# *PikoCNC Macro manual*

***wersja 4.0 12.07.2021***

## Wstęp

Makra pełnią rolę pomocniczą umożliwiają tworzenie własnych prostych sekwencji ruchów, lub czynności, obsługę magazynków narzędzi itp.

Język programowanie makr składniowo jest podobny z zastosowanym w skryptach pascalem. Różnice są następujące:

- Blok z komendami musi być poprzedzony znakiem „%”
- Nie można wprost korzystać ze zmiennych globalnych.

Pliki makr muszą znajdować się w katalogu Macros/ naszego katalogu z profilem (najczęściej to katalog ProfilDef). Plik jest najpierw szukany w katalogu Macros/ a jeśli go tam nie ma, to przeszukiwany jest katalog Macros/Elcosimo/ . Jednak własne makra zawsze trzeba zapisywać w Macros/ gdyż katalog Elcosimo/ jest nadpisywany przy każdej instalacji programu.

W katalogu Macros/Elcosimo/ można znaleźć szereg przykładów i otwierając za pomocą wbudowanego edytora makr prześledzić kod.

Podstawowe typy zmiennych na jakich operują funkcje opisano w temacie funkcji SetVar.

# Lista funkcji

## *Funkcje związane z obsługą argumentów*

Wykonując makra z okienka MDI możemy podawać argumenty np. G0 X1.0 Y12 gdzie G0 to nazwa makra a X1.0 i Y12 to argumenty. Argumenty muszą być rozdzielone znakiem spacji. Nazwa argumentu to pierwszy znak, pozostałe muszą być liczbą i tak: „X” i „Y” to nazwy argumentów a występujące za nimi liczby to ich wartość. Wyjątkową nazwą argumentu jest znak „?” za którym nie musi być wartości liczbowej. Może być wykorzystany do trybu „help”. Argumentami można także przekazywać wartości binarne (klucze) wtedy przed argumentem dodajemy znak „-” np. „M6 T2 -M” Klucz -M wymusza w tym wypadku pomiar narzędzia.

| Nazwa | Opis |
|---|---|
| **IsArg** | function IsArg(name:string):boolean<br><br>Sprawdza czy argument o danej nazwie został podany. |
| **GetArg** | function GetArg(name:string):variant<br><br>Zwraca wartość argumentu o danej nazwie. Jeśli argument o takiej nazwie nie został podany to zostanie zgłoszony błąd i makro zostanie zatrzymane. |
| **GetArgDef** | function GetArgDef(name:string; deflaut:variant):variant<br><br>Zwraca wartość argumentu o danej nazwie. Jeśli argument o takiej nazwie nie został podany to zwrócona zostanie wartość deflaut. |
| **UserExe** | function UserExe():boolean'<br><br>Zwraca TRUE jeśli makro uruchomione jest ręcznie z okienka MDI lub przycisku z zakładce „Macro”. Jeżeli makro uruchomił program otrzymamy FALSE. |

## *Funkcje zgłaszania stanów wyjątkowych*

| Nazwa | Opis |
|---|---|
| **UserError** | procedure UserError(info:string)<br><br>Otwiera okienko z komunikatem „info”. Makro zostaje zatrzymane. |

## *Funkcje obsługi zmiennych*

| Nazwa | Opis |
|---|---|
| **SetVar** | procedure SetVar(num:cardinal; v:variant)<br><br>Ustawia zmienną o danym numerze wartością „v”. Funkcje SetVar/GetVar pracują na zmiennych typu „variant” czyli akceptują takie typy zmiennych jak:<br><br>Extended – liczba zmiennoprzecinkowa np. 56.123 / -0.45 / 3.14<br>Cardinal – liczba całkowita bez znaku np. 10 / 123 / 19203<br>Integer – liczba całkowita ze znakiem np. -200 / 150 / -4534<br>String – ciąg znaków (tekst) np. 'ala ma kota'<br>Bool – TRUE / FALSE.<br><br>Zapis zmiennej za pomocą SetVar równocześnie nadaje typ tej zmiennej. Dlatego przed odczytem zmiennej wcześniej trzeba ją zapisać aby miała ustalony typ. Nie należy też mieszać typów przy odczycie i np. próbować uzyskać zmienną typu string z indeksu gdzie zapisaliśmy cardinal. Tablica zmiennych nie jest kasowana przed wykonaniem makra dlatego może służyć do wymiany informacji między kolejnymi makrami. Przy standardowej długości tablicy zmiennych parametr „num” musi być w zakresie 0-249. |
| **GetVar** | function GetVar(num:cardinal):variant<br><br>Zwraca wartość zmiennej o danym numerze. |
| **SetVarSize** | procedure SetVarSize(num:cardinal)<br><br>Ustawia rozmiar tablicy zmiennych, gdzie „num” – nowy rozmiar tablicy zmiennych. Domyślnie jest to 250. Jeśli potrzebna jest większa ilość można tą funkcją to zmienić. Wywołanie powyższych funkcji z indeksem przekraczającym rozmiar tablicy spowoduje błąd i zatrzymanie makra. |

