Power Brick LV
Data Structures & Variables

Data Structure Elements

- **Main provided method (predefined by ODT) of organizing Power PMAC information**
  - Hardware and software (memory) registers
    - Saved setup elements
    - Non-saved control elements
    - Read-only status elements

- **Accessible through on-line commands, script and C environments**

- **The IDE has a built-in "intellisense" database of structure names**
  - Automatically presents possible “completions” as you type
  - Can select from list to finish name
  - F1 “Help” function key provides full manual description

Data Structure Classes

- **Saved Setup Elements**
  - Have factory default values (**$$$*****)
  - **SAVE** command copies present active values to flash memory
  - Last-saved values copied from flash memory on normal power-up or reset (**$$$**)
  - Suggested to reside in a config. or include file in the IDE project

- **Non-saved Control Elements**
  - Have default values (usually 0) set on power-up, reset, or reinitialization
  - If required, suggested to be initialized in the IDE Project (e.g. in a config. or include file, or startup PLC)
  - Values can be set at any time in application
  - Not affected by **SAVE** command

- **Status (read-only) Elements**
  - Values automatically set by Power PMAC's firmware
  - Most are write-protected in Script environment
  - Some permit user modification for special operations

Data Structure Groups

- **Sys.**
  - Global “system” elements
  - E.g. Sys.Time
- **Motor[x].**
  - Motor elements, indexed by Motor # x
  - E.g. Motor[1].JogSpeed
- **Coord[x].**
  - Coordinate system elements, indexed by CS # x
  - E.g. Coord[1].Feedtime
- **EncTable[n].**
  - Encoder table elements, indexed by entry # n
  - E.g. EncTable[1].ScaleFactor
- **CompTable[m].**
  - Comp table elements, indexed by table # m
  - E.g. CompTable[0].OutCtrl
- **PowerBrick[i].**
  - DSPGATE3 Servo IC elements, by IC # I
  - E.g. PowerBrick[0].PhaseFreq
- **PowerBrick[i].Chan[j].**
  - DSPGATE3 channel elements, by channel # j
  - E.g. PowerBrick[0].Chan[0].EncCtrl
- **Other**
  - Brick Accessory
    - ACC84B[i].
  - Gather.
    - Data gathering elements
  - Macro.
    - MACRO ring elements

Note

Indices are integer constants or local L-variables. And they always start at 0.

Structure Name Aliases

- **Some users want to use hardware name in programs, and not ASIC name**

- **Can use "alias" name of hardware for data structure in Script**
  - Not available in C (but can do **#define** text substitution)

- **Alias names for Gate3[i]**
  - PowerBrick[i]
  - E.g. **Gate3[0].Chan[0].OutputMode** is the same as **PowerBrick[0].Chan[0].OutputMode**

Gate3 (PowerBrick) ASIC Basic Architecture

- **The Gate3 is a 4-channel interface ASIC**
  - Encoder and I/O inputs
  - Signal outputs
- **Performs "in hardware"**
  - Extremely fast and precise functions
  - Pre-processing
  - Logic
- **Saved and non-saved**
  - Multi-Channel Setup Elements e.g.
  - Channel-Specific Setup Elements e.g.

Gate3[i].Multi-Channel Setup Elements.

Gate3[i].PfmClockDiv ....
Gate3[i].PhaseClockDiv ....
Gate3[i].PhaseClockMult.......................
Gate3[i].PhaseFreq...............................
Gate3[i].PhaseServoClockCtrl..............
Gate3[i].PhaseServoDir ....
Gate3[i].ResolverCtrl............................
Gate3[i].SerialEncCtrl ....
Gate3[i].ServoClockDiv ....
....

Gate3[i]. Channel-Specific Setup Elements

Gate3[i].Chan[j].AdcOffset[k]...............
Gate3[i].Chan[j].AtanEna ....
Gate3[i].Chan[j].CaptCtrl ....
Gate3[i].Chan[j].CaptFlagChan ....
Gate3[i].Chan[j].CaptFlagSel ....
Gate3[i].Chan[j].EncCtrl.........................
Gate3[i].Chan[j].Equ1Ena ....
Gate3[i].Chan[j].EquOutMask...............
Gate3[i].Chan[j].EquOutPol ....
....

