ARB07

COLLABORATORS

	TITLE : ARB07	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARB07

1.1 ARexx For Beginners - Article 7 - Symbols Introduction

```
                AREXX FOR BEGINNERS

ARTICLE 7 - SYMBOLS INTRODUCTION

BY FRANK BUNTON

 COPYRIGHT © FRANK P. BUNTON 1995-1997

                What are Symbols?

                Names For Simple and Compound Symbols

                The SYMBOL() Function

                Simple Symbols

                Fixed Symbols

                Stem Symbols

                Compound Symbols

                Unitialised Symbols

                Assigning Values to Variable Symbols

                Limitations of Assigning With =

                Initialising Program Symbols

                Removing Symbol Values

                The DROP Instruction
               === End of Text ===
```

1.2 Article 7 - Symbols Introduction - What Are Symbols?

```
WHAT ARE SYMBOLS?

Those who have experience in some other programming languages will be
familiar with the terms:-

Constant - an item which does NOT change in value during the course of
           program execution.

Variable - an item whose value CAN change during program execution.

ARexx uses the term "Symbol" for the same things.

"Fixed Symbols" are the same as "Constants" .

"Simple", "Stem" and "Compound" symbols are similar to "simple variables"
and "array variables". (If you are not familiar with these two terms in
relation to variables then forget about them. I mention them only for
the benefit of those who know them from other languages in order for them
to be able to relate to some known entity.)

At this stage, it may be best for beginners to concentrate only on fixed
and simple symbols and leave stem and compound symbols until I deal with
them in a later article. However, I will include a brief description of
stem and compound symbols here for the sake of completeness and for those
with experience in other languages who may wish to jump ahead of
me.


=== End of Text ===
```

1.3 Article 7 - Symbols Introduction - Symbol Names

```
                SYMBOL NAMES FOR SIMPLE AND COMPOUND SYMBOLS

Fixed symbol names are numbers as explained under the heading
                Fixed Symbols
                .

The following notes relate to the general rules for naming variable
symbols.

These general rules are all you need to name simple symbols.

These general rules also apply to stem and compound variable symbols but
some special rules also apply to them which are discussed under the
headings:-


                 Stem Symbols

                 Compound Symbols
```

The names that you give to simple, stem and compound symbols can be made
up from any of these groups of characters:-

```
  a-z A-Z 0-9 . ! ? $ _
```

However, the FIRST character of a symbol name must NOT be:-

- one of the numeric characters 0-9

- The full stop character "."

For example:-

```
  Score1 is legitimate
  Score. is legitimate

  1Score is illegal
  .Score is illegal
```

Any other characters must not be used. This especially applies to the
Arithmetic Operators which we will discuss in Article 9.

For example, if you tried to use this as a symbol name:-

```
  My-Score
```

then ARexx would think that you are trying to subtract the value of one
symbol (Score) from that of another (My)!!

You can test to see if the characters that you use in a symbol name make
up a legitimate symbol name by using the
SYMBOL()
function.

As far as the alphabetic characters are concerned, the interpreter will
convert all alphabetics to upper case. Thus these names:-

```
grade
Grade
GRadE
GRADE
```

are all converted to "GRADE" and are the same symbol.

Although you can give symbols any name that you like within the rules,
such as "QDFRG", it is best to use something that is meaningful. For example,
you will find it a lot easier to read a program that, for instance, records
peoples names, addresses, etc., if you use the symbol names in the left
column rather than those in the right column:-

```
NAME         QWER
ADDRESS_1    ZXCVBNBN
ADDRESS_2    OPIPOUIP
PHONE        GFHGFH
```

The program would work just as well using the symbol names in the right

```
column (the computer is cleverer than us in this regard!) but can you
imagine trying to understand the program when you, the human, tries to
read it?!?!

There is a limit to the number of characters that you can use in the name
of a symbol. Don't worry though. This limit is 63995 characters. Above
that you get an error message. I don't think you or I will ever write
a program with symbol names this long!!

However, I think you will find it easier to handle programs that contain
symbol names that are as short as possible while still maintaining meaning
without ambiguity.


=== End of Text ===
```

1.4 Article 7 - Symbols Introduction - The Symbol() Function

```
                THE SYMBOL() FUNCTION

We have not covered functions yet but, as this one is useful in handling
symbols, and is very easy to use, I will tell you about it now.

If you enter this at a Shell/CLI prompt (substitute your own symbol name
where it says "SymbolName" and enclose it in quotes):-

  > RX "SAY SYMBOL('SymbolName')"

you will be given one of these outputs:-

  LIT if "SymbolName" is legitimate but does not yet hold a value
  VAR if "SymbolName" is legitimate and it does hold a value
  BAD if "SymbolName" is not legitimate

If you do not get "BAD" then you will know that the name that you have
picked for your symbol is a legitimate name EXCEPT that it does NOT show
BAD if you start with a number or a dot. This is because a number (including
a decimal place) is considered to be a
                fixed symbol
                .

In these examples, the output follows the "-->" indicator:-

  SAY SYMBOL('Score') --> LIT
  Score = 100 ; SAY SYMBOL('Score') --> VAR
  SAY SYMBOL('My-Score') --> BAD

The "BAD" output in the last example is because the character "-" is not
allowed in symbol names.


=== End of Text ===
```