## *Funkcje związane ze skokami*

| Nazwa | Opis |
|---|---|
| **GotoLabel** | procedure GotoLabel(name:string)<br><br>Wykonuje skok do etykiety o podanej nazwie. Etykieta to nazwa występująca jako jedyna w linii poprzedzona znakiem „:” np.:<br>...kod...<br>:main_loop // etykieta o nazwie main_loop<br>...kod... |
| **ProcedureLabel** | procedure ProcedureLabel(name:string)<br><br>Wykonuje skok do etykiety o podanej nazwie, ale w odróżnieniu od GotoLabel po napotkaniu instrukcji Return wróci do miejsca z którego skok został wykonany (tzn. do kolejnej po nim instrukcji). |
| **Return** | procedure Return()<br><br>Jeśli wystąpi w głównym wątku programu spowoduje jego zakończenie. Jeśli w wątku wywołanym przez ProcedureLabel to nastąpi powrót do miejsca wywołania. |

## *Funkcje związane z odczytem aktualnej pozycji maszyny*

| Nazwa | Opis |
|---|---|
| **PosX** | function PosX():Extended<br><br>Zwraca aktualną pozycję maszynową osi X. |
| **PosY** | function PosY():Extended<br><br>j.w dla osi Y. |
| **PosZ** | function PosZ():Extended<br><br>j.w dla osi Z. |
| **PosA** | function PosA():Extended<br><br>j.w dla osi A. |
| **PosB** | function PosB():Extended<br><br>j.w dla osi B. |

## *Funkcje związane z wykonywaniem ruchu oraz zmianą pozycji osi*

| Nazwa | Opis |
|---|---|
| **SetX** | procedure SetX(pos:extended)<br><br>Ustawia rejestr pozycji dla osi X. Jest to zawsze pozycja maszynowa ! |
| **SetY** | procedure SetY(pos:extended)<br><br>j.w. dla osi Y. |
| **SetZ** | procedure SetZ(pos:extended)<br><br>j.w. dla osi Z. |
| **SetA** | procedure SetA(pos:extended)<br><br>j.w. dla osi A. |
| **SetB** | procedure SetB(pos:extended)<br><br>j.w. dla osi B. |
| **ExeMove** | procedure ExeMove(speed:integer)<br><br>Wykonanie ruch na podstawie zawartości rejestru pozycji. Argument „speed” może przyjmować następujące wartości:<br>0 - Wykonany ruch będzie G0.<br>>0 - Wykonany ruch będzie G1 z podaną prędkością.<br>-1 - Ruch nie będzie wykonany natomiast wyczyszczony zostanie rejestr pozycji.<br><br>Po wywołaniu ExeMove kolejna linia programu będzie wykonana gdy zadany ruch się zakończy. Dlatego nie wolno w jednej linii programu umieszczać więcej niż jedną taką komendę. Funkcje ExeMove, WriteMove, ExeSetPos po wywołaniu czyszczą automatycznie rejestr pozycji osi.<br><br>np.;<br>SetZ(20.0); ExeMove(0); // Przejazd G0 osi Z na pozycję 20.0<br>SetX(10.0); SetY(20.0); ExeMove(1000); // Przejazd G1 F1000 na pozycję |
| **ExeMoveExt** | procedure ExeMoveExt(speed:integer; rel:boolean; translation_mode:cardinal)<br><br>Rozbudowana wersja funkcji ExeMove<br>speed - j.w.<br>rel – ruch będzie przesunięciem względem aktualnej pozycji jeśli TRUE.<br>translation_mode – określa jaka będzie translacja ruchu:<br><br>TR_NONE – surowa pozycja maszynowa<br>TR_MACHINE – pozycja maszynowa uwzględniająca mapę korekcji geometrii.<br>TR_WORK – pozycja materiałowa |

| Nazwa | Opis |
|---|---|
| **WriteMove** | procedure WriteMove(speed:integer)<br><br>Podobnie jak ExeMove, ale nie czeka na zakończenie zadanego ruchu, tylko od razu przechodzi do kolejnej linii programu – dlatego trzeba ostrożnie ją stosować, ze świadomością konsekwencji jakie to może nieść. |
| **ExeSetPos** | procedure ExeSetPos()<br><br>Ustawia pozycję osi na podstawie zawartości rejestru pozycji.<br><br>np.;<br>SetZ(0.0); ExeSetPos; // Zerowanie pozycji osi Z |
| **ExeSetCorrection** | procedure ExeSetCorrection()<br><br>Ustawia korekcję osi na podstawie zawartości rejestru pozycji. |
| **ExeCancelCorrection** | procedure ExeCancelCorrection()<br><br>Zeruje korekcję wszystkich osi. |

