number (0-18)
n: which info:
0: geocentric longitude (degrees)
1: geocentric velocty (degrees per day)
2: geocentric latitude (degrees)
3: geocentric distance value (AU)
4: geocentric right ascension (degrees)
5: geocentric declination (degrees)
6: heliocentric longitude (degrees)
7: heliocentric velocity (degrees per day)
8: heliocentric latitude (degrees)

Please note, that the argurments pla and n are taken from the integer stack and the result is placed on the floating point stack.

FPPTR

Stack: r v FPPTR → X Y

Convert polar coordinates (r,v) to rectangular (x,y)

All values communicate on the FP stack.

FPRTP

Stack: x y FPPTR → r v

Convert rectangular coordinates (x,y) to polar (r,v)

All values communicate on the FP stack.

FSTKZ

Stack: siz FPstack →

resize FP stack to siz. Siz is taken from the integer stack.

Initially the FP stack has the size 32.

See also FSTOZ ISTOZ ISTKZ

FSTOZ

Stack: siz FSTOZ →

resize FP STO array to siz. Siz is taken from the integer stack.

Initially the FP STO array has the size 32

See also FSTKZ ISTOZ ISTKZ

FUNX

Stack: X FUNX →

Call a single PCA function, that is, those who have a single letter shortcut, you normally can activate by pressing a letter key, e.g. Radix, Aspects etc.

X must be the ascii-value for the letter you would have pressed in the menu, e.g. 65 for A (aspects) 80 for P (progressed) etc. Consult an ascii table to find the numbers you need.

This code is for single function calls only. It will usually be a more flexible solution to use the CALL function instead, which is more flexible and will execute a complete macro.

FTOI (convert floating point value to integer)

Stack: fp FTOI → int

This pops a value off the floating point stack, converts it to integer and puts it on the integer stack.

see also ITOF

GAZ

Choose a gazetter (atlas). Argus has two gazetters:

1: A simple one used for demo versions (32000 cities)

2: The ASC atlas (approx 271000 cities)

0 GAZ selects the simplified one-line

1 GAZ selects ACS

GCLR

Clear graphics stack. With graphics you may push and pop graphics state to make partial transformations (translate scale and rotate)

See also: GPUSH and GPOP

GET

Stack: X GET → (value at ZZO marker + X)

Pick a value from the integer stack placed at offest X from anchor set by code ZZO.

Gets the value X positions from the ZZO marker on the stack. You must at some point have set the ZZO marker before you use GET and PUT.

Examples:

ZZO 3 7 2 0 GET → 3 7 2 3
ZZO 3 7 2 1 GET → 3 7 2 7
ZZO 3 7 2 2 GET → 3 7 2 2

See also: PUT ZZO DUP FETCH

GETGL

Stack: n GETGL → size latitude longitude area

Get 1 line from atlas

n is the line number in the atlas to retrieve.
size returns the total lines in the atlas
latitude in minutes of arc (negative if south)
longitude in minutes of arc (negative if W)
Area code number part (if any else 0)
string results:
string 0: Location name
string 1: Letter part of area code
string 2: Population letter

GETTI

FP Stack: GETTI → GMT time in days since 1. jan 1900 at 0h GMT

get GMT time from input data as FP number. This will be a positive number if the date is in 19xx or later, and negative if before.

See also: SETTI GETZO SETZO MEGET MEPUT

GETZO

GETZO Get zone from input data measured in days as FP number. The result will be a decimal number between 0 and 1.

See also: SETTI GETTI SETZO MEGET MEPUT

GMODE

- not used

GPOP

GPOP Pop graphics stack.

See also GPUSH GCLR

GPUSH

Push graphics stack

The graphic stack holds the origin, the rotation, scaling, pen color and -width, brush color and width.

See also: GPOP GCLR

GRAF

Stack: X Y.... Z ascii GRAF

This code is used to produce horizontal bar graphs. The bars will have lenghts showing the size of numbers, e.g. element strengths, planetary power or whatever. The numbers to show graphically must be pushed on the stack (in reverse order as usual).

The graphs will appear in the text area starting where the character # is found. So where you want a bar to start, place a # and leave the remainder of the line blank, because it will be overwritten by the graph anyway. The bar lines can be mixed with lines, rulers etc., any text to explain or enhance the appearance.

You must put a blank line below the graph lines to make sure the graphs are printed.

There must be the same number of lines with a #-character as you want bars, and there must be the same number of numbers pushed on the stack (X Y Z..)

The numbers to show must be limited to positive numbers maximum 80, else you will get a range check error message. If the graph starts in a column other than 1 (as in the example below) the remainder of the line will be less than 80, and the number entered should be limited adequately.

After the numbers to show, you must stack the ASCII code of the character, with which you want to build the bar, typically a block graphics character (e.g. 220)

Example: Print a bar graph of the numbers in register (RCL) 20-23:

```
$
23 RCL 22 RCL 21 RCL 20 RCL 220 GRAF

---------------------------------------
Cell. 20 #
Cell. 21 #
Cell. 22 #
Cell. 23 #
---------------------------------------
$
```

GRCOL

Stack: X GRCOL →

Set graphics colour. This sets the colour used for any following graphics output. X is a number between 0 and 39. It represents the colours chosen for the different items in the PCA colour installation menu. X=0 will thus set the colour chosen for the text background, X=7 to the colour chosen for Taurus etc. The following is a complete list of colours:

0 : panel background
1 : Text foreground
2 : Graph background
3 : Graph main line
4 : Graph angles
5 : Graph houses
6 : Aries
7 : Taurus
8 : Gemini
9 : Cancer
10 : Leo
11 : Virgo
12 : Libra
13 : Scorpio

14 : Sagittarius
15 : Capricorn
16 : Aquarius
17 : Pisces
18 : Sun
19 : Moon
20 : Mercury
21 : Venus
22 : Mars
23 : Jupiter
24 : Saturn
25 : Uranus
26 : Neptune
27 : Pluto
28 : Node
29 : Conjunction
30 : Opposition
31 : Square
32 : Trine
33 : Sextile
34 : Semisquare
35 : Sesquisquare
36 : Inconjunct
37 : Semisextile
38 : Semi-quintile
39 : Quintile
40 : Tri-decile
41 : Bi-quintile
42 : Septile
43 : Bi-septile
44 : Tri-septile

So with this code you will not be able to set the colour to specifically "red" or "pink" or any absolute colour, you must select from the colour set already chosen in the colour installation menu. If however, you need a special colour setup, you may temmporarily create this suing the COL code.

See also COL BRCOL TXCOL DFCOL

GROFF (Graphics off)

Stack: GROFF →

Turn off graphics mode and closes the current graphic..

Note:! When XLI returns to PCA, a GROFF is automatically executed. Therefore, you do not need to state GROFF unless you wish to continue XLI in text mode.

See also GRON

GRON (graphics on)

Stack: GRON →

This code opens a graphic.In Argus from version 4.0 and later, output goes to a window (document), which can hold mixed text and graphics.

A graphic is a rectangular area on the output window, normally placed after the latest text output. The area width will fit the page width * the percentage defined in the Argus preferences "print page options", the same as used for the chartwheel.

Plot area: The default is a square 2048 x 2048 draw units with (0,0) in the centre. By using the SCALE code before GRON you can decide other rectangular dimensions. The graphic will occupy the same width in the output, but the plot units will be different. For example 4000 2000 SCALE will make GRON produde a plot area 8000 wide and 4000 high, so you can use coordinates x (--4000 to 4000) and y (-2000-2000).

To make a small graphic, you can of course plot with small numbers, but the area occupied will still take up most of the page width. A better way is to change the "wheelsize" option in the preferences. This can be done with XLI codes:

10 1 NTOS 15 -1 SYSTR ;set graphsize to 10% of page width

The original wheelsize should be restored at module exit (use PTMP code), you wont like Argus showing tiny chartwheels.

See also GROFF SCALE RSIZE ROTAT GPUSH GPOP GCLR

GTR (Graphic transits)

Stack: X Y GTR →

You may use this code to create a customized version of the graphic transits feature. The graphic transits/progressions are not real graphics but using text-mode and special characters (chr 178 if shaded blocks or chr 184-191 if bargraphs chosen in preferences).

The parameters you may change are the following:

1: Orb
2: Time slice
3: Calendar scale layout
3: Which transit and radix positions to include
4: Where to put horizontal lines

X is the time slice, calculated as 365 / X days.
Y is the orb in minutes of arc.
The other parameters are read from the first 3 lines in the text field:

<u>1.st line (choice of planets).</u>

Planets are assigned to letters, so that:

	Radix	Transit
Sun	A	a
Moon	B	b
Mercury	C	c
Venus	D	d
Mars	E	e
Jupiter	F	f
Saturn	G	g
Uranus	H	h
Neptune	I	i
Pluto	J	j
Node	K	k
Part fortune	L	
MC	M	
ASC	N	
11. house	O	
12. house	P	
2.house	Q	
3. house	R	

Note that you may not use a transit position of higher than the Moon's node.

The planets setup are in groups of five characters, for example:

1MNfk

This specifies the aspects to consider.

MN means: count from radix MC to radix ASC
fk means: count from transit Jupiter to transit Moon's node

The succession will be: First all aspects to radix MC, then all aspects to radix ASC. The last to letters are the "inner loop" or the fastest counting series. If you want the transits in the outer loop, that is, first all aspects from transit Jupiter, then from transit Saturn etc. you must put fk before MN:

2fkMN

Note the number before the block, that was 1 in the first example and 2 here. If you want the planets in the inner loop to be quoted first, write 2, else write 1. For example:

1fkMN MC tri Jupiter .....
2fkMN Jupiter tri MC .....

Normally you will want the transit first, so put 2 if you write the transits first and 1 if you write the radixes first.

You may put several blocks of 5, so you may have several sets of planet counts after each other in the same printout. For example if you want the transits Sun-Mars, but not Moon, you must setup first a loop for the Sun, then for Mercury to Mars:

1AKaa1AKce

You may put a line between the blocks by inserting an 'L':

1AKaaL1AKce

This will print first transit Sun to radix planets, then a line, and then transit Mercury-Mars to radix planets.

The planet setup line must be no more than 80 characters total.

2. Layout string:

The layout string has two purposes:

1: Determine the time span which is the number of characters times the time slice defined by the GTR instruction. For example if Y is 60 the time slice will be 365/60 = approx 6 days. If the string length is 60, you will have exactly one year of transits.

2: Design the ruler, i.e. the characters to appear, between the aspect markers (the shaded characters).

Below an example of a year transit ruler:

|....|....|....|....|....|....|....|....|....|....|....|....|

and a month transit ruler

. . . . | . . . . | . . . . | . . . . | . . . . | . . . . | .

!!3. The date marker line:

The third line determines where to put the date indicators in the header line. It may contain the following characters:

Y: Write the year at this position and increment year count

y: Just increment year count.
M: Write the month at this position and increment month count
m: Just increment month count.
D: Write the day at this position and increment day count
d: Just increment day count.
>: Start all the aspect rulers just after this position

Note: Y,M and D uses the starting year, and does not automatically take their position into account. Therefore the increment is necessary for possible further printouts. If you are not printing all increment steps, you must place lowercase letters in between to get the count correct.

The counts will just continue, so do not rely on correct calendar adjustment. However, if a month exceeds 12 its count will start over from 1, and the new year will be printed at its place (without updating the year count). See the XY option in PCA to view the effect.

Below is shown the example of a month transit date marker line. Only every 5 dates are shown.

M Y >D d d d D d d d d D d d d d D d d d d D d d d d D d d d d D

Study the TTRANSIT.XLI for detailed examples of alternative printouts.

HITS

Stack: T A1 A2 B1 B2 P1 P2 HITS →

This code is used for calculating aspects which has culminated between two instances in time. It therefore needs four charts: A1 and B1 which are the two aspecting charts at time t1 and A2 and B2 which are the same two charts aspecting at time t2. If A1 and B1 are the same chart it will mean I-aspects and the redundants are automatically omitted. Also sign shifts, house shifts, and stations may be checked.

T is the type of events:

1: aspects
2: sign shifts
4: house shifts
8: stations

If you want combinations, these figures can be added into a compound: eg. sign shifts and stations will make T=10.

A1 A2 B1 B2 are the chart numbers. Refer to the chart index number shown under MOVCH.

The two P arguments are not just two numbers, but several numbers defining a planet set. P1 is the planet set for chart A, (the promissors), P2 is the set for chart B (the significators). Planet sets are set up like this: P Q R ... S N. The last number is the count,

e.g. 1 2 3 3 means Sun, Moon, Mercury totalling 3. 0 6 7 8 9 10 6 means Chiron and Jupiter through Pluto totalling six.