Gate3[i].PhaseFreq

Gate3[i].Chan[**0**].EncCtrl	Gate3[i].Chan[**1**].EncCtrl
Gate3[i].Chan[**2**].EncCtrl	Gate3[i].Chan[3].EncCtrl

Addresses & Pointers

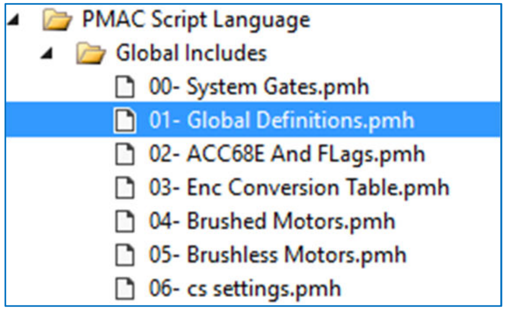
- **The “.a” suffix added to the end of an element name specifies the “address of” the element**
  - Generally do not need to know the numerical value of this address

- **A “p” at the beginning of an element name specifies “pointer to” address**
  - These elements are set to an address register (designated by the .a) rather than a constant value

Motor[1].pDac = PowerBrick[0].Chan[0].Pwm[0].a

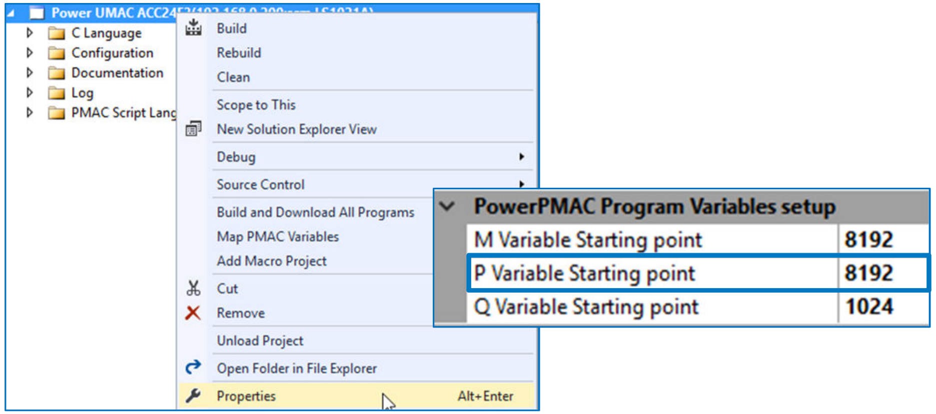
Global User Variables

- **"GLOBAL" Variables (translated into P–variables)**
  - 65K (65,536) general purpose user variables
  - Double-precision (64-bit) floating-point
  - Typically, declared in a "global includes" file in the IDE project
  - Globally accessible in both script and C




```
GLOBAL MyGlobalVar1;                                         // Not Initialized
GLOBAL MyGlobalVar2 = 1.23456;                               // Init. as Constant
GLOBAL MyGlobalVar3 = Motor[1].JogSpeed;                     // Init. as element
GLOBAL MyGlobalVar4 = MyGlobalVar3 * 1.0 / Motor[1].JogSpeed;  // Init. as expression
GLOBAL Part(10);                                             // Array
```

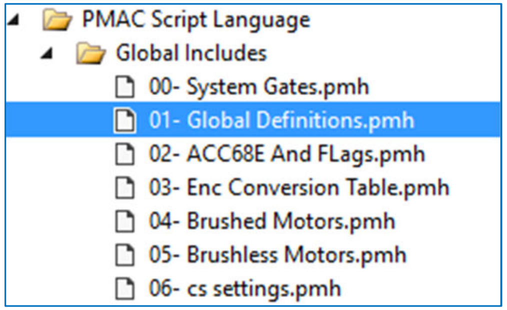
- When downloaded, the IDE project substitutes the user names with P-variables starting from 8192, default (configurable)
- Lower numbered (0 – 8191) P-variables available for manual assignments or “raw” use.
- Can be used for array creation