## *Funkcje związane z jazdą referencyjną*

| Nazwa | Opis |
|---|---|
| **RefX** | procedure RefX(dir:integer)<br><br>Ustala kierunek jazdy referencyjnej dla osi X. Parametr „dir” określa kierunek:<br>(dir = -1) Kierunek w lewo. (dir = 1) Kierunek w prawo. |
| **RefY** | procedure RefY(dir:integer)<br><br>j.w. dla osi Y. |
| **RefZ** | procedure RefZ(dir:integer)<br><br>j.w. dla osi Z. |
| **RefA** | procedure RefA(dir:integer)<br><br>j.w. dla osi A. |
| **RefB** | procedure RefB(dir:integer)<br><br>j.w. dla osi B. |
| **ExeRef** | procedure ExeRef(mode:cardinal; speed:integer)<br><br>Wykonanie jazdy referencyjnej na podstawie zawartości rejestru pozycji. Argument „mode” może przyjmować następujące wartości:<br><br>HOME_ON_SOFT Jazda do momentu aktywacji wejścia HOME osi z łagodnym hamowaniem.<br>HOME_ON - Jazda do momentu aktywacji wejścia HOME osi.<br>HOME_OFF - Jazda do momentu dezaktywacji wejścia HOME osi.<br>INDEX_ON - Jazda do momentu aktywacji wejścia INDEX osi.<br>PROBE_ON - Jazda do momentu aktywacji wejścia PROBE. |

| Nazwa | Opis |
|---|---|
| | PROBE_OFF - Jazda do momentu dezaktywacji wejścia PROBE.<br>PROBE_ON_SOFT - Jazda do momentu aktywacji wejścia PROBE z łagodnym hamowaniem.<br><br>Parametr „speed” określa prędkość jazdy. Jeśli speed=0 to prędkość będzie jak do G0. Kolejna linia programu będzie wykonana gdy jazda referencyjna się zakończy. |
| **SoftLimit** | procedure SoftLimit(st:boolean)<br><br>Za pomocą funkcji można kontrolować wykrywanie kolizji „soft limit”. Niekiedy na czas jazdy referencyjnej zachodzi potrzeba wyłączenia SL np. w sytuacji gdy czujnik długości narzędzia ma ujemną wysokość. Po operacji zawsze trzeba załączyć SL. |

## *Funkcje związane z pozycją elementów maszyny*

| Nazwa | Opis |
|---|---|
| | |
| **PosSafe** | function PosSafe():extended<br><br>Zwraca pozycję bezpieczną dla osi narzędziowej. Jest to pozycja krańcówki HOME minus wpisany w ustawieniach zjazd. Zazwyczaj jest to pozycja HOME osi „Z”. |
| **PosBase** | function PosBase(axis:cardinal):extended<br><br>Zwraca pozycję krańcówki HOME danej osi. Wszędzie gdzie trzeba podać numer osi można stosować zadeklarowane stałe: AXIS_X, AXIS_Y, AXIS_Z, AXIS_A<br><br>np.;<br>SetX(PosBase(AXIS_X)); // Ustawia rejestr pozycji osi X położeniem jego krańcówki HOME |
| **PosPark** | function PosPark(axis,num:cardinal):extended<br><br>Zwraca pozycję park gdzie: axis – numer osi, num – numer pozycji PARK (0-7). |
| **SetPosPark** | procedure SetPosPark(axis,num:cardinal; pos:extended)<br><br>Ustawia pozycję park gdzie: axis – numer osi, num – numer pozycji PARK (0-7).<br><br>pos – nowa pozycja. |
| **PosMaterial** | function PosMaterial(axis:cardinal):extended<br><br>Zwraca pozycję materiału w danej osi. |
| **SetPosMaterial** | procedure SetPosMaterial(axis:cardinal; pos:extended)<br><br>Ustawia pozycję materiału gdzie: axis – numer osi, pos – nowa pozycja. |
| **MaterialSize** | function MaterialSize(axis:cardinal):extended<br><br>Zwraca rozmiar materiału w danej osi. |

| Nazwa | Opis |
|---|---|
| **SetMaterialSize** | procedure SetMaterialSize(axis:cardinal; size:extended)<br><br>Ustawia rozmiar materiału gdzie: axis – numer osi, size – nowy rozmiar. |
| **PosToolSensor** | function PosToolSensor(axis:cardinal):extended<br><br>Zwraca pozycję czujnika długości narzędzia w danej osi. Dla osi narzędziowej zwraca wysokość czujnika. |
| **SetSpindle** | SetSpindle(val:cardinal)'<br><br>Ustawia parametr „S” |

