Skrypty sterujące w systemie Golem OEE

Spis treści

1 Wstęp - Wersje

System golem OEE SV jest w szerokim zakresie konfigurowalny. Niestety, niezależnie od tego jak bogata będzie lista ustalanych parametrów zawsze spotkamy się z taką pożądaną konfiguracją której nie będziemy mogli zrealizować.
Dlatego wprowadzono możliwość tworzenia skryptów sterujących dla stacji dzięki którym można znacznie poszerzyć możliwości systemu.
Skrypt to program działający w obrębie stacji zbierania danych który może manipulować dostępnymi w niej danymi.

Skrypt sterujący jest prostym programem napisanym w języku Pascal. Wymaga co prawda znajomości tego języka ale tylko w stopniu absolutnie podstawowym – analiza tego dokumentu powinna wystarczyć każdemu kto kiedykolwiek napisał jakikolwiek program w dowolnym języku.
Ponadto w Internecie znajduje się mnóstwo kursów Pascala. Dlatego pisząc niniejszy dokument zakładamy wiec że czytelnik zna podstawy programowania.

Ważne !
Konfigurując system, ustalając poszczególne parametry w definicji modelu czy nadzorcy mamy prawo oczekiwać aby możliwe ustawienia były LOGICZNE.
Pisząc skrypt sterujący to my jesteśmy odpowiedzialni za logikę programu i musimy mieć pełną świadomość tego co robimy.

Nieumiejętne posługiwanie się mechanizmem skryptów może spowodować destabilizację systemu.

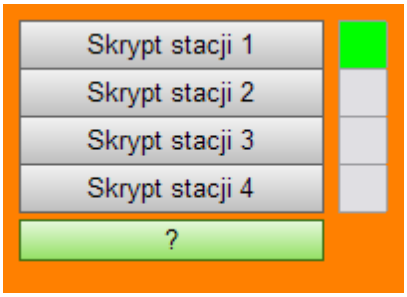
Skrypty dostępne są od wersji 2.3.16
Aktualny dokument dotyczy wersji 2.4.19
Skrypty nie są dostępne w wersji OEM

Historia wersji
2.4.19
- dodano rozkaz NextOrder

2 Budowa i kompilacja skryptów , Edytor skryptów

2.1 Aktywacja i kompilacja skryptu

W oknie programu konstruktora widoczny jest panel z przyciskami otwierającymi edytor skryptu dla poszczególnych stacji.
Obok przycisków umieszczono indykatory pokazujące czy skrypt jest aktywny (załączony)