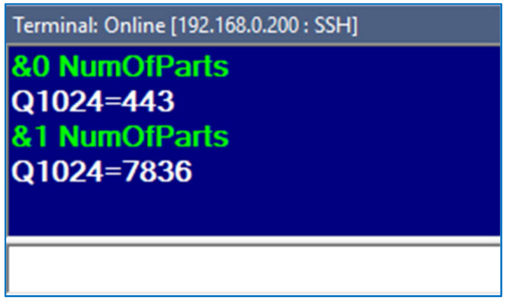


CS Global User Variables

➢ **"CSGLOBAL" Variables (translated into Q–variables)**

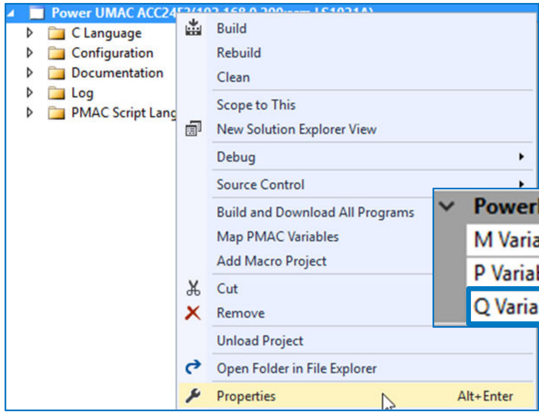
- 8K (8,192) coordinate system (each, no overlap) specific variables
- Double-precision (64-bit) floating-point
- Typically, declared in a "global includes" file in the IDE project
- Globally accessible in both script and C

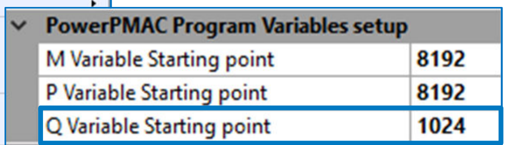





```
CSGLOBAL RunLength;
CSGLOBAL NumOfParts;
CSGLOBAL LineSpeed;
```

- When downloaded, the IDE project substitutes the user names with Q-variables starting from 1024, default (configurable)
- Lower numbered (0 – 1023) Q-variables available for manual assignments or "raw" use.





Note The presently addressed coordinate system defines which set of Q-variables the command accesses.

Local User Variables

- **"local" Variables (translated into L-Variables)**
  - 8K (8,192) general purpose local variables (per online processor, PLC, program)
  - Double-precision 64-bit floating-point
  - Typically, declared in the command processor or within a specific program
  - Only variable allowed to substitute elements' indices
  - Created upon program start, destroyed upon program completion
  - Can only be accessed "easily" in the environment where they were created

Terminal: Online [192.168.0.200 : SSH]

```
L0 = SIND(30) * 2 / SQRT(2)
L1 = L0 / COSD(45)
L1
L1=0.999999999999999889
```

```
OPEN PLC ExamplePLC
LOCAL CycleCount;

WHILE (CycleCount < 5)
{
    CycleCount++
}
DISABLE PLC ExamplePLC
CLOSE
```

Watch: Online [192.168.0.201 : SSH]

Send	On Demand	Command	Response
	☐	L10 = L10 + 1	
	☐	L11 = SIND(L10) L11	0.104528463267653471
	☐		

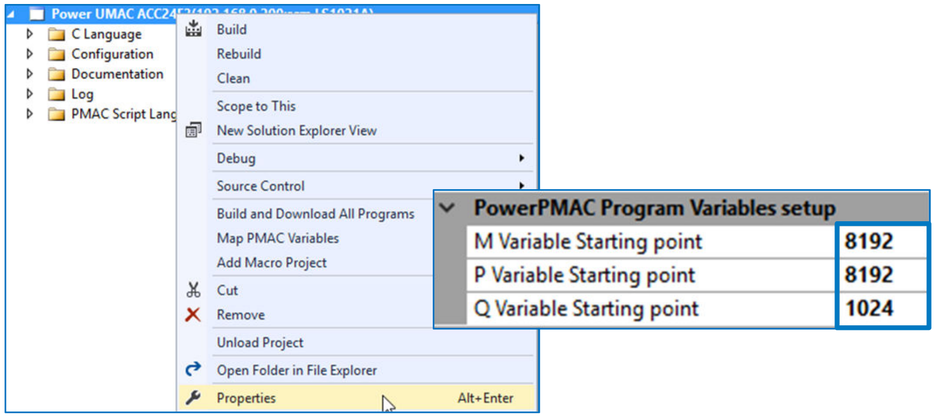
```
OPEN PLC ExamplePLC
LOCAL MotorNum;

Motor[MotorNum].JogSpeed = 0.010
DISABLE PLC ExamplePLC
CLOSE
```