## *Funkcje związane z osią techniczną*

| Nazwa | Opis |
|---|---|
| **TAMove** | procedure TAMove(pos:cardinal)<br><br>Inicjuje przejazd osi technicznej do danej pozycji. Pozycja wyrażona jest w impulsach. Kolejna linia programu będzie wykonana gdy zadany ruch się zakończy. |
| **TAMoveInc** | procedure TAMoveInc(pos:integer)<br><br>jw. Inicjuje przejazd osi technicznej o podaną liczbę impulsów – czyli relatywnie względem aktualnej pozycji. Kolejna linia programu będzie wykonana gdy zadany ruch się zakończy. |
| **TAMoveRef** | procedure TAMoveRef()<br><br>Inicjuje cykl jazdy referencyjnej osi technicznej. Kolejna linia programu będzie wykonana gdy jazda referencyjna się zakończy. Funkcja ta powoduje jazdę w kierunku ujemnym, z twardym hamowaniem w momencie aktywacji wejścia TA_PROBE. Po zakończeniu cyklu licznik pozycji osi jest zerowany. |
| **TAMoveRefExt** | procedure TAMoveRefExt(mode:cardinal; dir:integer; speed:cardinal)<br><br>jw. Inicjuje cykl jazdy referencyjnej osi technicznej. Kolejna linia programu będzie wykonana gdy jazda referencyjna się zakończy. UWAGA - po zakończeniu, licznik pozycji osi nie jest ustawiany żadną wartością.<br><br>Argument „mode” może przyjmować następujące wartości:<br>TA_REF_ON - Jazda do momentu aktywacji wejścia TA_PROBE<br>TA_REF_ON_SOFT - Jazda do momentu aktywacji wejścia TA_PROBE osi z łagodnym hamowaniem.<br>TA_REF_OFF - Jazda do momentu dezaktywacji wejścia TA_PROBE osi.<br>TA_REF_OFF_SOFT - Jazda do momentu dezaktywacji wejścia TA_PROBE osi z łagodnym hamowaniem.<br><br>Parametr „dir” określa kierunek:<br>(dir = -1) Kierunek w lewo. (dir = 1) Kierunek w prawo.<br><br>Parametr „speed” określa prędkość wyrażoną w procentach prędkości maksymalnej ustawionej w parametrach osi technicznej np. jeśli wartość maksymalna STEP ustawiona jest na 5KHz to speed = 50 spowoduje jazdę z krokiem o częstotliwości 2.5KHz. |

| Nazwa | Opis |
|---|---|
| **TASpeed** | procedure TASpeed(speed:integer)<br><br>Ustala bieżącą częstotliwość STEP dla osi technicznej. Tak samo jak wyżej, wyrażoną w procentach częstotliwości maksymalnej. Od tego momentu wszystkie ruchy osi będą odbywały się z zadaną częstotliwością STEP. |
| **TASetPosition** | procedure TASetPosition(pos:integer)<br><br>Ustawia pozycję osi wyrażoną w impulsach. |
| **RefNeed** | function RefNeed():boolean<br><br>Pierwsze wywołanie po resecie tej funkcji zwraca TRUE, kolejne FALSE. Może informować o konieczności jazdy referencyjnej. |

## *Funkcje związane z obsługą magazynka narzędzi*

| Nazwa | Opis |
|---|---|
| **GetActTool** | *function GetActTool():cardinal*<br><br>Zwraca numer aktualnego narzędzia. |
| **SetActTool** | procedure SetActTool(num:cardinal)<br><br>Funkcja ustawia narzędzie o podanym numerze jako aktualne. Jeżeli jako numer narzędzia podamy zero oznacza to brak narzędzia w uchwycie, a przede wszystkim wyzerowanie korekcji dla osi narzędziowej (najczęściej „Z”). Ważne jest to szczególnie w makrach wymiany narzędzia gdzie istotne jest aktualne położenie oprawki narzędzia a nie jego końca. |
| **GetReqTool** | function GetReqTool():cardinal<br><br>Funkcja zwraca numer narzędzia wymaganego przez program. (wykorzystywana w makrach automatycznej wymiany narzędzia). |
| **SetToolPos** | procedure SetToolPos(posZ:extended)<br><br>Ustala pozycję końca narzędzia. |
| **GetToolSlot** | function GetToolSlot(num:cardinal):integer<br><br>Zwraca numer slotu narzędzia o danym numerze. Zwrócona wartość -1 mówi o tym, że narzędzie nie ma przypisanego slotu. |
| **GetToolMultiSlot** | function GetToolMultiSlot(num:cardinal):cardinal<br><br>Zwraca 32-bitową liczbę, której każdy bit oznacza wartość odhaczoną w magazynku narzędzi w kolumnie MultiSlot. |
| **ValidTool** | function ValidTool(num:cardinal):boolean<br><br>Sprawdza czy narzędzie o danym numerze jest już zmierzone. |

| Nazwa | Opis |
|---|---|
| | |
| **GetActToolLen** | function GetActToolLen():extended<br><br>Funkcja zwraca długość aktualnego narzędzia (czyli aktualną korekcje dla osi narzędziowej). |

## *Funkcje czasowych opóźnień*

| Nazwa | Opis |
|---|---|
| **WaitTime** | procedure WaitTime(time:extended)<br><br>Funkcja zatrzymuje wykonywanie makra na czas „time” wyrażony w sekundach. |