The returned hits are placed in the memory (STO) cells in position 100 onwards in sets of 10, so the first hit starts in cell 100, next hit in 110, next in 120 etc. Cell 100 shows the total of hits. Each hit has the following format:

offset meaning

0 Number of hits left
1 Type (1=aspect, 2=signshift etc)
2 Day
3 Month
4 Year
5 A Planet no
6 Aspect number/leaving sign no/leaving house no/dir1
7 B Planet no./entering sign no/entering house no/dir2

"dir" means direction 1 for direct, -1 for retrograde. So if the second hit is: Mercury is turning retrograde, cell 115 will be 3 (Mercury), 116 will be 1 (was direct) and 117 will be -1 (but changed to retrograde).

Note: This code was introduced to produce a general output facility for aspect timing. It was abandoned again, because it performs very slowly, so it cannot compete with software having dedicated aspect schedule output. Anyway, the code is still available for experimental use.

The position indices used are:

Name	pla. no	index	used by	
Current	0.. 18	0	current	(earlier radixchart)
Radix	20.. 38	1	radix	(earlier current chart)
Current	40.. 58	2	current	(earlier auxchart1)
Aux1	60.. 78	3	xli only	(earlier auxchart2)
Aux2	80.. 98	4	xli only	(earlier presentchart)
Present	100..118	5	present	

HOUSE

Stack: X → HOUSE

Calculate one house cusp using the input menu values.

Please note, that if you change the menu values using the MEPUT code, you must also use DNORM to prepare the data correctly for HOUSE.

HPOS (house position)

Stack: X HPOS → (intang position of X)

This code gives the position of house X in intang units. You may convert this value to minutes of arc using the ITOM code.

The house numbers allowed are 1-12.
See also: PPOS

HRU (house ruler)

Stack: X HRU → (ruler)

Find ruler of house no X in latest chart or latest radix chart. For a listing of the house numbers allowed, see HSI. The ruler numbers are like RUL.

HSET

Stack: pos speed block house HSET →

Change calculated house position for a chart

pos: position to insert (intang)
speed: house speed to insert (intang)
block: 1=radix chart 2: current chart
house: house number (1-12)

See also: PSET

HSI (house in sign)

Stack: X HSI → (Sign occupied by house cusp X)

Find the sign on a house cusp, either in the latest chart or in the latest run radix chart. The numbers allowed are: IF

X:	Latest chart	Latest radix chart
1. House	1	21
2. House	2	22
3. House	3	23
4. House	4	24
5. House	5	25
6. House	6	26
7. House	7	27
8. House	8	28

9. House	9	29
10. House	10	30
11. House	11	31
12. House	12	32

HV

Stack: X HV → (speed of house X)

This code gives the speed of house X measured in intang units per day/year etc, that is, per the natural unit for the chart type in question. To convert to minutes of arc, use the ITOM code. In radix, transit and return charts, where the houses move very fast, you must add 360 degrees to get the speed correctly.

The house numbers allowed are listed under HSI.

See also PV PPOS HPOS

IF

Stack: A IF →

means that if A is zero, following code will stay inactive, until a matching ENDIF is met. If A is anything other than zero, the following code will work.

You must always have a matching number of IF's and ENDIF's. They may be "nested", for example IF.. IF... ENDIF... ENDIF. The "inner" IF and ENDIF are the matching pair.

You may well put an IF code in one paragraph, and the matching ENDIF in a later one. This will make the paragraphs inbetween active or inactive depending on the IF test.

See also: ENDIF ELSE

IN
IN~

Stack: X 0 P Q .. R IN → (1 or 0)

Member or group

Checks if X has one of the values P.....R. If so the result will be 1 (true) else 0 (false). The zero between X and the list is mandatory. The code is mainly used for testing if for

instance the sign occupied by a planet belongs to a certain group of signs, e.g. water signs, barren signs or whatever.

The above is equivalent to:

X P =
X Q = OR
....
X R = OR

The FP variant works on the FP stack

INC
INC~

Stack: X INC → X ?

(increment stack pointer)

This code restores the number last removed from the stack. To use this code correctly needs some stack knowledge. For example the code IF will use and remove the number on the stack, but INC may get it back again:

X Y IF → X
X Y IF INC → X Y

This may be useful if you need the IF condition for further purposes inside the IF-ENDIF construction.

INFIL (input line of data from file)

Stack: X INFIL → (data, data, data) result code

Description: Reads one line of string and or numeric data from a text file. This could be numeric data created by an external program for graphic or tabular presentation, which PCA could output through its screen or printer drivers. Or it could be birth data imported from a database or from another file format.

If X=1: Read one line. If the file is not open, it will be opened.

If X=0: close the file (if it is open).

The file name must be given as the first line in the text field. You may have only one input file open at a time. If the file is open and you state a different filename, the existing file will be used and the new name ignored.

The file is read sequentially.

A format string must be given as the second line in the text field. This will specify the line reading sequence. The data may be mixed numeric and string data. String data is put into the string array, and numeric data is pushed on the stack. Numeric data may not exceed the size of the stack (1024).

If you need mass data handling, you must create a loop reading one line at the time and use the data before the next line is read.

The result code is the last number pushed on the stack. It may have the following values:

-2 : Data read incomplete, data missing in line.
-1 : End of file reached
0 : File closed
1 : Successful read;

The line is read from left to right using a line read pointer.

The format string may consist of the following elements:

SPACE(s) : Skip any number of contiguous spaces in the data line, so that the line read pointer points to the first non-space.

#: read one number from the data line and put it on the stack. If the line read pointer does'nt point to a digit, it will be moved forward until it does, or the data line is exhausted. Negative numbers may be read as well. If the number exceeds 32767 or -32768 it will be set to these maximum (minimum) values.

~: read one FP number from the data line and put it on the FP stack. If the line read pointer does'nt point to a digit, it will be moved forward until it does, or the data line is exhausted. Negative numbers may be read as well.

@x read string of characters ending with character (x) from data line and place it in string array 0. X may be any character (!!except @ or number digits). If the string exceeds the available space in the string array, it will be truncated. !!X itself is not read.

@nnx read string of characters ending with character (x) from data line and place it in string array nn. X may be any character (except @). nn may be any number between 0 and 99.

INT~

FPstack: d INT~ e

Remove decimals from FP number and put result back on the FP stack. This code works also on negative numbers.

See also FLOOR

ITOF (convert integer value to floating point)

Stack: n ITOF →

This pops a value off the integer stack, converts it to a floating point value and puts it on the floating point stack.

See also FTOI

ISTKZ

Stack: n ISTKZ →

Resize integer stack to n. Default is 1024

See also: FSTOZ ISTOZ FSTKZ

ISTOZ

Stack: n ISTOZ →

Resize integer STO array to n

See also: ISTKZ FSTOZ FSTKZ

JD (julian date number)

Stack: JD → (JD 10000nds) (JD ones) (JD fraction)

This will take the menu values for date, time and zone and convert them to julian date number. If you change the menu values using the MEPUT code, you must also use DNORM to prepare the data correctly for JD.

The function returns three numbers even if julian day is just one figure. This is because the XLI interpreter handles short integers only, which would overflow trying to hold a JD.

The last part, (the "JD fraction") is the time of day expressed as a fraction.

To print the Julian date and time, you may use the following code:

```
JD
NUMS

JD: #### ####.####
```

This will work for positive Julian dates only, so do not move back before JD 0.

See also: GETTI SETTI

KEY

Stack: n KEY →

(simulate a keypress)

Normally the XLI interpreter is not meant to operate the user interface. Rather user interface is operating the interpreter. Otherwise a power struggle could arise.

There are however exceptions to this rule. The KEY code will simulate the user pressing a key with the ASCII value n. So 13 KEY will simulate pressing the ENTER key and therefore toggle output window focus. 32 KEY will simulate the SPACE bar and therefore toggle the data input dialog box on and off. Other codes are doing other specialized jobs like:

255	startxli;
254	TerminatePCA;
253	Window | fullsize
252	Window | tile
251	Window | cascade
250	Window | new
249	Window | close
245	show output screen
244	show command panel
243	start debug window
242	minimize Argus window
241	restore Argus window state
61	get data from database to input menu

KM

Stack: KM → distance in km

Distance in kilometer between current and radix position. The two locations should be found in the input data menu and the radix data menu.

The locations could be entered manually by getting data in the atlas, from the database or typed in by latitude and longitude. You may use the double-data options to get two input tabs and enter the two locations in each tab, or you may enter the first data, then run radix and enter the second data after that.

You could also enter the coordinates in XLI using MEPUT.

KUN (Kündig sections)

Stack: KUN → D M Y HH MM SS D M Y HH MM SS D M Y HH MM SS

For users of the Kündig rectification method, this code offers the possibility of some automatizing. From the data in the main menu, the three closest Kündig sections are calculated and the results put on the stack in chronological order. Each calculation includes Day, month, year, hours, minutes and seconds. The timezone, longitude and latitude are considered constant, as given in the menu.

What is meant by the "closest" section is: The second section will always be the closest to the given time, the two other are the closest on EACH side. So there will always be at least one section before and one section after the given time.

LOGX

Stack: LOGX →

Stop logfile output

See also: XLOG PROFL DEBUG

M2XLI

Used mostly to interpret a Macro with embedded XLI code. In an XLI file, you would use XLI functions directly, not put them into a macro.

Execute macro sting as XLI code

MACn

This is a complete series of codes calling calculations and charts. MACR calls the radix, MACT calls transit, MACW calls the bi-wheel etc. It is a replacement for the more clumsy CALL and then having a macro line. For long macros and for composite macros CALL is still the prefered method, while for a single command, MAC.. is more convenient

Special MAC commands:		Equivalent macro character
MAC.	Clear window	.

MACOF	Suppress all output
MACON	restore output
MAC*	Overwrite namefile position
MAC+	Insert current data in namefile
MAC<	Select radix data
MAC=	Fetch currently pointed data from namefile into input menu
MAC>	Select current data
MACA	Output aspects
MACB	Output solar arc chart
MACD	Output day chart
MACE	Output Tertiary chart
MACF	Output Minor progressed chart
MACG	Output Composite chart
MACH	Output Relationship chart
MACL	Output Lunar return chart
MACP	Output secondary progressed chart
MACQ	Quit Argus
MACR	Output radix chart
MACRG	Call program registration box
MACS	Output Solar return chart
MACT	Output transit chart
MACU	Start the clock chart
MACV	Draw chartwheel
MACW	Draw bi-wheel
MACv	Draw no-house chartwheel
MACw	Draw no-house bi-wheel

MAX
XMA~

Stack: X Y MAX → (X or Y whichever is the largest)

Takes two numbers off the stack and pushes the largest one back

The FP version does the same on the FP stack.

MEGET (Menu Get)

Stack: X MEGET → (menu value)

The PCA input menu consists of a number of integers representing the inputted data. With this code and the corresponding MEPUT, you may manipulate data in the input menu. The menu values are:

0 MEGET → Date, Day

1 MEGET → Date, Month
2 MEGET → Date, Year
3 MEGET → Date, 0=AD 1=BC
4 MEGET → Time, hours
5 MEGET → Time, minutes
6 MEGET → Time, seconds
7 MEGET → Sex, 0=male 1=female neutral=2 Horary=3 event=4 Country=5
8 MEGET → Timezone, hours
9 MEGET → Timezone, minutes
10 MEGET → (always zero)
11 MEGET → Timezone, east=0 west=1
12 MEGET → Geo latitude, degrees
13 MEGET → Geo latitude, minutes
14 MEGET → (not used)
15 MEGET → Geo latitude, north=0 south=1
16 MEGET → Geo longitude, degrees
17 MEGET → Geo longitude, minutes
18 MEGET → (not used)
19 MEGET → Geo longitude, east=0 west=1

The shown parameter values work on the "current" data-input data set. You may also access the "radix" and "present" data input values by adding 20 or 40: 20-39 will access the radix data and 40-59 will access the present data, for example

22 MEGET → Radix data - year
44 MEGET → Current hour

See also: MEPUT GETTI SETTI GETZO SETZO

MENAM

n MENAM (insert name in string n)
-n MENAM (insert string n in the name input field)

This is similar to the NAME code, but the name is taken from the current input menu, while NAME takes it from the current saved chart. So if the user enters new data, NAME will not fetch the new name until a chart is calculated, while MENAM will. MENAM is equivalent to MEGET. If n is negative, the name is moved from the string n back to the input menu.