Assigning User Variables Manually

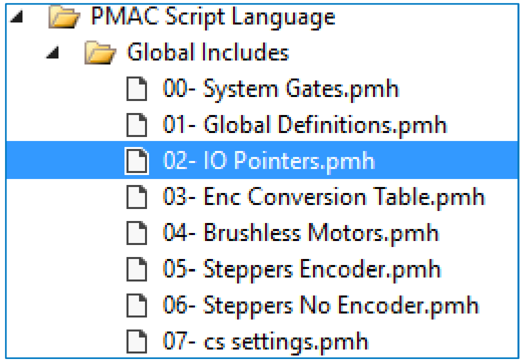
- Generally, all variables should be declared in the IDE project using **GLOBAL**, **PTR**, **CSGLOBAL**, or **LOCAL** syntax
- Occasionally, users may want to lock in variables (e.g. to tie to a user interface)

```
#define True             1
#define False            0

#define GapVoltage       P100
#define SetPressure      P102

#define StartButton      M101
#define StopButton       M102

#define NumOfCycles      Q100
#define NumOfParts       Q101
```

- Simple text substitution on download
- Should use numbers below IDE starting number



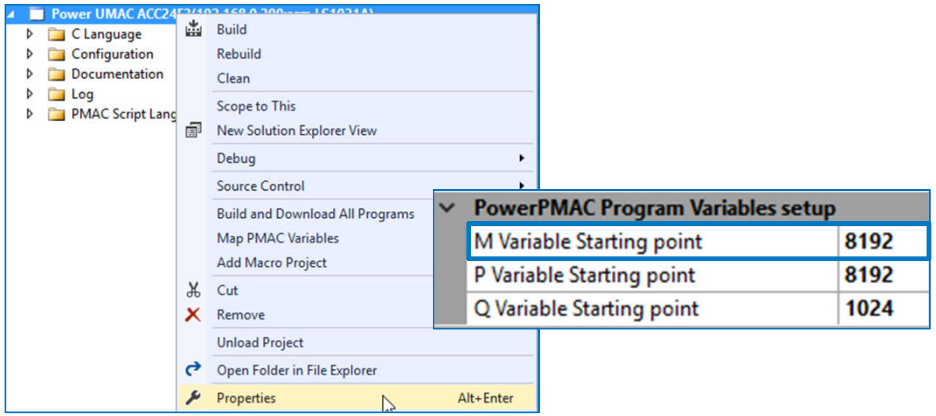
Pointer User Variables



- **"PTR" Variables (translated into M–variables)**
  - 16,384 pointers to registers
  - Used to access general purpose I/Os and memory registers
    - They take on the format of the register they are pointing to
  - Typically, declared in a "global includes" file in the IDE project
  - Globally accessible in both script and C

```
PTR Input1->PowerBrick[0].GpioData[0].0.1         // PBLV Input #1
PTR Output1->PowerBrick[0].GpioData[0].16.1       // PBLV Output #1
PTR Ch1HomeFlag->PowerBrick[0].Chan[0].HomeFlag   // PBLV Channel 1 Home Flag
```

  - When downloaded, the IDE project substitutes the user names with M-variables starting from 8192, default (configurable)
  - Lower numbered (0 – 8191) M-variables available for manual assignments or "raw" use.



Alternate Pointer Declarations

➢ General definition in I/O space

Format — Start Bit (0 – 31)

```
PTR LaserOn->U.IO:$900250.16.1
```

Variable Name Or M# (no PTR) — Address Offset — Width (1 – 32)

➢ Address Offset

Watch: Online [192.168.0.201 : SSH]

Command	Response
L0=PowerBrick[0].GpioData[0].a - Sys.piom L0	[H]:0x00900250

➢ Self-defined

- Useful to create own numerical format

Format

```
PTR LaserPwr->*S.16
```

Variable Name Or M# (no PTR) — Width (1 – 32)

➢ Formats

- S - Signed integer that saturates
- I - Signed integer that rolls over
- U - Unsigned integer that rolls over
- F - Short (32-bit) floating-point. No start/width.
- D - Long (64-bit) floating-point. No start/width.