## *Funkcje odczytu przetwornika ADC*

| Nazwa | Opis |
|---|---|
| **GetADC** | function GetADC(chanel:cardinal):extended<br><br>Funkcja zwraca wartość podanego kanału przetwornika ADC. Zwracana wartość to liczba z zakresu 0.0 – 1.0 |

## *Funkcje zapisu/odczytu rejestrów PLC*

num – numer bitu (0-31).
devname – nazwa bitu zdefiniowana w PLC.
st – docelowy stan bitu.
W – zapis do rejestru, R – Odczyt rejestru.

| Rejestr | R/W | Funkcja / procedura |
|---|---|---|
| MEMO | W | procedure SetMemo(num:cardinal; st:boolean) |
| | W | procedure SetMemoN(devname:string; st:boolean) |
| | R | function Memo(num:cardinal):boolean |
| | R | function MemoN(devname:string):boolean |
| IN | R | function Input(num:cardinal):boolean |
| | R | function InputN(devname:string):boolean |
| OUT | R | function Output(num:cardinal):boolean |
| | R | function OutputN(devname:string):boolean |
| TIMER | W | procedure SetTimer(num:cardinal; time:extended)<br><br>Procedura ustala czas timera gdzie:<br>num – numer timera (0-7), time – czas timera w sekundach (0.001 – 65 sek.) |

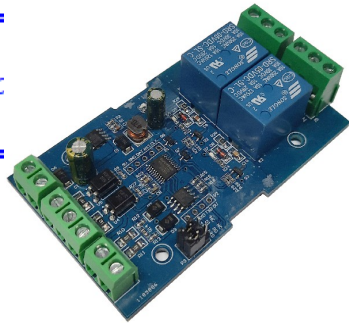
## *MODBUS*

Z poziomu makr możemy komunikować się poprzez protokół Modbus RTU z zewnętrznymi urządzeniami takimi jak zdalne wyjścia/wejścia, PLC, itd. Konfigurację połączenia należy dokonać w oknie ustawień kontrolera pozycja „MODBUS”. Szczegółowy opis samego protokołu można znaleźć pod linkiem: MODBUS

| Nazwa | Opis |
|---|---|
| **MB_SendStr** | function MB_SendStr(data:string):boolean;<br><br>Wysyła ramkę danych zapisaną jako ciąg liczb szesnastkowych. W ramce pomijamy ostatnie dwa bajty sumy kontrolnej – jest ona wyliczana automatycznie.<br><br>Przykład:<br><br>MB_SendStr('FF 05 00 00 FF 00'); |
| **MB_Send** | function MB_Send():boolean;<br><br>Wysyła ramkę danych utworzoną za pomocą procedur: MB_InitFrame, MB_wr_BYTE, MB_wr_WORD. Tak, jak wyżej w ramce pomijamy ostatnie dwa bajty sumy kontrolnej.<br><br>Przykład wysłania ramki identycznej jak w powyższym przykładzie:<br><br>MB_InitFrame($FF, 5);<br>MB_ wr_WORD(0);<br>MB_ wr_WORD($FF00);<br>MB_Send(): |
| **MB_InitFrame** | procedure MB_InitFrame(addr:byte; command:byte);<br><br>Inicjuje tworzenie nowej ramki.<br><br>addr – adres urządzenia slave, command – komenda Modbus. |
| **MB_wr_BYTE** | procedure MB_wr_BYTE(data:byte);<br><br>Zapisuje kolejny bajt do ramki danych do wysłania. |
| **MB_wr_WORD** | procedure MB_wr_WORD(data:word);<br><br>Zapisuje kolejne słowo (2 bajty) do ramki danych do wysłania. |
| **MB_rd_BYTE** | function MB_rd_BYTE(indx:cardinal):byte;<br><br>Odczyt bajtu danych z ramki, którą wysłało urządzenie.<br>indx – indeks bajtu (numeracja od zera). |
| **MB_rd_WORD** | function MB_rd_WORD(indx:cardinal):word;<br><br>Odczyt słowa (2 bajty) danych z ramki, którą wysłało urządzenie.<br>indx – indeks pierwszego bajtu (numeracja od zera). |
| **MB_rd_STR** | function MB_rd_STR(indx:cardinal):string; |

| Nazwa | Opis |
|---|---|
| | Funkcja zwraca ramkę powrotną w postaci tekstowej (bez bajtów sumy kontrolnej) – jako ciąg bajtów zapisanych szesnastkowo. np. 'FF 05 00 00 FF 00'. |
| **MB_RxCount** | function MB_RxCount():cardinal;<br><br>Funkcja zwraca ilość bajtów w ramce powrotnej – bez dwóch bajtów sumy kontrolnej. |