See also: NAME BDATA NPR BDPR

MENU

Stack: MENU →

This code in combination with the code OPT lets you setup a very simple menu where you can choose between some options by pressing a key on the keyboard (no mouse).

First you set up a paragraph having just the code MENU in it code part.

The text part should be set up to show which keys you can use, and what you can expect from pressing them. The text part is just information.

The following paragraph must hold the code OPT to stop the program waiting for your keypress.

It is possible to insert stored values and text strings into the menu using the NUMS code (see this). If you do this, you should put NUMS first, then the MENU code.

See also: OPT

MENUX (XLI defined dialog box)

This code sets up a dialog box with buttons, edit fields, labels, checkboxes and radiobuttons as required. When the user closes the box, the return value is pushed on the stack, and any other values or entered strings are transferred to where they were taken from. Pressing ESC (or clicking a cancel button if present) will leave old values as they were.

You may edit strings in the XLI string list, but you cannot edit numbers in the STO array.

MENUX takes a large number of arguments to define its controls. The format is:

kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
.....
n width height caption
MENUX

The first n lines define one control each.

The last line before MENUX define the number of controls and the size and caption of the dialog box itself. The dialog box will always appear on the screen center.

The arguments for each control are:

Kind:

- 255 Edit box
- 254 Label
- 253 Checkbox
- 252 Radiobutton
- 240..251 reserved, not used

0..239 Button

Buttons kind-numbers must be different to distinguish which button has been pressed. This number will be the one which is pushed on the stack when the menu box is closed. If the kind number is nonzero, clicking that button will cause the dialogbox to close.

Left, top, height, width:
Placements and dimensions of the control relative to the dialog box measured in pixels.

Key:
Button: if set to 1 this means, that pressing the ENTER key will will be the equivalent to clicking this button. If set to 2 this means, that pressing the ESC key will have same effect as clicking this button. Only one button may have key 1 and only one button may have key 2.

Radiobutton or checkbox: STO cell number from where to get or put the button status: nonzero=on, zero=off.

Other controls: no effect.

valueindex: This is the XLI string number from which a text is taken to use as:

Button: Caption
Editbox: Default input text
Label: Label caption
Checkbox: Checkbox caption
Radiobutton: Radiobutton caption

Reserved:
not used, could be set to zero. In future PCA version, other arguments may be needed, so these three position are reserved for this, so that it will not be necessary to change the format and cause backwards incompatibility.

The last definition line holding information about the dialog box itself has these arguments:

n:
The number of controls, i.e. the number of lines above to use. Note, that there may be a restriction on the size of n accoring to the size of the XLI stack. Each line holds 10 values, so the number of controls allowed is (stacksize-4)/10. The stacksize is currently set to 1024 allowing for 100 controls.

width,height:
Width and height of dialog box. It may take a bit of experimenting to get this right and place the controls to look right.

caption:
The index of the XLI string list to use as text on the dialog box's caption bar.

Functionality:

When the menu definition is set up, and the XLI meets the MENUX code, it will create the dialog box according to the above definitions. The user can then press buttons, edit editfields etc as needed.

To close the box, the user must click a button, which has kind neither zero or 2 Or by pressing ENTER if any button has kind<>0 and <>2 and key=1.

If the box is closed with a button of kind<>2 (not an ESC-button), the values in any editbox will be transferred back to the string list item it was taken from (defined by the ix argument). If checkboxes or radiobuttons are present, the result value (1 or 0) will be transferred back to the STO cell, from which their initial value was taken (defined by the "key" argument).

To escape the box without making any changes, it can be closed using the closebox on the caption bar, by pressing ALT-F4 or by pressing a key whose kind<>1 or by pressing ESC and a key has kind=2 and key=2. Such a key should be labeled "CANCEL"

If any character in the range 128..191 is used for labels, buttons etc. the ARGUS font is used, so it is possible to insert planet symbols etc.

menubox test example 1;

```
$
1 CARY  ;Sun is written with & and letters, to show that ALT-U can be used

,S&un,,,ƒ,„,…,†,‡,ˆ,‰,Š,Cancel
$
3  10  50  50 20 0  1 0 0 0   ;Button (1 and 2 are reserved for OK and CANCEL)
4  10  75  50 20 0  2 0 0 0   ;Button
5  10 100  50 20 0  3 0 0 0   ;Button
6  10 125  50 20 0  4 0 0 0   ;Button
7  10 150  50 20 0  5 0 0 0   ;Button
8  10 175  50 20 0  6 0 0 0   ;Button
9  10 200  50 20 0  7 0 0 0   ;Button
10  10 225  50 20 0  8 0 0 0   ;Button
11  10 250  50 20 0  9 0 0 0   ;Button
12  10 275  50 20 0 10 0 0 0   ;Button
2  10 310  70 30 2 11 0 0 0   ;CANCEL-button
11 350 350 0 MENUX 2 SUB
NUMS

ModalResult= #######
$$$
```

;menubox test example 2;

```
$
1 11 STO
0 12 STO
0 13 STO
1 14 STO
2 CARY

TEST XLI-DIALOG BOX,STO 11,STO 12,STO 13,STO 14
String 6,String 6 to edit,OK,CANCEL
$
253  10  25 150 20 11 1 0 0 0 ;checkbox
253  10  50 150 20 12 2 0 0 0 ;checkbox
252 170  25 150 20 13 3 0 0 0 ;radiobutton
252 170  50 150 20 14 4 0 0 0 ;radiobutton
254  10  75 150 20 0  5 0 0 0 ;label
255 100  75 200 20 0  6 0 0 0 ;editbox
1  50 125 100 30 1  7 0 0 0 ;OK-button
2 200 125 100 30 2  8 0 0 0 ;CANCEL-button
8 350 200 0 MENUX
14 RCL 13 RCL 12 RCL 11 RCL
6
NUMS

String 6 after edit: @@@@@@@@@@@@@@@@@@@@@@@@@@@@

STO values:                    11                    12
13                    14
   ####          ####          ####          ####

ModalResult= #######
$$$
```

See also: MENU OPT

MEPUT (Menu Put)

Stack: X Y MEPUT →

The value X is forced into menuvalue number Y. Y can have the following vaues:

0..19	current data
20..39	radix data
40..59	present (clock) data
60..79	data of calculated radix chart
80..99	data of calculated current chart
100..119	data of calculated present chart

However, you will in most cases use 0..19 or maybe 20..39

See code MEGET for an explanation of the menuvalues and index.

If you change date and time, you should use the code DNORM as well to normalise the date if out of range, and to make the system accept it as the "current time" for a number of functions.

See also: MEGET GETTI SETTI GETZO SETZO

MIN
MIN~

Stack: X Y MIN → (X or Y whichever is the smallest)

Takes two numbers off the stack and pushes the smallest one back

The FP version works on the FP stack

See also: MAX MAX~

MOD

Stack: X Y MOD → (X+Y) mod Y

The modulus returns the remainder, when dividing X with Y. In Argus modulus of negative numbers is always positive. For example, if you have an angle of -35 ° modulus 360, for astrological needs, you want the result 325°, not -35, which would be the traditional result.

See also: FPMOD

MONS monospaced

1 MONS changes output to monospaced, meaning that all characters are forced into position, even if a proportional font is chosen. This may cause wide proportional characters to overlap.

0 MONS changes back to using the fonts own character spacing.

See also: OEM VBTIM

MOVCH move chart data around

When calculating charts, all chart information is saved in a data structure. There are 5 such structures:

Name pla. no index used by

,

Current	0.. 18	0	current
Radix	20.. 38	1	radix
Current	40.. 58	2	current
Aux1	60.. 78	3	xli only
Aux2	80.. 98	4	xli only
Present	100..118	5	present

So when any chart is calculated, its data is saved in structure "current", except the horary clock, which uses its own structure (present). Calculating a radix, will be saved in both "Current" and "Radix". Later calculations of chartwheels, aspects, midpoints etc are using these data.

With MOVCH you can move data from one group to another. For instance 1 3 MOVCH will move the data from Current into Aux2 overwriting what might be there. The following example demonstrates this by swapping the radix and the current data, so that calling the bi-wheel will have radix positions in the outer wheel and current positions in the inner:

```
0 3 MOVCH      ;move current to aux1
1 0 MOVCH      ;move radix to current
3 1 MOVCH  ;move aux1 (former current) to radix
MACW       ;call macro bi-wheel
```

The two aux charts are for temporary storage. However, you may access the positions etc using expanded indexes. It was always possible to use e.g. 5 PPOS to get current Mars, and 25 PPOS to get radix Mars. But now you can get Aux1 Mars writing 45 PPOS, Present Mars writing 85 PPOS etc. This applies to all the codes, where you normally would add 20 to get the radix data.

MOVIE

Stack: MOVIE →

Start live chart

MUL
MUL~

Stack: X Y MUL → X*Y

Multiplies two numbers.

The FP version works on the FP stack

See also: ADD SUB MUL DIV ADD~ SUB~ MUL~ DIV~

MULT

Stack X Y Z MULT → X * (Y/Z)

Multiply by fraction

This arithmetic routine will multiply X and Y as longints to avoid overflow, and then divide by Z. The result will be converted back to short integer which may or may not produce an overflow, but this is the responsibility of the programmer.

The routine is useful for scaling coordinates.

See also MUL MUL~

NAME

Stack: X NAME →

Insert the current name into string array number X.

See also: MENAM

NAX

Stack: NAX →

This code gets notes (if any) from the selected data in the namefile into the string array. The lines of notes are placed in string[0] and onwards.

Warning: This code clears the string array for previous data. For better control, you should use the more recent code NOTE

See also: NOTE

NDATE

Stack: X NDATE → (year) (month) (day)

This is a code specialised for interpretation of graphic transits. When using the code NTA, you will get start and end time for an aspect. These times are given as sector numbers. This code translates the sector number (X) to day, month and year.

Example:

```
$
NTA XY NDATE 4 FETCH NDATE NUMS

From: ## ## ####   to: ## ## ####
$
```

The dates will be approximate, if the time sectors are more than 1 day. The yearly transits use 6 day sectors (intervals), whereas the monthly transits use half-day intervals.

NEXT

Stack: NEXT →

This code ends a FOR loop. There must always be a corresponding number of FOR's and NEXT's.

See also: FOR XIF

NFI

Stack: X NFI → Y

Move data from namefile entry no. X to the input menu.

The result (Y) should normally be zero unless X points beyond the last entry or the namefile entry X has a blank name. In that case Y will be the size of the namefile. The namefile pointer itself will not be moved.

The function may be used to find the size of the namefile:

```
1000 NFI
```

You may also use the function to selectively fetch a specified entry or series of entries.

If you insert a series of events into the namefile and terminate with a blank name, you may let the program run a series of progressions for these entries and stop when the blank is met:

```
$
0 NPNT 1 DUP NPNT DEC 1000 FOR    ;count from current entry
1 CNT NFI                         ;get entry
XIF                               ;exit loop if blank
CALL                              ;call progressed chart

P
```

$
NEXT

$$$

See also: NPNT MAC= MAC+ NAX

NFN (next file name)

Stack: X NFN

File branching. If a module consist of more than one file, which is the case in most interpretations, you normally put the name of the next file in the chain after the three $$$-signs at the end of the file.

But you may put more than one filename, using one line for each, and use the NFN code to select which file to branch to. 1 NFN (default) will branch to the first file, 2 NFN to the filename on the second line after the $$$ signs etc. 0 NFN will exit the XLI module and the same will for instance 5 NFN if there are less than five filenames in the list.

The NFN code may appear in any paragraph in the file, the value will be remembered when the $$$-signs are met.

NOT
NOT~

Stack: X NOT → not X

Checking if a condition is false. For example to check if X is "greater than or equal to 5" is the same as "not less than 5", and you may code like this:

X 5 > NOT

The codes NOT > < or = produce a boolean value. 1 if true and 0 if false. Not will reverse this so 0 NOT will produce a 1 (true) and 1 NOT a 0 (false).

Actually will NOT convert any nonzero value to 0.

You may use any boolean result as an integer. Sometimes you can e.g. multiply a value by the result of NOT or = instead of using IF .... ENDIF. This may simplify the coding, but being less obvious to read, you may find it bad programming style:

```
1 DUP 0 > IF DEC 0 ENDIF        ;if negative change to zero - traditional
1 DUP 0 > MUL                   ;same using the boolean result directly
```

The FP version works on the FP stack.