1.5 Article 7 - Symbols Introduction - Simple Symbols

```
                SIMPLE SYMBOLS

A simple symbol is a group of characters (i.e. its
                name
                ) to which varying
values can be
                assigned
                .

The name of a simple symbol should begin with one of the alpha characters:-

  a-z or A-Z

It can begin with one of these non alpha characters:-

  ! ? $ _

but I think you will find it better to use alphabetics as it will make
it easier for you to read the program at a later time.

After the first character you can use any of:-

  the alphabetics a-z A-Z
  the numerics 0-9
  the characters ! ? $ _

but NOT the period(.). The period is reserved for
                Stem
                 and
                Compound
                 symbols,
or for decimal places in
                fixed symbols
                .

Examples of simple symbols have already been given
                above
                 when I was talking
about using names that had some meaning such as the names, addresses,
phones, etc. of people.


=== End of Text ===
```

1.6 Article 7 - Symbols Introduction - Fixed symbols

```
                FIXED SYMBOLS

A fixed symbol is one that carries the same value throughout the course
of the program. There are two types of fixed symbols which we will be
studying. They are:-
```

```
- Numbers

- Labels

If a fixed symbol is a label, then the same rules that applied to

                simple symbol names
                 also apply to labels.

If a fixed symbol is a number then it must contain only numeric characters.
In this respect, numeric characters include:-

   0-9
   .    (the decimal point)
   E    (but only where SCIENTIFIC Notation is being used.)

(Beginners can forget about the "E" and "Scientific Notation" for the time
being.)

The value of a fixed symbol is that of its name.

For example, in the line:-

  Grade = 45.3

the "45.3" is a fixed symbol which has the numeric value 45.3 and "Grade"
is a simple symbol which, after the above line, will carry the value 45.3
until changed by another line. Thus in these lines:-

  Grade = 45.3
  ... more programming
  Grade = 66.2
  ... more programming
  Grade = 99.8

The simple symbol, or variable symbol, "Grade" has had its value changed
three times by reference to three different fixed symbols, or
numbers.

If there are any non numeric characters in the name of a fixed symbol
then arithmetic calculations on the fixed symbol value WILL NOT BE
POSSIBLE!!


=== End of Text ===
```

1.7 Article 7 - Symbols Introduction - Stem Symbols

```
                STEM SYMBOLS

A Stem Symbol is one which follows the same naming rules as a simple symbol
but ends with a period (.). For example:-
```

```
   Name.
   Grade.

Stem symbols are used as a basis for
                compound symbols
                .


=== End of Text ===
```

1.8 Article 7 - Symbols Introduction - Compound Symbols

```
COMPOUND SYMBOLS

A compound symbol is one that contains more than one part to its name.
Each part is separated by a period (.). For example:-

  name.number

  name.1.2

The first has two parts (name and number) and the second has three parts
(name, 1 and 2).

Note that the stem of the compound symbol follows the same naming rules
as those mentioned for stem symbols above but that ANY of the characters
a-z, A_Z, 0-9 and !?$_ can be used for the subsequent parts of the compound
name. However, the period (.) must ONLY be used to separate the various
parts of the compound symbol.

Stem and compound symbols will be discussed in Article  29 but, as I
mentioned above, beginners would do best to forget them until they reach
that article.


=== End of Text ===
```

1.9 Article 7 - Symbols Introduction - Unitialised Symbols

```
                UNITIALISED SYMBOLS

Before a symbol has been assigned a value it is said to be unitialised.

An unitialised symbol has a value equal to its own name converted to upper
case. For example:-

  SAY Score
  --> SCORE

This is demonstrated in
                Example7-1
```

```
=== End of Text ===
```