Niżej przykład użycia popularnego modułu z dwoma wejściami i dwoma wyjściami. Program w zamkniętej pętli załącza i wyłącza naprzemiennie przekaźniki, natomiast stan wejść przepisuje na bity 30,31 rejestru MEMO. Program znajduje się w przykładach dołączonych do programu. Wyjścia sterowane są pojedynczo za pomocą komendy „5” Modbus, natomiast wejścia odczytywane komendą „2” i odczytany stan bitów kopiowany do 10 i 11-tego rejestru zmiennych. Moduł ma przypisany adres 255 ($FF).



```
//=============================================
//
//                          Przykład użycia MC
//
//=============================================

procedure MB_OUT(num:byte; state:boolean);
begin
  MB_InitFrame($FF,5);
  MB_wr_word(num);
  if state then MB_wr_word($FF00) else MB_wr_word($0000);
  MB_Send();
end;

procedure MB_READ_IN();
begin
   MB_SendStr('FF 02 00 00 00 08');
   SetVar(10, boolean(MB_rd_byte(3) and $01));
   SetVar(11, boolean(MB_rd_byte(3) and $02));
end;

%
:LOOP
  MB_READ_IN();
  SetMemo(30, GetVar(10));
  SetMemo(31, GetVar(11));
  MB_OUT(0, TRUE);
  MB_OUT(0, FALSE);
  MB_OUT(1, TRUE);
  MB_OUT(1, FALSE);
  Gotolabel('LOOP');
```

### *Funkcje związane z drukowaniem komunikatów w okienku MDI*

| Nazwa | Opis |
|---|---|
| **Log** | procedure Log(txt:string)<br><br>Dodaje w oknie komunikatów nową linię z podanym tekstem. |
| **LogClr** | procedure LogClr()<br><br>Czyści okno komunikatów. |
| **LogShow** | procedure LogShow()<br><br>Otwiera okienko MDI jeśli nie jest widoczne. |
| **FloatToStr** | function FloatToStr(v:Extended):string<br><br>Konwertuje zmienną typu Extended na tekst. |
| **IntToStr** | function IntToStr(v:integer):string<br><br>Konwertuje zmienną typu integer lub cardinal na tekst. |
| **VarToStr** | function VarToStr(num:cardinal):string<br><br>Konwertuje zmienną o danym numerze na tekst. Typ zmiennej rozpoznawany jest automatycznie. |
| **Time** | function Time():string<br><br>Zwraca aktualny czas w postaci tekstu. |

## Komentarze

Możemy stosować dwa rodzaje komentarzy dla pojedynczej linii „//” lub dla większej ilości linii nawiasy klamrowe: „{„ oraz „}”. Znaki klamry muszą być jednak jako pierwsze znaki w linii nie licząc spacji i tabulacji. Natomiast komentarz typu „//” może być w dowolnym miejscu linii tekstu.

## Pozostałe zadeklarowane stałe

| Nazwa | Opis |
|---|---|
| **SMT_HEIGHT** | Wysokość czujnika długości narzędzia zadeklarowana w ustawieniach. |
| **SMM_HEIGHT** | Wysokość czujnika wysokości materiału zadeklarowana w ustawieniach. |