See also BOO = < > IF

NOTE

Stack: line i NOTE

Get currently pointed namefile entry notes line

line: is the line in the notes area (if any) of currently pointed namefile entry
i: is the string index to copy the line to

If no notes present or line is out of range, nothing is copied
if line is negative the number of lines is pushed on stack
if line is -2 then all lines are copied to string i separated by a CRLF
if line is -3 then all lines are copied to string 0, 1, 2 onwards (max i)

See also: NAX

NPNT

Stack: X NPNT → Y

This code changes the namefile pointer and hands back the former pointer position. If the pointer points to entry 13 and you write:

0 NPNT

the result will be 13 and the namefile pointer will be changed to entry 0.

If you just wish to know the pointer position without changing it, you must write:

0 NPNT 1 DUP NPNT DEC

You cannot set the pointer outside the size of the namefile. A negative number will put the pointer to zero and if you put a number greater than the namefile size, the pointer will be put at the end of the file.

NPR (birth name)

Stack: NPR →

Output name in the current input menu using one line.

NPUT

Stack: X NPUT →

Inserts a name into the PCA input menu taken from string array number X.

NTA
NTAX

Stack: NTA → S E T R A
Stack: NTAS → S E T R A

where S = startsector
E = endsector
T = transitplanet
R = radixplanet
A = aspect number

This is a code specialised for interpretation of graphic transits. After running graphics transits / progressions, a list of aspects found is saved in a special array, where they can be retrieved with NTA.

It will fetch the next aspect in the list of transit aspects. The aspects are those calculated in the latest graphic transit calculation. If the list is exhausted (no more "next" aspects) S will be 0. This also applies if no graphic transits have been calculated yet, since PCA was started.

The sector numbers S and E will be a number between 1 and 61. If you have setup your own graphic transits using the code GTR the range may be different, starting with 1 and ending with the number of timeslices defined.

The transit planet (T) will be a number 1-11 (Sun-Node).

The radix planet (R) will be a number 20-38.

The aspect number (A) will be a number 1-9.

For explanation of planet numbers, see code PSI.
For explanation of aspect numbers, see code ANUM.

The two codes NTA and NTAX works the same, but NTA filters off Chiron aspects, NTAX does not.

See also: RTA

NTOS

Stack: n i NTOS →

Convert number n to string and place it in string array no i.

NUMS

Stack: X Y Z .... NUMS → X Y Z .....

This code lets you insert variable numbers and strings into the text field. A paragraph, whose coding part contains the NUMS code will always be written, it does not matter if the top of stack is 1 or 0.

The text field should contain "templates" that is fields of #######'s (for integer numbers), ~~~~~~.~~~ (for floating point numbers) or @@@@@@@'s for text, which will be replaced by the numbers or text when output.

The number of arguments (X Y Z .....) depends on the number of values or strings you wish to print, and must match the number of templates in the text field.

Reverse order: Arguments must be put on the stack in reverse order, that is, the first value to display must be put on the stack as the last.

An example: The following coding will display the values in memory cells one and two:

```
$
2 RCL 1 RCL

Value in memory cell 1: #######
Value in memory cell 2: #######
$
```

To print strings, these must be defined in the string array. The matching number put on the stack must be the number of the string in the string array. For example to print the sun sign in the latest chart, try this:

```
$
1 CARY

,ari,tau,gem,cnc,leo,vir,lib,sco,sgr,cap,aqr,psc
$
```

1 PSI NUMS

The sun sign is: @@@
$

If the number on the stack is negative, the string will be looked up in the system strings. In the older versions of PCA (the predecessor of Argus), the language definitions were part of the system strings. Now they are in a separate file, so you cannot make NUMS read them. But they can be read into strings with SYSTR and then used with NUMS.

For example
```
        $ 501 1 SYSTR 1  NUMS  (@@@)
```
will print Jan

Planet, sign and aspect names also have negative numbers, but for ease of access, there are the codes PNAME, SNAME and ANAME which will translate them, so for example to print the Moon symbol (or the string Moo dependant on system settings):

2 PNAME NUMS

@@@

Text, string- and number templates may be mixed freely in the text part. the text templates may be any length. Numbers will be written in the right side of the template and strings in the left side.

Short templates: If a number template is too short to hold the number, the number will have only the last digits displayed. To display a number with leading zeroes, you may add a bigger power of 10 and make the template too short to hold the leading 1. for example:

2 PDEG 1000 ADD NUMS

###

If the Moon is in e.g. 13 Aries this will print 013

Short string templates: The string will miss its last part. With the PAD code you have options to expand or shrink the template on the go depending on string length.

Text and number templates may touch each other for example: ##@@## to make the compact planetary position representation: 28ta51 for 28 51 Tau.

If you wish two templates of equal type to touch, for example to merge two strings, you must use the line continuation mark "\" and break the line. For example the julian day example, where a long number is mixed of two short ones:

```
$
JD
NUMS
```

JD: ####\
####.####
$

This will print e.g. `2449054.9102`.

The NUMS code is extremely useful for exporting data!

Note, that the NUMS code not only works with output but also with lines used for arguments to other functions including: STDEF, CARY, CALL, CML, EXEC and FONTS

See also: PAD

OEM

In the old DOS days the available character set was different, Especially language characters were assigned differently and the DOS set had some nice grid characters to make tables.

To keep compatibility and to make some DOS features available, Argus comes with a special font (pca-oem.ttf) mapped in DOS style and with the gric characters.

1 OEM will make Argus try to emulate the DOS character set, which means that if your module is using accented or other national characters, they will be translated from the DOS set to fit windows ANSI standard. Also the frame characters will be emulated, so that it is possible to draw grids etc using the IBM/DOS framing characters (only the single line ones). To switch it off, use 0 OEM.

PLEASE NOTE: The MONO setting will override the OEM, so do not use both. OEM is itself monospaced.

OPT

Stack: 0 X Y...Z OPT → (key pressed)

As described for the MENU code, you must use OPT to wait for keypresses.

Together with OPT, you define wich keypressed to wait for. You MUST (as shown) always start with a zero, then X, Y etc will be the different keys you may press. The key numbers are 65 for A, 66 for B etc. which in fact are the ASCII codes for the symbol pressed. OPT is not case sensitive, so e.g. 65 will work for both capital and lowercase "A".

You may put no more than 31 possible keypresses.

The result of OPT is a number between 1 and N, where N is the number of possible keypresses you have defined. For example:

0 88 65 66 OPT

will produce 1 if key "A" is pressed, 2 if key "B" and 3 if key "X" (88) is pressed. If you click the x top-right corner of the dialog, it will exit with value 2

The result may be used for example with code NFN to determine which file to branch to, or it may be stored and tested with IF to decide which instructions to do.

See also: MENU MENUX

OR
OR~

FP OR 1.0 if round(x) or round(y) <>0

Stack: X Y OR → X OR Y

Checks if either of two conditions are true. This means that:

1 1 OR → 1
1 0 OR → 1
0 1 OR → 1
0 0 OR → 0

1 represents "true" and 0 "false".

The FP version works on the FP stack, it first rounds the two arguments, and after or-ing them converting them to FP and pushes them back on the FP stack.

Technical note: if AND, OR is used on other numbers than 1 and 0 the result will be a "bitwise" comparison. To be able to analyse the result, you will have to convert the numbers to binary, i.e. each 16 ones and zeroes, and compare each set of bits separately. This may be used creatively if you are experienced in this field. If not, better be sure, that you use the function on zeroes and ones only.

See also: AND XOR BOO IF

ORGIN

x y ORGIN (displace graphics origin)

You can move the origin for succedent drawing, so that succedent drawing will use the new x,y ase.g. 0,0 is the lower left corner, or if for example you are drawing a group of things at a special location, then it may be an advantage to replace the origin.

This means, that you can change a whole group of graphics elements just by changing the origin, rather than changing all the x an y's in the drawing. X and Y used in the ORGIN

command are always relative to the original origin in the center, so it is independant of what previous ORGIN commands were doing.

The X and Y scaling is the units defined by the SCALE code used before GRON, or if no SCALE command has been issued, the units are +- 1024.

See also: SCALE ROT RSIZE GPUSH GPOP

PAD

n PAD

When printing strings using the NUMS code and a @@@@@@@ template, the string will normally be shorter than the template. If the interpreter is in padding mode, the remaining part of the template will be filled with spaces so that lines remain the same length independant of the string length. In non padding mode the template will be truncated to fit the string. 1 PAD (default) will put XLI in padding mode and 0 PAD will put it in truncate mode. Note, that the code SYN also puts XLI in truncate mode.

Example: @@@@@@@@@@@ and @@@@@@@@@@@ are friends

0 truncate mode: Tom and Jennifer are friends

1 padding mode: Tom and Jennifer are friends

2 tab mode: inserts tabs instead of spaces. ARGUS uses tabs to produce e semi-monospaced print.

3 expand template to hold the full length of the string, you may use just a single @ for the template.

PASSW

Stack: s q PASSW

Encrypt string s with key q

encrypt password string. The encryption algoritm is non-standard.

PDEG (planet degrees)

Stack: X PDEG → (degrees 0-360)

This code finds a planet's position in degrees. So if the Moon for instance is in 3-51 Sagittarius, the coding:

2 PDEG

will produce the value 243. The minutes of arc are just removed, not rounded up, so any position between 3-00 and 3-59 Sgr will result in the value 243.

This code may be used for "Degree astrology" for instance "Sabian symbols", or to find decanates, for example:

4 PDEG 30 MOD 10 DIV

will produce following results for Venus in:

First decanate (any sign): 0
Second decanate (any sign): 1
Third decanate (any sign): 2

The planet numbers allowed are listed under PSI.

PENC

c PENC

000 PENC makes the graphic pen black, 888 PENC makes it white, 800 PENC makes it red etc.

The colour numbers in ARGUS are 000 to 888, so use only digits 0-8, e.g. 699 is not allowed. Color numbers with digits 9 (9xx, x9x or xx9) are undefined unless you redefined them with DFCOL

The default pen color is black (000)

See also: PENW DFCOL COL BRCOL GRCOL TXCOL

PENUP

Stack: PENUP →

Graphics lift pen. Think of the line drawing capability as moving a pen, that may be either lifted over the paper, thus not drawing or be down touching the paper as it moves, thus drawing a line. PENUP will lift the pen, so that the next following DRAW will just move the pen to the new position without drawing.

PENW

w PENW

Set penwidth to w in user coordinates. So 2048 penw will make following draw commands draw lines filling the whole plotting area.

Pen width defaults to 1.

See also PENC

PFILE

Stack: PFILE →

show file content in output window

The file name must be placed in the text part. If the file is not found, nothing is output.

PHS (planet in house)

Stack: p PHS → (house position of planet p)

Finds the house position of planet X. The house position is in the planets own chart, not crossreferenced to latest radix (as with the ordinary PCA chart printouts). The planet numbers are listed under PSI. The result is a house number between 1 and 12.

To find cross-referenced houses, use code BHUS instead

The planet numbers allowed are listed under PSI.

See also: BHUS

PLA (planet)

Stack: p PLA → (speed) (position)

This code calculates a single planetary position and speed using the information in the input menu. p must be in the range 1-12. The values for position and speed are intang units.

Note, that if you change the menu values using the MEPUT code, you shouold also use DNORM to prepare the data correctly for PLA.

PLIN

PLIN is obsolete and does nothing. Was used in DOS PCA for printing information.

PLACE

Stack: X PLACE →

Insert the current input menu city name (if available) into string number X.

See also GETGL

PMIS

Stack: s PMIS → 1 or 0

s is the string number holding a lookup string. The result is 1 if the string is part of the system string features as inserted in the registration data.

Lookup if given string is contained in features

PNAME

Stack: p s PNAME →

Converts an planet number p to a negative system string number for use with NUMS. PNAME is the equivalent of writing: 540 ADD CHS

See also ANAME, SNAME

POLY

Draw polygon using the current pen and brush colours. To draw a polygon, you must first provide the coordinates for the corners. This is done the same way as drawing, that is e.g.

```
$
GRON
8 PENC
80 BRCOL
PENUP
-200 -200 DRAW
-200 200 DRAW
200 200 DRAW
200 -200 DRAW
POLY
```

$$$

So instead of terminating a shape with PENUP, you use POLY. POLY will automatically close the shape, if the last point is not the same as the starting point.

The brush color will fill the polygon, a shape drawn with PENUP will not get filled.

See also PENUP

PPATH

n PPATH (get Argus program path)

Loads the program path into string array[n]
The result should normally be:

```
C:\argus4\
```

see also XPATH

PPOS (planets position)

Stack: X PPOS → (intang position of X)

This code gives the position of planet X in intang units. You may convert this value to minutes of arc using the ITOM code.