User Shared Memory

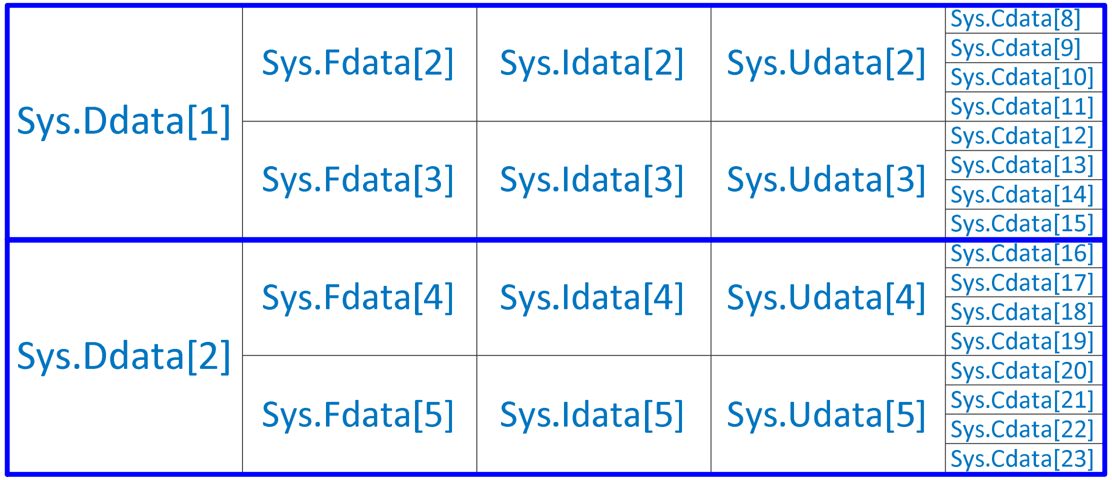
➢ Open user shared memory

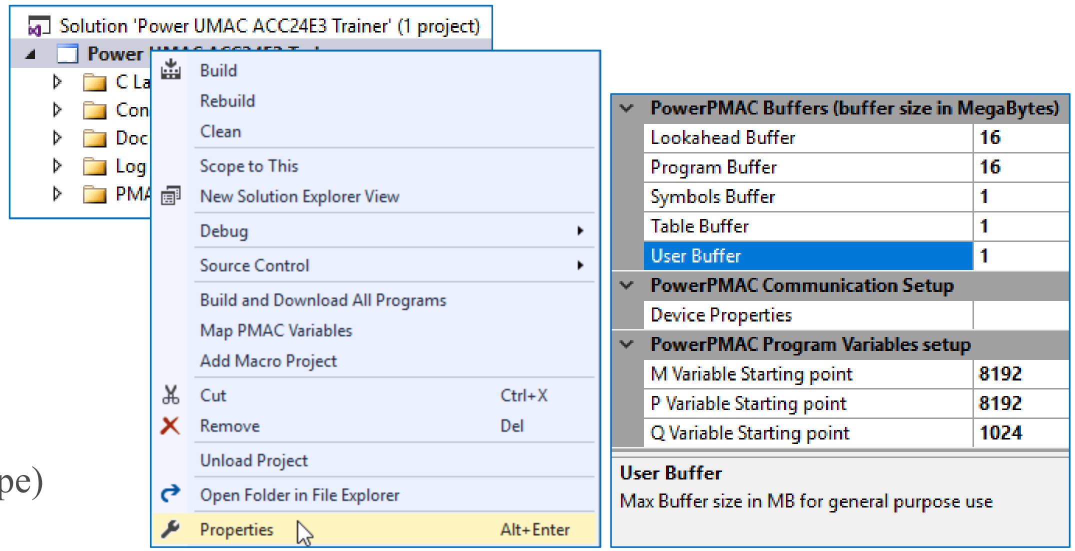
- Typically, unused by PMAC firmware
- General purpose use for storing or passing data

Note These structures elements access the same registers. Using multiple formats within the same register could result in conflicts.

- Sys.Ddata[i] Double precision (64-bit) floating-point
- Sys.Fdata[i] Single precision (32-bit) floating-point
- Sys.Idata[i] Signed Integer (32-bit)
- Sys.Udata[i] Unsigned Integer (32-bit)
- Sys.Cdata[i] Character (8-bit)

- Typical i range:
  - 0 – 8388607 for D data
  - 0 – 16777215 for I, F, and U data
  - 0 – 67108863 for C data
- Must use L-variable instead of i if greater than range (for each type)
  - Example: L0 = 12000000 Sys.Ddata[L0] = 1





User Shared Memory Access

- Can use structure elements as is
- Bitwise mapping requires pointer (M-variable) access