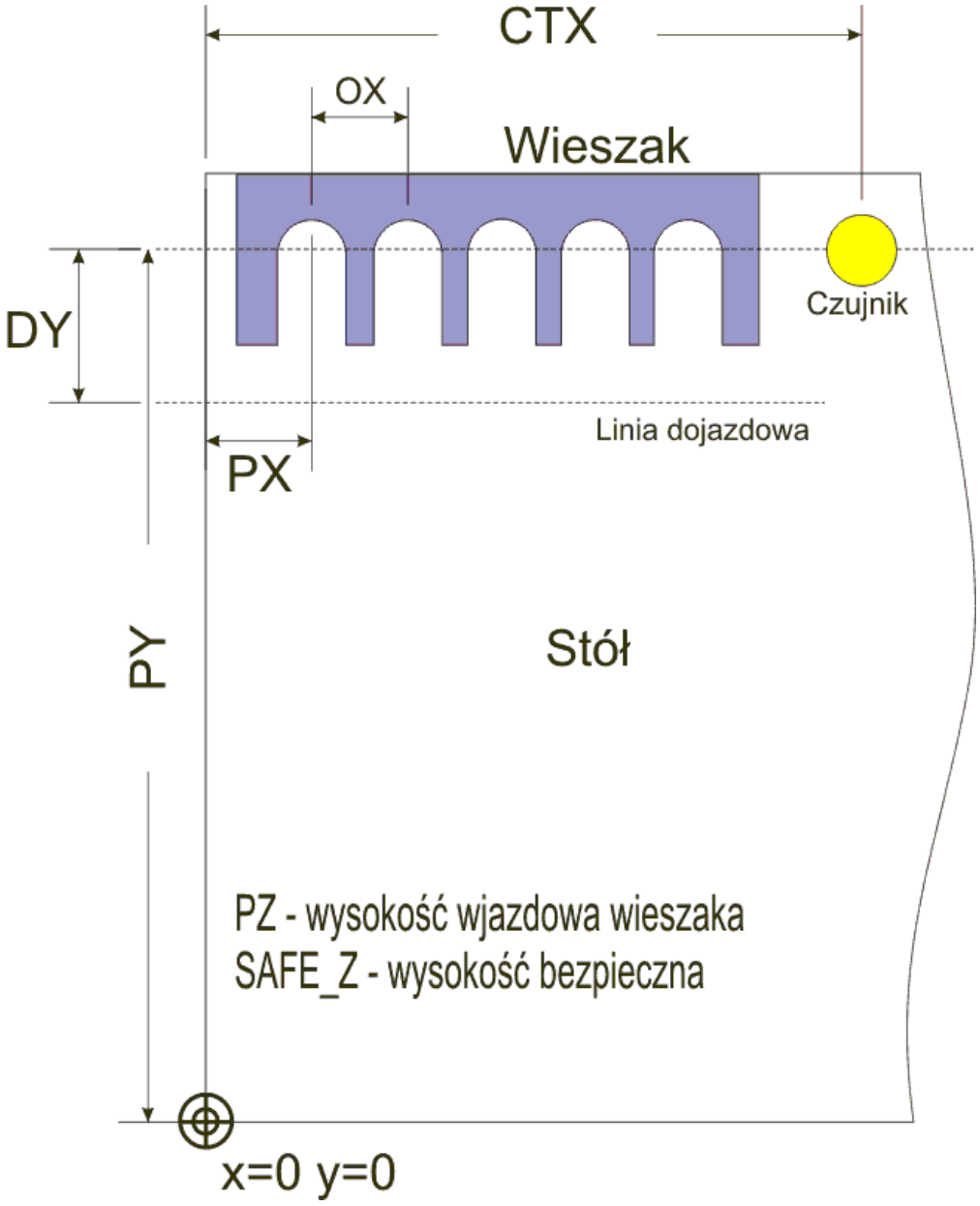
# Przykłady

## *Makro wymiany narzędzia M6*

Załączenie automatycznej wymiany narzędzia dokonuje się w ustawieniach zakładka „G kod / Zmiana narzędzia (M6)”. Należy tam zaznaczyć „Makro M6”. Makro można także wywoływać z okienka MDI np. wpisując „M6 T6” zmienimy narzędzie na T6.

Przedstawione niżej makro można znaleźć w katalogu macros/exaples. Aby poeksperymentować z nim należy je przenieś do katalogu macros/.

Rozmieszczenie elementów: wieszak na oprawki narzędzi zamocowany u góry stołu, obok czujnik do pomiaru długości narzędzia.



```
Const                          // Deklaracja stałych
  F_NAJAZD=400;
  F_ZJAZD=800;
  F_MESS=600;
  PX=20.0;                     // deklaracja wymiarów do powyższego rysunku
  PY=270.0;
  PZ=80.0;
  DY=30.0;
  Z_UP=40.0;
  OX=40.0;
  CTX=250;
  CT_HEIGHT=29.0;        // wysokość czujnika długości narzędzia
  ZAMEK=6;               // Numer bitu dla zamka
  TOOL_IN_HANDLE=6;  // Wejście wykrywające narzędzie w uchwycie

  USER_POS_X=0.0;        // Pozycja dojazdowa narzędzia w sytuacji uruchomienia makra ręcznie
  USER_POS_Y=0.0;        // ….

  ACT_TOOL=0;            // Indeks zmiennej pamiętającej numer aktualnego narzędzia
  REQ_TOOL=1;            // Indeks zmiennej pamiętającej numer żądanego narzędzia

%
  if OutputN('SPINDLE') then UserError('Wrzeciono jest załączone !');
  SetZ(PosSafe); ExeMove(0);             // Uniesienie „Z” na wysokość bezpieczną
  SetVar(ACT_TOOL,GetActTool);           // Zapamiętanie aktulnego narzędzia
  if UserExe then SetVar(REQ_TOOL, GetArgDef('T',0)) else SetVar(REQ_TOOL,GetReqTool);
  if (GetToolSlot(GetVar(REQ_TOOL))<0) then UserError('Narzędzie T'+IntToStr(GetVar(REQ_TOOL))+' nie
        ma przypisanego slotu !');
  if (GetToolSlot(GetVar(ACT_TOOL))<0) then UserError('Narzędzie T'+IntToStr(GetVar(ACT_TOOL))+' nie
        ma przypisanego slotu !');
  if (GetVar(ACT_TOOL)=0) and (GetVar(REQ_TOOL)=0) then return;
  if (GetVar(ACT_TOOL)=0) then GotoLabel('M6_GET');  // Skok do M6_GET jeśli w uchwycie nie ma
        narzędzia
  if (GetVar(ACT_TOOL)=GetVar(REQ_TOOL)) then GotoLabel('M6_MESS');

:M6_PUT
  SetX(PX+(OX*(GetToolSlot(GetVar(ACT_TOOL)) - 1))); SetY(PY-DY); ExeMove(0);
  SetActTool(0);                 // Odwołanie korekcji „Z” -- BARDZO WAŻNE !!!
  SetZ(PZ); ExeMove(0);          // Uchwyt na pozycję wjazdową „Z”
  SetY(PY); ExeMove(0);          // Wjazd w uchwyt
  SetMemo(ZAMEK, TRUE);                // Zwolnienie zamka
```

```
  WaitTime(0.2);                    // Oczekiwanie 0.2 sek.
  SetZ(PZ+Z_UP); ExeMove(0);        // Wyjazd w górę
  SetMemo(ZAMEK, FALSE);            // Załączenie zamka

:M6_GET
  if (Input(TOOL_IN_HANDLE)) then UserError('Narzędzie nadal w uchwycie !');
  if (GetVar(REQ_TOOL)=0) then GotoLabel('M6_ENDPROCES');
  SetX(PX+((GetToolSlot(GetVar(REQ_TOOL))-1)*OX)); SetY(PY); ExeMove(0); // Dojazd nad odpowiedni
        slot
  SetMemo(ZAMEK, TRUE);             // Zwolnienie zamka
  SetZ(PZ); ExeMove(0);             // „Z” na pozycję wyjazdową (najazd na narzędzie)
  SetMemo(ZAMEK, FALSE);            // Załączenie zamka
  WaitTime(0.2);
  SetY(PY-DY); ExeMove(0);          // Wyjazd z uchwytu
  SetActTool(GetVar(REQ_TOOL));     // Ustalenie narzędzia jako aktualnego
  SetZ(PosSafe); ExeMove(0);        // Uniesienie na wysokość bezpieczną

:M6_MESS
  if (not Input(TOOL_IN_HANDLE)) then UserError('Brak narzędzia w uchwycie !');
  if ValidTool(GetActTool) or UserExe then GotoLabel('M6_ENDPROCES');
  SetX(CTX); SetY(PY);  ExeMove(0);
  RefZ(-1); ExeRef(PROBE_ON, F_MESS);
  SetToolPos(CT_HEIGHT);

:M6_ENDPROCES
  SetZ(PosSafe); ExeMove(0);
  if UserExe then begin SetX(USER_POS_X); SetY(USER_POS_Y); ExeMove(0); end;
  return;
```