The planet numbers allowed are listed under PSI.

PROC

n PROC (define a subroutine number n)

You may use subroutines in XLI. They are identified by a single number (n) of your own choice (but not by a name). To define a subroutine, you must place it somewhere in your code BEFORE it is used. This is because the XLI interpreter must do one scan (dummy run) first, which does nothing but make note of the address.

It is perfectly legal to place the subroutine in another file than where it is going to be called from.

Each subroutine must be defined with an individual n, else your calls will be ambiguous.

After the PROC must follow the actual code of the subroutine. This can be several paragraphs, or even several chained files, and must be terminated by the RETN code. Without the RETN code, your XLI program will just scan forever, never execute.

Soubroutines in XLI cannot be used recursively.

See also: SUBR RETN

PROFL

Stack: b PROFL

Start (b=1) or stop (b=0) a profile timer. Profiling lets you check how long it takes for a certain code sequence to execute.

Argus has only one profile timer available.

When stopped (b=0) a line will be written on the output screen showing the time elapsed since the timer was started. The timer is not reset, when the module terminates.

See also: LOGX XLOG DEBUG

PSET

Stack: pos speed block planet PSET →

Change calculated planet position for a chart

pos: position to insert (intang)
speed: planet speed to insert (intang)
block: 1=radix chart 2: current chart
planet: planet number (0-18)

See also: HSET

PSI (planet in sign)

Stack: X PSI → (sign number of planet X)

Get the sign occupied by planet X in the latest calculated (any) chart or latest calculated Radix chart.

Planet numbers are:

Latest	curr	radix	curr	aux1	aux2	pres	aux3
Chiron	0	20	40	60	80	100	120

Sun	1	21	41	60	80	101	121
Moon	2	22	42	62	82	102	122
Mercury	3	23	43	63	83	103	123
Venus	4	24	44	64	84	104	124
Mars	5	25	45	65	85	105	125
Jupiter	6	26	46	66	86	106	126
Saturn	7	27	47	67	87	107	127
Uranus	8	28	48	68	88	108	128
Neptune	9	29	49	69	89	109	129
Pluto	10	30	50	70	90	110	130
Node	11	31	51	71	91	111	131
Part of fortune	12	32	52	72	92	112	132
MC	13	33	53	73	93	113	133
ASC	14	34	54	74	94	114	134
11. house	15	35	55	75	95	115	135
12. house	16	36	56	76	96	116	136
2.house	17	37	57	77	97	117	137
3. house	18	38	58	78	98	118	138

The result is a sign number (1-12)

See also HSI HPOS PPOS

PSTAT

Stack: X PSTAT →

PSTAT controls the output suppression much the same as MACON and MACOF. It is a leftover from the DOS PCA which neede more control of output to screen and printer.

0 PSTAT: turns output off
1 PSTAT: turns output on
8 PSTAT: returns 1 if ointput is turnted on 0 if not

See also: MACON MACOF

PTMP (parameter change temporarily)

Stack: PTMP →

If you change the PCA settings using XLI, you may assure, that the changes are temporary just while the XLI module is running, and will be set back to the old values, as soon as the module has finished. This may be a special colour setup, whatever else you can do with the XLI configuration codes.

PTMP will also restore the menudata. I for instance you have calculated a series of charts by changing the current data using MEPUT it will automatically be restored to the values it had when code PTMP was executed. So normally, you would place PTMP at the very start of your code. The restore will happen at the time, when all macros and XLI files are exhausted. If you wish to restore earlier, you should use the code TIDY.

The system variables are saved in a temporary file called SYSVARS.$$$.

If a module produces an error and exits, it may not tidy up properly. Trying to run the module again or another module using PTMP may then give an error "cannot create SYSVARS.$$$" (because it already exists). To fix this, try to run an empty XLI macro just with one $ sign.

see also TIDY

PTR

Polar to rectangular conversion of coordinates. If you need to draw circles or other shapes, where you would normally need to calculate sines and cosines, this code will do the conversion for you.

The polar coordinates are

V: the angle measured counter-clockwise in intang
R: radius (integer)

So: V R PTR → X Y

Example: draw a circle radius 700 in steps of 10 degrees, which is approx 1820 intangs:

```
$
GRON
0 34 FOR
1 CNT 1820 MUL 700 PTR DRAW
NEXT
PENUP

$$$
```

There is no integer code reversing from rectangular to polar coordinates,

See also: FPPTR FPRTP

PUT

Stack: X Y PUT →

Writes one value into the stack, overwriting the value already there. The position is offset from the ZZO marker, which MUST have been set before at some point. If the ZZO marker is not set, you risk to crash the program or produce unreliable results.

Examples:

ZZO 3 7 2 11 0 PUT → 11 7 2
ZZO 3 7 2 11 1 PUT → 3 11 2
ZZO 3 7 2 11 2 PUT → 3 7 11

See also: GET ZZO

PV

Stack: X PV → (speed of planet X)

This code gives the speed of planet X measured in intang units per day/year etc, that is, per the natural unit for the chart type in question. It may be negative, if the planet is retrograde. To convert to minutes of arc, use the ITOM code.

For house cusps and the part of fortune in radix and transit charts which runs very fast, you must add 360 degrees to get the speed correctly.

This code may be used to calculate the orbspeed of an aspect:

```
X PPOS Y PPOS SUB AZP XY 12 MUL
X PV   Y PV   SUB DIV
```

This coding will result in the number of months an aspect is from its exact value, negative is separating positive if applying. Note however the following problems:

1. If X and Y runs equal speeds, you will get a division by zero error.

2. If one of the planets is a radix, you should take only the speed of the moving planet.

3. If one of the "planets" is a transit angle or a solar or lunar return, the speed will be more than one revolution a day, and the formula will fail.

The planet numbers allowed are listed under PSI.

See also: HV PPOS HPOS

RCL
RCL~
NRCL~

(recall)

Stack: X RCL → (value from memory cell X)

Fetch a value from memory cell number X. The memory cells are not initialised, so you have to put something there using code STO before it will have any meaning to read it back using RCL.

The integer storage has 3000 positions, so X must be in the range 0-3000. You may however resize the range using the code ISTOZ.

The FP version RCL~ has only one storage cell and will place this on the FP stack
The NRCL~ has one argument and 32 positions (resizeable with code FSTOZ)

examples 4.0 NRCL~ will place contents of FP sto cell 4 on the FP stack.
4.0 RCL~ will place the contents of the single storage on the FP stack

See also: STO STO~ NSTO~ FSTOZ ISTOZ

RE2SP
SP2RE

(FP rectangular to spherical coordinates and reverse)

FPStack: x y z RE2SP → h v r

r v h SP2RE → z y x

See also: SP2RE ROT ROTX ROTY ROTZ

RECH (recent chart)

Stack: RECH → (chart type)

This code will tell which type the "latest calculated chart" is, e.g. Radix, Progressed, Solar arc or whatever. It may be useful to print a heading or to branch to a certain interpretation.

It may also be useful to condition your output, which may be different depending on chart type. For example if it is a non-radix, you may want to output extra information about the relationship to the calculated radix.

The chart type numbers are the ASCII value for the key you press to calculate it. For the ones called with the X (extra) prefix for example month transits (XM) use lower case value (m=109). Here is the complete list:

82 R Radix

80 P Progressed
84 T Synastry (Transit as second chart)
76 L Lunar return
83 S Solar return
69 E Tertiary 1
70 F Tertiary 2
66 B Solar arc direction
71 G Composite
72 H Relationship
68 D Day chart

109 m Month graphic transit
121 y Year graphic transit
97 a 5 year graphic progressions
98   b 60 year graphic progressions
77   M 1 month graphic collective transits
89   Y 1 year graphic collective transits

Having the RECH number, you may want to get the name of the chart type:

For example
RECH 601 ADD 1 SYSTR
will place the name of the the recent chart in string 1

REF

1 REF switches referenced interpretation on, 0 REF swtiches it off.

RETN (return from subroutine)

This code terminates your subroutine. When the subroutine is scanned, this code means: "end of scan", while when it is actually later executed, this code means "return to calling point".

See also: SUBR PROC

REVNO

Stack: REVNO→ Argus revision number

RND

Stack: X RND → random(X)

Produces an integer random number between 0 (including) and X (not including).

ROT (2D) rotate

FP stack: x y v ROT → v y x

Given x,y coordinates, rotate coordinate system counterclockwise by angle v to get new coordinates.

You could consider the task the same as rotating point x,y by angle v clockwise around the origin (0.0)

See also ROTAT ROTX ROTY

ROTAT (rotat)

Stack: v ROTAT →

Change the rotation of future DRAW commands, so that they will get rotated counterclockwise around current origin by angle v (intang units).

See also: GPUSH GPOP GCLR DRAW PENUP POLY

ROTX (3D)

FP stack: R V H rot ROTX → h v r

With spherical coordinates rotate coordinate system around X axis (clockwise as seen from x) to get changed coordinates.

R spherical coordinate radius
V spherical coordinate angle degrees
H spherical coordinate height angle degrees
rot angle to rotate (0-360°)

result is new h v r coordinates

See also ROTY ROTZ ROT

ROTY Rotate equivalent to ROTX

ROTZ Rotate equivalent to ROTX

RSIZE

Stack: n RSIZE

n is a percentage. n=0 will plot everything in a dot at the origin. n=200 will plot everything double sized and possibly be cropped.

You may make the following graphics draw smaller or bigger using this code. n is the percent magnification. For instance, the chartwheel will normally print in a +- 1024 square. To make it print half size, you use 50 RSIZE first.

The area taken up on the screen remain unchanged. Only the content is resized.

Se also SCALE

RTA

Stack: RTA →

This is a code specialised for interpretation of graphic transits. It will reset the aspect counter to the first in the sorted order. See also the code NTA.

See also: NTA NTAX RTA

RUL

Stack: X RUL → (ruler)

Find the ruler of sign number X. The rulers are the "modern" ones including Uranus, Neptune and Pluto. So for example: 8 RUL will produce 10 (Pluto) which is the ruler of Scorpio (sign 8).

If you need "old" rulers, where e.g. Saturn rules Aquarius, you could set up a table:

ZZO 0 5 4 3 2 1 3 4 5 6 7 7 6

Now s GET will give you the old ruler of sign s

Another way is to convert rulers 8,9 and 10 to 7,6 and 5:

RUL 1 DUP 7 < IF CHS 15 ADD ENDIF

RX

Stack: X RX → (1 or 0)

If planet X is retrograde the result is 1 (true), if direct, it is zero (false). The planet numbers allowed are listed under PSI.

SCALE

x y SCALE (define graphic frame)

Normally graphics work within a coordinatesystem of x and y between +-1024. This can be changed using SCALE, which sets alternative user coordinates. With SCALE, x and y can also be different, so one is not any more confined to using a square graphic area. The GRON and window resize handler will then adapt the device plotting area to make the graphic as big as possible within the window allowed.

The x and y units are still equal so that e.g. drawing a square of 100 * 100 will actually look like a square on both screen and printer.

The SCALE code must be executed before GRON to have any effect. GRON will do the actual rescaling and clear the graphic frame.

This outer frame is made from the left and right, top and bottom margins of the printed page. The margins are defined in the preferences page layout menu of ARGUS.

Then the graphic frame is scaled down according to the wheelsize defined in the page layout preference.

To get rid of these restrictions, you may change the page layout from within XLI before calling SCALE. There is no single code for this, but here is the procedure:

```
$
PTMP
0   1 NTOS  1 -1 SYSTR          ;set left margin to 0
1   1 NTOS  6 -1 SYSTR          ;set right margin to 1
100 1 NTOS 15 -1 SYSTR          ;set plot scale to 100 %
0   1 NTOS 18 -1 SYSTR          ;set vertical margin to 0 %

$
```

The PTMP assures, that the Argus page layout change is local to the current XLI session.

See also: GRON GROFF RSIZE ROTAT GPUSH GPOP GCLR

SCM

Stack: SCM →

Inside an encrypted module, you may read the encryption key using this clode

Default value: -1 (not encrypted)

See also USR

SETTI

FP Stack: gmt GETTI →

Insert FP time in days since 1. jan 1900 at 0h GMT as menu values into the date and time fields. The value includes time of day as decimals.

The content of the zone field is used to adjust the inserted GMT to match the menu zone, which is unchanged.

get GMT time from input data as FP number. This will be a positive number if the date is in 19xx or later, and negative if before.