1.10 Article 7 - Symbols Introduction - Assigning Values

```
ASSIGNING VALUES TO VARIABLE SYMBOLS

The simplest way of assigning a value is with the EQUALS (=) sign. For
example:-

  Name = 'Fred Bloggs'

This can done with the variable symbols which are the simple, stem and
compound symbols.

By their very nature, fixed symbols cannot be assigned values. This is
because they are numbers which stay fixed in value throughout the course
of the program. However, it is not an error to use something like:-

  1 = 2

With this type of usage, the = sign becomes a comparison operator which
we will look at soon.

When used like this, the expression:-

  Number1 = Number2

will take on a Boolean value of 1 if the two numbers are equal or 0 if
they are unequal.

But to get back to assignments to variable symbols using =, there are
a few points to watch out for. These are best illustrated in this example
program:-

  /* Example7-1.rexx */

  /* Section 1 */

  SAY
  SAY 'Name1 is ' Name1
  SAY 'Name2 is ' Name2
  SAY 'Value1 is ' Value1
  SAY 'Value2 is ' Value2

  /* Section 2 */

  Name1 = Fred Bloggs
  Name2 = 'Fred Bloggs'
  Value1 = 2.35
  Value2 = Value1 + 2.2

  SAY
  SAY 'Name1 is ' Name1
  SAY 'Name2 is ' Name2
```

```
  SAY 'Value1 is ' Value1
  SAY 'Value2 is ' Value2

  /* Section 3 */

  Name1 = 2.35
  Name2 = Name1 + 2.2
  Value1 = Fred Bloggs
  Value2 = 'Fred Bloggs'

  SAY
  SAY 'Name1 is ' Name1
  SAY 'Name2 is ' Name2
  SAY 'Value1 is ' Value1
  SAY 'Value2 is ' Value2
```

(When we come to look at FUNCTIONS we will find that there is a much neater way of doing this program. If you want to see it now, have a look at Example7-1a which gives exactly the same result.)

If you run this program the display in your window would be:-

```
  Name1  is NAME1
  Name2  is NAME2
  Value1 is VALUE1
  Value2 is VALUE2

  Name1  is FRED BLOGGS
  Name2  is Fred Bloggs
  Value1 is 2.35
  Value2 is 4.55

  Name1  is 2.35
  Name2  is 4.55
  Value1 is FRED BLOGGS
  Value2 is Fred Bloggs
```

There are a few things to note here.

Firstly, in Section 1 the four symbols have not yet been given any value by the programmer. Their values, as displayed by the SAY instructions, are their own names converted to upper case as is always done by ARexx - see Article 5.

Secondly, you do not need to declare the type of variable (symbol). Some other programming languages require you to indicate whether the variable will contain numeric or string values and, after that, they must stay as that type of variable (string or numeric) for the rest of the program.

With ARexx, this is not necessary. A symbol can be either type of variable (numeric or string) and its type can be altered at any time during the program operation simply by assigning a different type of value to it. This has been illustrated in the above example program by Sections 2 and 3.

Thirdly, you can use arithmetic operators (See Article 9) in setting the value of numeric variables as in the lines in Sections 2 and 3:-

```
  Value2 = Value1 + 2.2
  Name2 = Name1 + 2.2
```

Fourthly, if a value is assigned to a symbol and that value is NOT enclosed in quotes the ARexx considers it to be a symbol or a group of symbols and not a string.

This can have a number of effects:-

* The value will be converted to upper case. For example:-

```
    Name = Fred Bloggs
    SAY name

    --> FRED BLOGGS
```

   (provided that the symbols FRED and BLOGGS have not previously been assigned values.)

* If the value contains arithmetic operators then ARexx will try to carry out calculations on it. For example, if you use:-

```
    Name = Smith-Jones
```

  then ARexx will try to subtract "Jones" from "Smith" and, if "Jones" and "Smith" have not previously been given numeric values, then the program will abort with an error message.

  You should use:-

```
    Name = 'Smith-Jones'
```

* If the value you assign contains symbols that have been used elsewhere in the program, then the value of those symbols will be used. For example:-

```
   Smith = 20
   Jones = 8

   ... More programming

   Name = Smith-Jones
   SAY name

   --> 12
```

The moral is that, if you are assigning a string value to a symbol, you should always enclose the string in quotes. In Article 5 we discussed rules for using single and double quotes with the SAY command. The same rules apply to enclosing strings when assigning them as symbols values.

=== End of Text ===

1.11 Article 7 - Symbols Introduction - Limitations To Assigning With =

```
LIMITATIONS OF ASSIGNING WITH =

This method of assigning values to symbols has the very real limitation
of not allowing for any input from the program user. All values must be
set by the programmer when the program is first written.

In the next article will see how to allow the user to input symbol values
as would be necessary in some situations.


=== End of Text ===
```