## *Makro G0*

Makro można znaleźć w katalogu Macros/Elcosimo/. Aby przetestować w okienku MDI można wpisać np. „G0 X0 Y0” +ENTER .

```
//=============================================================================
//  Makro symulujące kod G0. Parametry:
//  X Y Z A - Docelowa pozycja osi.
//=============================================================================
%
  if IsArg('X') then SetX(GetArg('X')+PosMaterial(AXIS_X));
  if IsArg('Y') then SetY(GetArg('Y')+PosMaterial(AXIS_Y));
  if IsArg('Z') then SetZ(GetArg('Z')+PosMaterial(AXIS_Z));
  if IsArg('A') then SetA(GetArg('A')+PosMaterial(AXIS_A));
  ExeMove(0);
  Return;
```

## *Makro do testowania położenia maszyny*

Makro można znaleźć w katalogu Macros/Elcosimo/

```
//============================================================================
// Makro do sprawdzania położenia maszyny
// ( odchyłek od poprzedniej jazdy referencyjnej )
//============================================================================
const
F_NAJAZDU = 0;    // Prędkość najazdu na czujnik  ( 0 oznacza max )
F_REF = 600;
DIR_X = 1;        // Kierunki szukania krańcówek HOME 1 w prawo, -1 w lewo
DIR_Y = 1;        // ...
TOOL_MESS = TRUE; // Czy pomiar narzędzia...
FAST_DOWN = 0.0;     // Zakres szybkiego ruchu w dół do czujnika

%
  RefZ(1); ExeRef(HOME_ON_SOFT, F_NAJAZDU);
  RefX(DIR_X); RefY(DIR_Y); ExeRef(HOME_ON_SOFT, F_NAJAZDU);
  RefX(-1*DIR_X); RefY(-1*DIR_Y); RefZ(-1); ExeRef(HOME_OFF, F_REF);

  LogShow;
  Log('=========================================');
  Log('['+Time+'] Test położenia osi...')
  Log('=========================================');
  Log('dX = '+FloatToStr(PosX - PosBase(AXIS_X)));
  Log('dY = '+FloatToStr(PosY - PosBase(AXIS_Y)));
  Log('dZ = '+FloatToStr(PosZ - PosBase(AXIS_Z)));

  SetX(PosToolSensor(AXIS_X)); SetY(PosToolSensor(AXIS_Y));  ExeMove(0);
  if not TOOL_MESS then GotoLabel('END_PROCES');

  SetZ(PosZ - FAST_DOWN); ExeMove(0);
  Softlimit(FALSE);
  RefZ(-1); ExeRef(PROBE_ON, F_REF);
  Log('dT = '+FloatToStr(PosZ - PosToolSensor(AXIS_Z)));
  SetZ(PosSafe); ExeMove(0);
  Softlimit(TRUE);

:END_PROCES
  Log('=========================================');
```

**PPHU ELCOSIMO**
**Andrzej Woźniak**
**ul. Zielona 1B**
**62-110 Damasławek**