See also: GETTI GETZO SETZO MEGET MEPUT

SETZO

FP Stack: z SETZO →

Insert timezone (including DST) as menu values.
z is the timezone measured in days east positive west negative, a number between -0.5 (12 h west) and +0.5 (12 h east);

See also GETZO GETTI SETTI

SHAPE

Stack: SHAPE →

Draw shape from values in string. The shape points coordinates are compacted into a string. This format is also used in the shape tables stored in system variables 248-420

Header is the first four positions in the string, To be interpreted as header, at least one of these four characters must be outside the range "0"-"F" (chr 48-70). The header is used to describe what the symbol is about, for example Vir (virgo). The header may be omitted, in which case the string must be all valid HEX codes.

The following characters are Hex-codes in groups of 4: XXYY

XX="00" and YY="00" plots to (-510,-510)
XX="FF" and FF="00" plots to (+510,+510)

So drawing is done in a square matrix of 256 x 256 units

The shape is scaled down by 0.6 if the header contains a ":" and by 0.8 if the header contains a ".".

In case two succedent groups are equal, the shape up to that point is closed and drawn with the current pen color and pen width and filled with the current brush color: This way it is possible to draw composite shapes with multiple fills.

See also: DRSYM

SIN

FPstack: X SIN → sin(x)

FP sine (degrees)

See also COS TAN ATAN ACOS ASIN

SKZ

Stack b SKZ →

If b is zero: ignore the rest of the paragraph and any output it may produce.

See also: WAIT CONT

SNAME

Converts a sign number to a negative system string number for use with NUMS. SNAME is the equivalent of writing: 570 ADD CHS

see also ANAME, PNAME

SNO (serial number)

Stack: SNO → (number)

Get Argus serial number. This code is obsolete, as it only takes the lowest 16 bits of the number, i.e. et cannot handle serial numbers higher than 32767.

See also: USR SCM PMIS PASSW

SORT

FP stack: x1 x2...xn N SORT→ p1 p2..pn

A tiny bubble sort of max 15 FP values (items)

N is the number of items to sort
X1, x2.. the items
p1, p2.. are pointers to xmin....xmax
The values themselves are lost.

Note: This sorting is obsolete. you could probably get a better control and performance by XLI programming the sort yourself. Using the above, You should keep a copy of the original values. A pointer to where the value was placed before is not very useful.

SP2RE

FP Spherical to rectangular coordinates

See also: RE2SP ROT ROTX ROTY ROTZ

SPRCH

Stack: lopla hipla width SPRCH →

Spread planets of given chart. The chart memory in Argus holds arrays of positions and other planetary and house info. A second array is reserved for "spreaded" positions, that is, planets positioned too close to each other to display in the chartwheel without overlapping are moved a little away from each other to avoid the problem.

Initially the spreaded array holds the true positions, to spread them use the code SPRCH code.

lopla: the lowest planet to spread, normally the Sun (1)
hipla: the highest planet to spread, normally Pluto (10) or Moons node (11)
width: the minimum acceptable distance between two planets e.g. 8 (degrees)

The chartwheel functions in Argus automatically calls the spread with a width defined by the planet symbol size set in the preferences menu. You may need to call the function in XLI if you are using DRPLA to draw planets.

See also: SPRED

SPRED

Stack: lomem himem spread SPRED →

Spread planets like in chartwhel. This spreading function is more low level than SPRCH giving you more control, but it does not change the chart memory array for spreaded positions. The only way to do that is to use SPRCH.

The positions must be located in intang units in the STO array from lomem to himem. So there will be himem-lomem+1 planet positions to spread. z is the minimum angle allowed also in intang.

If spread is positive, the spreading is circular, e.g. a planet in 0 degrees are spread from a planet in 359 degrees. If spread is negative the spreading is linear. A linear spread can be used for drawing planets along a linear axis of a graph. The astromap module delivered with Argus 4.2 is an example of this.

See also: DRPLA

STCAT (string concatenate)

Stack: X Y Z STCAT →

Join to strings into one. X, Y and Z are string array indexes for 1st string, 2nd string and destination respectively.

So for example:

```
$
1 CARY

Julian,+,Claire,
$
0 1 3 STCAT     ;join Julian and '+' and place as string 3
3 2 3 STCAT     ;join Julian+ and Claire and put back to 3
3 NUMS

@@@@@@@@@@@@@@@@@@@@@@@@
$$$
```

The above should print Julian+Claire

See also: STCMP, STCUT, STCAT, STDEF, STLEN, STPOS CARY

STCHR

Stack: chr pos ix STCHR > oldchr

Manipulate single char in string

chr: character (ascii value) to insert
pos: position in string first position is 1
ix: string number to manipulate
result: oldch, the character previously at position pos

If pos is negative, you will get oldchr without changing the string

See also: STCMP, STCUT, STCAT, STDEF, STLEN, STPOS, CARY

STCMP (string compare)

Stack: X Y STCMP → (0 or 1)

Compare to strings in the string array. The result of the compare will be:

0: Strings are equal
1: Strings are different

The compare is not case sensitive:

```
$
1 CARY

Ähnlich,+,ähnlich,
$
0 2 STCMP NUMS

######
$$$
```

This will produce a zero, because string 0 and string 2 are regarded as equal, even if one starts with a capital, and the other with a lowercase letter.

See also: STCUT, STCAT, STCHR,STDEF, STLEN, STPOS CARY

STCUT (string cutout)

Stack: X Y P Q STCUT →

Cut out a substring from a string in the string array.

X: Source string index

Y: Destination index
P: Cutout start at character number
Q: Cutout end at character number

The following example prints 9 different cuts of the same word:

```
$
1 CARY

Macedonia,
$
1 9 FOR  ;loop 9 times
0                   ;index of string to cut
1                   ;index where to put the result
1 CNT               ;start of cut
1 CNT 3 ADD             ;end of cut
STCUT
1 NUMS

@@@@@@@@@@@@@@@@@@
$
NEXT

$$$
```

This will print like this:

```
Mace
aced
cedo
edon
doni
onia
nia
ia
a
```

See also: STCMP, STCAT, STCHR,STDEF, STLEN, STPOS, CARY

STDEF (string define)

Stack: X STDEF →

Insert a string into the string array number X. The string must be in the first line in the text field. For example to insert the phrase: "IBM the big blue" as string array number 17:

```
$
```

17 STDEF

IBM the big blue
$

This is used to define or replace single string array elements. If you wish to define the complete string array in one go, it is easier to use the CARY code.

See also: STCMP, STCUT, STCAT, STCHR, STLEN, STPOS, CARY

STLEN (string length)

Stack: X STLEN → (length of string X)

String length of number X in the string array.

Example:

$
1 STDEF

Atlantic
$
1 STLEN NUMS

#########
$$$

This will print the number 8 which is the number of characters in the word: Atlantic.

See also: STCMP, STCUT, STCAT, STCHR,STDEF, STPOS; CARY

STO
STO~
NSTO~

Stack: X Y STO → X
FP stack: X Y NSTO~ X
single storage position X STO~

Store X in memory cell number Y. There are 3000 memory cells available (numbered 0-2999), so it is possible to create tables etc. This limit can be expanded using the code

ISTOZ Note that X is not removed, so storing the same value in several cells could look like this:

26 0 STO 1 STO 2 STO 3 ST0 ....

which will store the number 26 in cell 0-3 inclusive.

The FP version (NSTO~) works on the FP STACK and storage. Initially Argus has 32 STO storage positions, but you may expand this using the code FSTOZ

There is also a single position memory, which can be accessed with the STO~ code.

See also: RCL RCL~ NRCL~ FSTOZ ISTOZ

STPOS

x y STPOS (find substring in string) → position Look for substring stringarray[x] in string stringarray[y].

If not found, returns 0, else return position no.

See also: STCMP, STCUT, STCAT, STCHR,STDEF, STLEN; CARY

SUB
SUB~

Stack: X Y SUB → X-Y

Subtracts two numbers

The FP version works on the FP stack

See also ADD DIV SUB MUL MOD DIVR MULT

SUBR

n SUBR (Call previously scanned subroutine)

This code will call subroutine n (if it was defined). If you call an undefined subroutine (no previous PROC with the same value of n) you will get an error message telling that no subroutine n exists.

Parameters:

Note, that you may use the stack to transfer parameters and results. In the example below, a chartwheel printed by the subroutine is resized with a figure, which is calculated by the calling routine and pushed on the stack. The subroutine takes this value from the stack and resizes appropriately.

Here is a very simple subroutine example:

```
$
1 PROC RSIZE 12 0 WHEEL RETN

$
GRON
0 15 FOR 1 CNT 10 MUL 1 SUBR NEXT
GROFF

$$$
```

Please note, that FOR loops do not reliably work across subroutines.

See also: PROC RETN

SWDAT

x y SWDAT

Swaps the input menu data

x y

1 0 radix <> actual
1 5 radix <> horary
1 6 radix <> selected
0 0 current <> horary
0 6 current <> selected
0 3 current <> temporary radix
0 4 current <> temporary current
5 6 horary <> selected

temporary: both radix and current are saved

NOTE: The swap includes the fields: Name, Place (city), Country and area code. Note that the selected card may have been forced by Argus to point to ACTUAL since the XLI was called. Swapping 0<>0, 1<>1 etc has no effect. X and y must be different, but the order is irrelevant, so 0 2 SWDAT and 2 0 SWDAT will have the same effect.

NOTE: index 3 and 4 are temporary copies of index 1 (radix) and 2 (current). If the code PTMP has been used the XLI will restore the original values upon return, meaning that the

data in index 3 and 4 will be used to restore the input menu. Accessing them here means, that it is actually possible to change even the restore values, even if this is normally not recommended. They should only be read.

The full list of possible indices is:

```
Index  0: Currentdata          (earlier radixdata)
       1: Radixdata           (earlier currentdata)
       2: Currentdata          (earlier presentdata)
       3: Temprdata           (earlier selecteddata)
       4: Tempcdata           (new)
       5: Presentdata          (new)
       6: Selecteddata         (new)
```

See also BDSEL

SWFLG

Stack f SWFLG →

Set extra Swiss Ephemeris flags

The swiss ephemeris specifies a couple of flags to control how planets are calculated.

SEFLG_JPLEPH	1L	use JPL ephemeris
SEFLG_SWIEPH	2L	use SWISSEPH ephemeris, default
SEFLG_MOSEPH	4L	use Moshier ephemeris
SEFLG_HELCTR	8L	return heliocentric position
SEFLG_TRUEPOS	16L	return true positions, not apparent
SEFLG_J2000	32L	no precession, i.e. give J2000 equinox
SEFLG_NONUT	64L	no nutation, i.e. mean equinox of date
SEFLG_SPEED3	128L	speed from 3 positions (poor performance)
SEFLG_SPEED	256L	high precision speed (analyt. comp.)
SEFLG_NOGDEFL	512L	turn off gravitational deflection
SEFLG_NOABERR	1024L	turn off 'annual' aberration of light
SEFLG_EQUATORIAL	2048L	equatorial positions are wanted
SEFLG_XYZ	4096L	cartesian, not polar, coordinates
SEFLG_RADIANS	8192L	coordinates in radians, not degrees
SEFLG_BARYCTR	16384L	barycentric positions
SEFLG_TOPOCTR	(32*1024L)	topocentric positions
SEFLG_SIDEREAL	(64*1024L)	sidereal positions

SYMIX (symbol index)

Stack: SYMIX →

Symbol mapping determines which symbols are drawn in the chartwheel and when using the DRSYM. These symbol shapes are defined in system variables 248 to 420 and can be redefined with the code SHAPE.

This table works both for position output using the pca-ansi.ttf font and for plottet output like chartwheels etc.

There is another (longer) symbol mapping table in system string 10 which works for font output only, and may be overridden if abbreviations are selected instead of symbols.

Both tables can also be changed manually in the system string preferences or whith the SYSTR code.

With SYMIX you may assign which of the symbols are shown by which codes. The symbol mapping table (system variable 31) is changed e.g. when you change which Pluto symbol to use in the preferences menu.

The assignment is determined by a string, which you should apply as the first line in the text field. Only the first 24 symbols may be reassigned, so the string should not exceed 24 characters. An example:

$
SYMIX

€□‚ƒ„…†‡ˆ‰Š‹Œ□Ž□□‘’“”•–—
$$$

The positions in the string represent the symbol to replace, the characters the replacement.

The replacement symbols are:

128-138	=	Sun to Pluto (Danish symbols)
145-156	=	Ari to Psc
139	=	Node
144	=	Arrow
141	=	German Uranus
142	=	German Pluto
143	=	English Pluto

See also: SHAPE SYSTR

SYN

Stack: SYN →

This is a highly specialised code for synastry output, of the type, where you merge the two person's names into the text.