Watch: Online [192.168.0.201 : SSH]

Se	Command	Response
	L0 = Sys.Ddata[8].a - Sys.pushm L0	[H]:0x00000040
	L0 = Sys.Fdata[32].a - Sys.pushm L0	[H]:0x00000080
	L0 = Sys.Idata[64].a - Sys.pushm L0	[H]:0x00000100
	L0 = Sys.Udata[128].a - Sys.pushm L0	[H]:0x00000200

```
PTR Ddata8->D.USER:$40;
PTR Fdata32->F.USER:$80;
PTR Idata64Bit4->I.USER:$100.4.1;
PTR Udata128Upper16->U.USER:$200.16.16;
```

Note It is not recommended to use i = 0 (Sys.pushm)

Scalar Functions

- Trig. Functions using degrees
  - SIND, COSD, TAND, SINCOSD
- Inverse trig. Functions using degrees
  - ASIND, ACOSD, ATAND, ATAN2D
- Hyperbolic trig. functions
  - SINH, COSH, TANH
- Log, exponent functions
  - LOG(or LN), LOG2, LOG10, EXP, EXP2, POW
- Root functions
  - SQRT, CBRT, QRRT, QNRT
- Rounding, truncation functions
  - INT, RINT, FLOOR, CEIL
- Miscellaneous functions
  - ABS, SGN, REM, SEED, ISNAN
- Trig functions using radians
  - SIN, COS, TAN, SINCOS
- Inverse trig functions using radians
  - ASIN, ACOS, ATAN, ATAN2
- Inverse hyperbolic trig. functions
  - ASINH, ACOSH, ATANH
- Random number generation
  - RND, RANDX

Operators & Comparators

MATH OP.	DESCRIPTION
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulo
&	Bit-by-bit AND
|	Bit-by-bit OR
^	Bit-by-bit XOR
~	Bit-by-bit inversion
>>	Shift right
<<	Shift left

ASSIGN OP.	DESCRIPTION
=	Simple assignment
+=	With arithmetic op.
–=	
*=	
/=	
%=	
&=	With logical op.
|=	
^=	
>>=	With shift op.
<<=	
++	Increment
--	Decrement

LOGIC OP.	DESCRIPTION
&&	Logical AND
||	Logical OR

COMP.	DESCRIPTION
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
~	Approx. equal to (0.5)
!	Conditional not

Special Numerical Representations

- **Infinity; INF, –INF**
  - E.g. Division by 0
  - Operations that generate **INF** do <u>not</u> create errors
  - INF is greater than any finite number in a comparison
  - –INF is less than any finite number in a comparison

- **Not a number; NAN**
  - E.g. SQRT(-1)
  - Operations that generate **NAN** do <u>not</u> create errors
    - Boolean ISNAN can be used to check

- **Minus zero; –0**
  - Obtained by underflow of negative values
  - Equivalent to **0** (plus zero) in any subsequent operation

- **Part of IEEE-754 standard**

Special Script Functions

- **Matrix manipulation**
  - **MMUL** multiplies 2 matrices together to create a 3rd matrix
  - **MMADD** multiplies 2 matrices together, adds product to a 3rd matrix
  - **MINV** inverts a square matrix to create a 2nd matrix
  - **MTRANS** transposes a matrix to create a 2nd matrix
  - **MSOLVE** solves simultaneous set(s) of equations represented by square (coefficient) matrix and 2nd (constant) vector/matrix
  - **MDET** calculates the determinant of a square matrix
  - **MMINOR** calculates specified minor determinant of square matrix

- **Vector manipulation**
  - **VADD:** adds 2 vectors together to produce a 3rd vector
  - **VCOPY** copies contents of vector into a 2nd vector
  - **VSCAL** multiplies each vector element by a common scale factor, places result in a 2nd vector
  - **SUM** adds a number of evenly spaced elements together (as for trace of matrix)
  - **SUMPROD** multiplies pairs of elements of 2 vectors together, adding products into returned value (as for dot product)

- **String manipulation (essentially the same as in C):**
  - **SPRINTF, STRCAT, STRCPY, STRNCAT, STRNCPY, STRTOLOWER, STRTOUPPER**
  - **STRCHR, STRCMP, STRCSPN, STRLEN, STRNCMP, STRPBRK, STRRCHR, STRSPN, STRSTR, STRTOD**