1.12 Article 7 - Symbols Introduction - Initialising Program Symbols

```
INITIALISING PROGRAM SYMBOLS

It is often advisable to "Initialise" symbols at the start of a program,
especially if it is a long program. Following is an example of initialising
symbols taken from Example16-11 which simulates the toss of a coin, allows
the user to pick heads or tails.

   /* Example16-11 */

   /* Initialise the Program Symbols */

   Win = 'You picked a WINNER' /* String to say if win */
   Lose = 'Your choice was a FIZZER' /* String to say if lose */
   Heads = 0 /* Keep track of number of heads */
   Tails = 0 /* Keep track of number of tails */
   Wins = 0 /* Keep track of number of wins */
   Losses = 0 /* Keep track of number of losses */
   HeadTail = RANDOM(,,time('s')) /* record result of toss */
   YesNo = /* record yes or no response */
   Choice = /* record choice of H or T or Q */

   /* Main Program Loop */
   .... Rest of Program


This example is somewhat over simplified but it demonstrates the points
I am about to make.

There are a number of reasons for doing this sort of thing.

In some cases you might need the symbol to have an initial value as soon
as the program is started. This is the case with the symbols "Win" and
"Lose" above. These symbols are used with the SAY command, as in:-

  SAY Win

which will not put "Win" into the display window but will put there the
value held by the symbol "Win", i.e.:-
```

```
  You picked a WINNER
```

or, if he loses, the symbol "Lose" is SAYed and the display is:-

```
  Your choice was a FIZZER
```

In the above example it is hardly worth setting up symbols for these small strings. However, I have done so to illustrate that it can be done. If a large string is to be used over and over it is helpful to have its value stored in a symbol name to save a lot of typing. Once the symbol has been set up with a value you can just enter the appropriate SAY instruction whenever you need it.

For example:-

```
  SymbolName = 'This is an extremely long symbol value that I must use a
  lot of times and would take up a lot of typing each time'

  SAY SymbolName
```

This is a lot better than typing this a lot of times:-

```
  SAY 'This is an extremely long symbol value that I must use a lot of
  times and would take up a lot of typing each time'
```

Another case for setting up initial values would be in a situation such as a game where you wish the player to start with a score of more than zero, say 100:-

```
  SCORE = 100
```

Another reason is that you may wish to do some arithmetic on a symbol's value. In our coin toss program we will keep track of the number of times things happen with lines such as:-

```
  Heads = Heads + 1
```

This line is really saying "increase the value of Heads by 1". However, if the symbol "Heads" has not previously been given a numeric value, then its value will be itself which is not numeric. ARexx will then attempt to add a number to a non number which is not possible. The program will stop with an error message when it reaches this line. Therefore, give the symbol an initial numeric value at the start of the program:-

```
  Heads = 0
```

And another reason is that us poor feeble minded humans often have trouble remembering what the symbols we used when we wrote a program a few months or years ago are really for! Not knowing the purpose of a symbol makes it very hard to understand the program.

Another thing us poor humans do, particularly in large programs written over a long period of time, or maybe added to after the initial writing, is to use a name for a symbol that was used earlier in the program for a different purpose. Having a table of symbols at the start means that we can refer to it before picking a name. Just make sure that every new

symbol has its name added to the table!!

Thus it is wise to include, at the start of the program, a brief description within a comment as I have done in the above list. This is the only reason for the inclusion of the symbols "YesNo" and "Choice" in the above list.

Note that I have put an = sign after these symbol names. Without this, ARexx will interpret the symbols as commands to be sent to an external program.

=== End of Text ===

1.13 Article 7 - Symbols Introduction - Removing Symbol Values

REMOVING SYMBOL VALUES

Having assigned values to symbols, there may be times when we want to remove those values.

Let's say a program can be sent back to the start after it has finished running (as can a game that can be replayed) without loading it up from disk again. If the program's symbols have values that are irrelevant when it starts again, then those values must be removed.

If you want a numeric symbol to start off with a numeric value of zero, then you simply use this (as we did when initialising symbols - see above):-

```
Symbol = 0
```

If you want a string symbol to have its value removed but retain the character of a string you can use:-

```
Symbol = ''
```

This is two quote marks with nothing in between.

In both these cases, the symbol is retained in ARexx's internal list of symbols. They are still initialised.

Note that using something like:-

```
YesNo =
Choice =
```

with nothing after the = sign (as I did in the symbol table above) then the symbols loose their previous values and become unitialised. The result is the same as using:-

```
DROP YesNo
DROP Choice
```

The

DROP
instruction is another way of removing symbol values but it also removes the symbols from the list of symbols in use.

=== End of Text ===

1.14 Article 7 - Symbols Introduction - The DROP Instruction

THE DROP INSTRUCTION

The DROP command is used to remove a symbol entirely from ARexx's internal list of symbols and return it to its unitialised state.

Its syntax is:-

```
DROP Symbol [Symbol] [Symbol] [.......]
```

This means that you must have at least one symbol name after DROP but that you can also have multiple names it, each separated by a space. For example:-

```
DROP Score

DROP Name Address


Score = 10
SAY Score --> 10
DROP Score
SAY Score --> SCORE
```

When we come to look at stem and compound symbols, we will see that using DROP on a stem symbol will drop the values of all compound symbols associated with that stem.

=== End of Text ===