SYN have two effects:

1) It inserts the current name and the radix name into position 1-4 in the string array. If any strings are already defined, their indexes will be incremented by 5. If radix person is Louise, and Radix2 person is Alexander, the string array will be:

String 0: empty
String 1: Louise
String 2: Alexander
String 3: Louise's
String 4: Alexander's
String 5: (former string 0) etc.

2) It turns on a non-padding mode which will change the way NUMS inserts strings into the text. Normally, when a row of @@@@@@@@@@@@ is met NUMS will insert a string and if there are more @'s than the length of the string, it will be padded with spaces. This assures, that all lines will keep their lengths. SYN will turn the padding off, so that the text will smooth without empty spaces after the names. However, you must put enough @@@@'s to hold the longest name you will accept, which may be up to 20. This will make most lines holding names truncate quite much, so better use say 10, and tell users not to insert longer names.

A better solution is to use 3 PAD which lets you use a single @ as text template which will then expand to the actual string length. In that case, you should add 3 PAD after SYN

See also: PAD

SYSAV

n l SYSAV (Save selected part of system strings to PCA.CFG)

Save l lines from line n onwards to PCA.CFG. This means, that XLI can make a change and update the disk, for example if you have an XLI module to make the user install options.

See also: SYSTR

SYSTR

L C SYSTR

SYSTR can access nearly all the PCA.CFG strings. A few are restricted The lines are numbered from zero onwards.

L is the line number in the configuration file.

C is the destination index in the string array.

Write: If C is negative, it means write configuration string,
Read: if it is positive or zero it means read.

So to read e.g CFG line 126 (F6 Macro) into string array no 34:
126 34 SYSTR.
To write it back:
1123 -34 SYSTR.

Special cases and restrictions for certain system strings:

0: read name of namefile
1: read header
3: read serial name
5: read current macro
67 read/write system string 3
68 read/write system string 0
69 read/write system string 1
71 read/write system string 5
73 (user number) is read only
81 C>0 applying orbs read system string 141
C<=0 write system string 141 and 142
82 C>0 applying orbs read system string 145
C<=0 write system string 145 and 146
83 C>0 applying orbs read system string 147
C<=0 write system string 147 and 148
84 C>0 applying orbs read system string 153
C<=0 write system string 153 and 154
232..456 (shape tables) C<0 write shape table stroke converted to fill
above 500 : read or write language string L

Overwriting system strings is powerful and should be used with care. Changes are normally temporary though. If you start your module with PTMP the changes are local to the module, i.e. changed back at module exit. Without PTMP changes are local to the Argus session and system strings will be restored next time you run Argus unless you use the Save Changes in the preferences menu or use SYSAV to save some or many lines.

See also: SYSAV

TAB

n TAB (tabulate)

Tabulate to column n. Tabulation in Argus is normally done by changing the font in the current line to monospaced (font: pca-ansi.ttf) and then insert spaces (if needed).

If the tab is less than the current point the next text is written from the nth column the line is cleared from column n and the new text starts there. Unless you use continuation line (putting a \ at the end of the previous line) tab simply inserts n spaces (changing font to monospaced).

See also: TABLN

TABLN

n TABLN

Print a horizontal line in text from current position till position n using the tabline color defined in PCA.CFG confix[6]. If n=0 it will draw length 80.

Please note, that no linefeed is added. So you may add text which will be kept on the same line. In many cases, you will want a linefeed which must be added by printing a blank line.

TAN

FPstack: X TAN → tan(x)

FP tan (degrees)

See also SIN COS ATAN ACOS ASIN

TIDY

This code tests if there has been a PTMP data save code during the current XLI session. If so, the values of the system variables and the 'current' birthdata will get restored to the value they had when the PTMP code was met. The test for PTMP is checking for the filename SYSVARS.$$$, which is then erased.

TXCOL

c TXCOL (change text color)

Change the current text output colour. C is an 3-digit RGB value between 000 and 888.

See also COL

USR

Stack: USR → user code

The interpreter uses the user number for encryption. The user code extracted is a a single 8-bit number derived from the user number: hi(user) xor lo(user) where hi is bit 8-15 and lo is bit 0-7.

See also: SNO

UTFIL

X UTFIL print to file, equivalent to INFIL

X=n write n lines to file
X=0 close file
X=-1 open file for appending

if X>0 then the two first lines in the text part are used:

line 1=filename
line 2=print line

if the file is already open, the filename is ignored and writing will go to the current writefile. When finished, the file should be closed by coding 0 UTFIL.

IF the code NUMS is included in the paragraph, variables can be output the same way as normally written to screen or printer. In that case you just include templates in the print line where you want the numbers or strings to be printed:

To append text to an existing file, you must first open it using -1 UTFIL having the first textline specify the filename. Succedent n UTFIL (n>0) will append to this file as normal.

See also: INFIL NUMS

VBTIM

Stack b VBTIM→

b=1 Dont translate character set in output
b=0 translate character set

Setting VBTIM to 1 has further effects:
disables inline font commands (see FONT)
disables translation of tabs to spaces

See also: OEM FONT

VICI

Stack: n VICI →

Lookup in the atlas n closest cities to the current input menu location. If n is very big, it make take some time to complete.

WAIT

Stack: WAIT →

Say you are writing an interpretation part with a heading and a number of paragraphs. The conditions for these paragraphs may be of such a kind, that in some cases none of them are true. It could for example be a heading called "Special features" treating different aspect patterns and other combinations. If a chart has none of these patterns, you may get the heading without any text. The problem is, that at the time, the heading must be printed, you do not know if it should be printed at all!

Of course you may make all the coding twice, one to check all conditions just to see if the heading should be printed, and then next to check for each paragraph text. But this is tedious and clumsy.

The solution is to collect all these paragraphs in one file. The first paragraph of this file must hold the heading and the code WAIT:

```
$
WAIT

SPECIAL FEATURES:
----------
$
```

After this all the other paragraphs belonging to this heading should follow.

Now the heading in the WAIT paragraph will be withheld, and not printed until a valid paragraph is met. First then the heading is printed, and then the paragraph.

The pending heading will be discarded when the file end or the code CONT is met. That is why you may put just the paragraphs belonging to this heading in that file. The CONT code will discard any waiting paragraph and continue normally.

You may have only one paragraph with the WAIT code, and this MUST be the first paragraph in the file.

See also CONT

WHEEL

n b WHEEL

Even if you can use MACV, MACW etc to make a chartwheel, these commands will start by clearing the graphics frame before drawing. The WHEEL code will just call the wheel without disturbing other graphics.

n=0 current chart
n=1 radix chart
n=2 current chart
n=3 aux 1 chart
n=4 aux 2 chart
n=5 present chart
n=12 double wheel with radix as outer and current as inner wheel
n=25 double wheel with current as outer and present as inner wheel

If n is two digits, the first is the outer and the second is the inner

b=0 houseless wheel
b=1 normal wheel
b=2 houseless wheel without data written below
b=3 normal wheel without data written below

The WHEEL code lets you mix a chartwheel with other graphics putting them into the same GRON GROFF part.

To use WHEEL, like other graphics command, you must first turn on graphics with the GRON code.

WKEY

Stack: WKEY →

Print the usual "Please press key" prompt, and wait for any keypress before continuing.

See also: MENU OPT KEY

XFEED

- Obsolete, no action

XIF (eXit if)

Stack: X XIF →

This code will exit a FOR loop immediately, if X is "true" (not zero), so that it jumps directly to the first instruction after NEXT without doing the instructions inbetween.

This may be used, if you do not know on beforehand how many times the loop should count, but that it should be dependant on a condition.

For example, to test the number of aspects in a graphic transit list, you will only know when the list is exhausted (NTA produces a 0), that you should stop counting.

```
$
RTA                 ;reset transit list
0 10000 FOR             ;start large margin count
1 CNT               ;get count number
NTA                 ;get aspect details
NOT XIF  ;exit if last (not) aspect
NEXT
NUMS

There were ##### transit aspects found!
$$$
```

See also: FOR NEXT CNT

XLOG

Stack: XLOG →

Stop logfile output

See also: LOGX, DEBUG, PROFL

XPATH

n XPATH (get current XLI module path)

Loads the XLI path into stringarray[n]

see also: PPATH

XPLA (extra planetary information)

Stack: X XPLA → (info)

This code retrieves further coordinates from a previous PLA calculation. You should place this code immidiately the PLA with only the X parameter in between.

X may have the values:

0: heliocentric longitude, intang unit
1: heliocentric latitude, intang unit
2: heliocentric longitude velocity, intang unit
3: geocentric lat, intang unit
4: geocentric distance value AU*1000
5: geocentric right ascension, intang unit
6: geocentric declination, intang unit

So for example to print the declination for the Moon in degrees and minutes, you may code:

2 PLA 6 XPLA ITOMS 60 DIVR XY ABS XY NUMS

###

Note: Do not try to retrieve these figures for the part of fortune, this will not work.

See also: PPOS PDEG PV PLA YPLA XPLA XPOS FPPLA

XOR

Stack: x y XOR → x xor y

Exclusive or bitwise operation.

See also OR AND NOT

XPOS

i b p XPOS → info

(enhanced)

p is the planet number (0-11)

b is the chart type:

0 current
1 radix
2 current
3 auxchart1
4 auxchart2
5 present
6 auxchart3

i is the information type:

0: helio longitude (intang)
1: helio latitude (intang)
2: helio long velocity (intang/day)
3: geo lat (intang)
4: geo distance value AU*1000
5: geo right ascension (intang)
6: geo declination (intang)
7: ayanamsha - Not working
8: harmonic*10
9: armc (intang)

Th heliocentric values are only available for planets 0..10, outside this range, you will get a wrong planet error message.

See also: PPOS PDEG PV PLA YPLA XPLA XPLA FPPLA

YPLA

Stack: p YPLA → e speed pos

Calculate other planet or body. With this code it is possible to calculate asteroids, transplutonians etc. but you will need the full Swiss Ephemeris installed for most of them. The Swiss Ephemeris can be downloaded from our international website.

p is the planet number

0	Chiron
1-10	Sun-Pluto
11	mean node
12	true node
13	true Lilith
14	earth
15	Lilith
16	Pholus
17	Ceres
18	Pallas
19	Juno
20	Vesta
999001	Eris
90377	Sedna
136108	Haumea
28978	Ixion
136472	MakeMake
90482	Orcus
50000	Quaoar

Other bodies may be calculated using the filename number of the ephemeris files available from Swiss Ephemeris. If not included in the download from EE, you may find them on the www.astro.com swiss ephemeris website.

results pushed on the stack are

e: ephemeris used as set in preferences: 0: none 1:ee 2: Moshier 3 Swiss
speed: intang/day
pos: longitude (intang)

In case of unsuccessful calculation, the e value returned will be 0

Additionally two values are pushed on the FP stack:

speed (degrees/day)
longitude (degrees)

Example: 50000 YPLA NUMS (~~~~.~~~)
will output the position of Quaoar using the time given in the input menu

The ephemeris file for this is placed in C.\sweph\ephe\se50000s.se1
This file should work from year 1500-2100.
A larger version se50000.se1 should work from 3000BC to 3000AD

See also: PPOS PDEG PV PLA XPLA XPLA XPOS FPPLA

XY
XY~

Stack: X Y XY → Y X

Swaps the two uppermost numbers on the stack.

The FP version works on the FP stack

See also DUP FETCH ZZO GET PUT

ZMODE text fold mode

Stack: ZMODE →

1 ZMODE will make long lines fold following the window width. So use this code in the start of your interpretation, if you want it to be auto-adjusting the paragraph width. The lines will not be expanded to obtain an adjusted right margin.

0 ZMODE will make long lines unfolded, so you will see only the first part of them.
1 ZMODE will make text left adjusted
2 ZMODE will make text centered ajusted
3 ZMODE will make text right adjusted
4 ZMODE will make text fully adjusted

ZNORM

Stack: ZNORM →

When you change date in the input menu manually across a timezone shift, and an area code is present in the zone input field, the zone hour (and minute) is looked up in the zone table and ajusted correctly.

Changing date/time in XLI e.g. with MEPUT or SETTI does not automatically make this lookup which could result in a wrong zone. The ZNORM code forces this lookup.

See also MEGET MEPUT SETTI GETTI

ZODOF

x ZODOF (Change zodiac origin to x}

This will change the angular offset for planetary and house logitudes in the current chart. For instance to calculate a draco chart, just do an ordinary chart and then call 11 PPOS ZODOF. The offset is calculated from the original zodiak origin, so if call ZODOF more than once with different origin angles, they will all be relative to tropical zero Aries.

See also: AYA

ZZO

Stack: ZZO → (no action)

Marks an origin on the stack. The stack will normally be accessed "dynamically" from the top. You may however wish to access it from the bottom, which needs an origin. ZZO will mark the current top of stack, so that the following numbers put on the stack will get position 0, 1, 2 etc. These may then be accessed using the codes GET and PUT.

It can be used for instance if you need a temporary table without using STO. Use ZZO then push the table values on the stack. From then, you can use GET to get the table values or even change them (with PUT)

This does not affect the ordinary action of the stack, you may still use all the other codes as before.

See also: PUT GET

APPENDIX 1 - Tables

MACROS

Macro 8 is a macro for changing the master orb limit and the orb scheme

8R set orbscheme R
8P set orbscheme P
8S set orbscheme S
8C set orbscheme C
8X set orbscheme none
8ddmm. set master orb to ddmm

Macro 7 will set the harmonic number:

7iii,fff.

The comma separates the integer and fractional part of the harmonic number and the point terminates the argument as ususal. So e.g. the macro 712,6. will change the harmonic number to 12.6. To put it back to 1, the macro is just 71.

ITEM NUMBERING

ITEM	current	radix	GTR radix	GTR transit	colour
Chiron	0	20	@	`	
Sun	1	21	A	a	18
Moon	2	22	B	b	19
Mercury	3	23	C	c	20
Venus	4	24	D	d	21
Mars	5	25	E	e	22
Jupiter	6	26	F	f	23
Saturn	7	27	G	g	24
Uranus	8	28	H	h	25
Neptune	9	29	I	i	26
Pluto	10	30	J	j	27
Node	11	31	K	k	28
Part.fort	12	32	L	l	
ASC	14	34	N	n	
2. house	17	37	Q	q	
3. house	18	38	R	r	
4. house					
5.house					
6.house					
7.house					
8.house					
9.house					
MC	13	33	M	m	
11.house	15	35	O	o	
12.house	16	36	P	p	
Conjunction					29
Opposition					30
Square					31
Trine					32
Sextile					33
Semisquare					34
Sesquisquare					35
Inconjunct					36
Semisextile					37
Semi-quintile					38
Quintile					39
Tri-decile					40
Bi-quintile					41
Septile					42
Bi-septile					43
Tri-septile					44

The two first columns are the numbers used for e.g. PPOS and other chart access.

The GTR columns are the letters used with code GTR and GTP setting up graphic transits and progressions.

The color items in the right column refer to the color definitions which are used both in the color preferences menu and also used with codes COL DFCOL FONT PENC BRCOL GRCOL TXCOL and DFCOL

Only the indices for Current and Radix positions are shown above. It is also possible to access the positions of horary and the auxchart1 and 2 etc (used as temporary storage):

Positions numbers: *)

0..18: Current
20..38: Radix
40..58: Current (equal to 0..18, earlier Auxchart 1)
60..78: Auxchart 1 (earlier auxhchart 2)
80..98: Auxchart 2 (earlier Horary chart)
100..118: Horary chart

*) Position numbers are used in the followin codes:

PSI PHS RX HSI
HRU APOW ANUM AORB
PDEG PPOS HPOS PV
HV

PCA.CFG System variables:

Line:

0: File heading, not used
1: Language number:
2: Page header for printouts
3: Page footer for printouts
5: Left margin for printouts (% of page width)
6: Right margin for printouts (% of page width)
7: Printer font: Name,Pointsize,Style
8: Screen font: Name,Pointsize,Style
9: Symbol font name:

Characters (128-175) will use this font with symbol size and style adapted, as if they were part of the rest of the text. Please note, that these symbols in the Argus font (pca-ansi.ttf) are wider than normal, so you should assign a space extra after the symbol to accomodate the width.

10: Symbol font mapping table:

This is a row of characters, normally chr(128-175).
If an alternative symbol font is used which has the symbols placed differently, you may change the mapping.

For example, if Moon, which in ARGUS is no 130 in thenew font is no 277, then the 3rd character in themapping table string must be chr(277).

Note, that thereis also a mapping table called SYMIX which works on both text printout and on the chartwheels.

11: If present and in the range 50-500 defines the output lineheight. Default is 100.

12: Top margin used in printouts as defined in layout preferences

13: Bottom margin used in printout as defined in layout preferences

14: Left,top.width and height of main window stat latest session

15: Plot scale: size of chartwheel on printouts (0-100%) as defined in layout preferences

26: Chart style: 0=Lotus, 1=English, 2=Universal, 3=French

27: Chart orientation:
0 ASC left
1 MC up
2 Radix ASC left
3 Radix MC up
4 Aries right
5 Aries up
6 Aries left
7 Aries down

28: Aspectstyle:
0 To Symbol
1 To center
2 To Degree
3 No aspect

29: Planet size (0-100)

31: SYMIX Symbol mapping table:

This table works both for the plotted symbols on the chartwheel and the printed fonts on the positions output. It is used to determine which symbols to use for Pluto and Uranus. It works the same way as the

font mapping table (system variable 10). You should first make table 10 match the Truetype font used for text printouts, then if necessary map the correct Pluto and Uranus symbol witn this table.

32: Size of degrees and minutes figures on chartwheel.

34: Aspect linewidth

35: Linewidth (linwid) unused

36: progtab years (blank=60)

39: Astromap system:
0: zodiacal positions
1: true positions

40: Moons node:
0 Mean node
1 True node

41: Astrological reference in interpretations (1=yes 0=no)

42: Midpoint aspects type
1 180 degr.
2 90 degr.
3 45 degr.

43: Midpoint sort system
0 360 degr
1 180 degr
2 90 degr
3 45 degr
4 221/2 degr

44: House system
0 placidus
1 Koch
2 equal
3 Regiomontanus
4 campanus
5 topocentric
6 nat. degree
7 porphyry
8 alcabitius

45: Secondary progressed house system
0 no movement
1 naibod
2 kundig

3 ar solar arc
4 ecl solar arc
5 1 degr
6 true motion 360/year

46: Tertiary progressed house system
0 no movement
1 naibod
2 kundig
3 ar solar arc
4 ecl solar arc
5 1 degr
6 true motion 360/year

47: Method for part of fortune
0 traditional
1 Kontinental

48: Age point system
0 part of fortune
1 huber age point
2 true huber age point
3 logarithmic (Mann)

49: Style for graphic transits/Progressions
0 shades (like DOS version)
1 bars

53: 1=Use orbsets 0=disable orbsets

54: Master orb limit DDMM, eg. 830 means 8 deg 30 min

55: Solar arc system
0 true houses
1 equal movement

56: solar return system
0 tropical
1 siderial

57: Composite chart system
0 all houses are midpoints
1 Robert Hand method

58: Relationship chart system
0 midpoint of lat and long
1 great circle midpoint

59: Day chart method
0 naibod

1 naibod + progressed dir. arc
10 naibod
11 naibod + progressed dir. arc
20 kundig
21 kundig + progressed dir. arc
30 ar solar arc
31 ar solar arc + progressed dir. arc
40 ecl solar arc
41 ecl solar arc + progressed dir. arc
50 1 degr
51 1 degr + progressed dir. arc
60 true motion 360/year
61 true motion 360/year + progressed dir. arc

60: max items in output screen

67: checksum (control number for serial name etc)

68: serial number

69: licensee name

70: Initial data for input menu (comma separated)
date,time,zonename,Latitude,Longitude,Charttype,,ZoneID:

71: Features switches:
d= demodisk
m=pcm runs any date from demo
a=acs atlas
txxxxxxxx=time limited until given date

72: Initial data for horary input menu
Name,date,time,zonename,Latitude,Longitude,Charttype,,ZoneID:

73: Customer number

78: Aspect order. The order, in which aspects appear in the aspects printouts.
The format is mainly the same as used in the GTR XLI definition, but simpler
in that it uses only lowercase and only blocks of four for each set.

79: Aspect types: (comma separated pairs)
List of ARGUS supported aspects. The first figure
in each pair is the angle of the aspect in the format
DDDMM, e.g. 13500 for a sequiquadrate and 5125 for a septile
(51 deg. 25 min). The second figure in each pair is either
0 (do not include) or 1 (include), which reflects the setting
of the checkboxes in the orb menu.

80: Configuration swithces(0-63), inherited from DOS version (comma separated)
0: bit 0=1 Chiron included

bit 1=1 Exclude midpoint aspect type and orb
bit 2=1 True Agepoint special formula
bit 3=1 No header
bit 4=1 No Footer
bit 5=1 Jobdate printed
bit 6=1 No page numbers on printouts
bit 7=1 extended page format: "page xx of qq" (always english)

3: 0 No degrees and minutes on chartwheel
1 degrees on chartwheel
2 degrees and minutes on chartwheel

5: 0 use symbols in printouts
1 use latin abbreviations in printouts
2 use national abbreviations in printouts

6: Tabline colour 000-888 or -1 for text color;

7: Reserved for black use of Argus (no menus, auto-quit)

8: bit 0=1 No orbspeeds, aspects printed in 4 columns
bit 1=1 unused
bit 2=1 No aspects on bi-wheel
bit 3=1 Short radix houses on bi-wheel
bit 4=1 include bonatti sections
bit 5=1 suppress chartwheel aspect to angles
bit 6=1 include PtFt on chartwheel
bit 7=1 horary clock graph includes all positions to the left
bit 8=1 center chartwheel on page

10: bit 1=1 suppress progressed moon in secondary and tertiary

11: bit 1=1 sets orbcombine to minimum for planet pairs with a zeroorb planet.
Other indexes are reserved for future use.

87: Output colours, comma separated
0 Argus panel background
1 Text colour
2 Graph background
3 Graph main line
4 Graph angles
5 Graph houses
6 Ari
7 Tau
8 Gem
9 Cnc
10 Leo
11 Vir
12 Lib

13 Sco
14 Sgr
15 Cap
16 Aqr
17 Psc
18 Sun
19 Moon
20 Mercury
21 Venus
22 Mars
23 Jupiter
24 Saturn
25 Uranus
26 Neptune
27 Pluto
28 Node+Chiron
29 Cnj
30 Opp
31 Sqr
32 Tri
33 Sxt
34 ssq
35 ses
36 qqx
37 ssx
colours are decimal representations of windows color numbers.

91: Livechart logfile (movielog.nfi) size (default=100)
The live chart keeps a separate namefile used as log.
So if you open the livechart with data for a person already registered here,
you will have the same checkboxes marked as last time.

92: Date style:
0: european
1: american

93: East/West default
0: East
1: West

95: warninglevels. If this is a number (0-65535) bits set will show
warning types to switch OFF.
bit 0 set: turn off data input warnings (DST warning)
bits 1..15 not yet implemented.

96: if this is a number <> 0 swiss ephemeris will be used if present.
set to 0 to force EE internal ephemeris.

97: if swiss ephemeris is installed with ephemeris files placed
in another folder than <argusdrive>\sweph\ephe then you can put

this path here.

101: Printfile filter character table

120: Startup macro

121-132: Macro definitions for F1-F12

140: Aspect orbsets assigned to chart groups
char[charttype]=digit
charttype[1]: Radix
charttype[2]: Transit
charttype[3]: Secondary
charttype[4]: Tertiary
charttype[5]: Minor
charttype[6]: Solar arc
charttype[7]: Solar return
charttype[8]: Lunar return
charttype[9]: Composite
charttype[10]: Relationship
charttype[11]: Daychart
charttype[12]: Clock chart
charttype[13]: graphic aspects

digits 1-8 matches orbsets RTPSHDCG
digit 0 matches none (no orbset assigned).

141-156 Aspect orb limits for sets 1-8
141+142 applying and separating orbset for set 1
143+144 applying and separating orbset for set 2
.... etc...
(DDMM eg. 830 for 8 deg 30 mins)
0: orb combine:
0 Minimum orb
1 Mean orb
2 Maximum orb
1-31 orb limits for aspects no 1-32
32-50 orb limits for planet number 0-18
51 orb limit for midpoints
52 >0 use second orbset for separating aspects

248-420: Shape tables for graphic symbols
The first four positions in each string tells what
the symbol is.

The following characters are Hex-codes in groups of 4: XXYY

XX="00" and YY="00" plots to (-510,-510)
XX="FF" and FF="00" plots to (+510,+510)
So drawing is done in a square matrix of 128*128