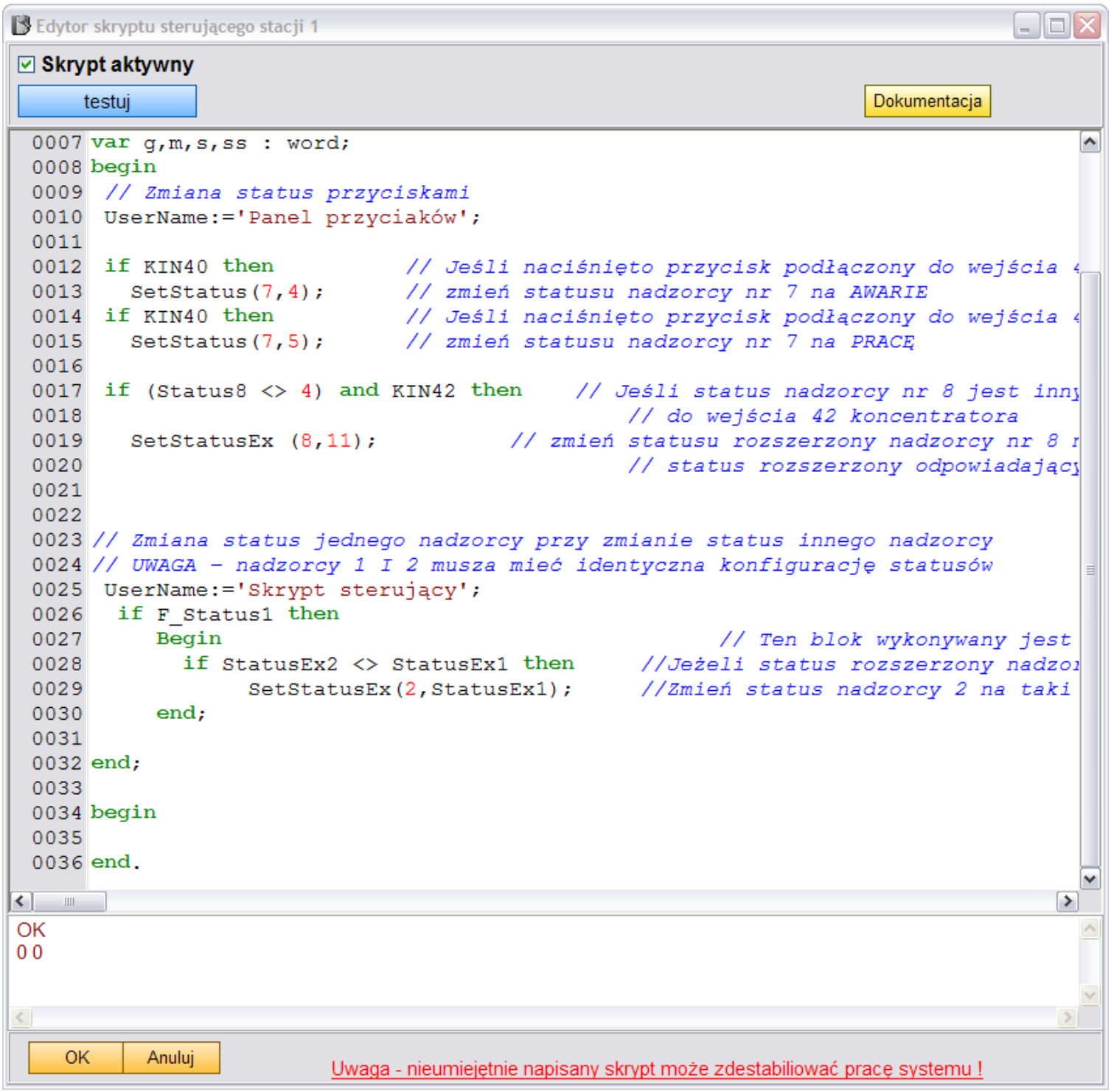


Odpowiednim przyciskiem otwieramy edytor dla określonej stacji a w nim głównym ustawieniem jest to czy skrypt jest aktywny.

Skrypt ładowany jest i kompilowany w momencie uruchomienia stacji. Jeżeli skrypt będzie zawierał błędy to stacja się uruchomi ale nie wystartuje i pojawi się okno z podglądem kodu i listą błędów w tym kodzie. Należy wtedy usunąć błędy w edytorze konstruktora i ponownie uruchomić stację.

UWAGA – w przeciwieństwie do innych zmian w konfiguracji nie jest możliwa zmiana skryptu przez przeładowanie stacji – trzeba ja zamknąć i ponownie uruchomić.

Edytor skryptu:



2.2 Budowa skryptu

Skrypt sterujący to program napisany w języku Pascal. Każda stacja może mieć własny skrypt sterujący który działa w ramach jej zakresu zmiennych.
Znaczy to że skrypt dla stacji 1 nie ma dostępu do wejść koncentratora ani zespołów nadzorców obsługiwanych przez inną stację.

Skrypt składa się z dwu części: jedna która jest wykonywana jeden raz po uruchomieniu programu i druga – procedura Task która wykonywana jest cyklicznie co około 200 ms.
Budowa skryptu:

```
procedure Task; // procedura uruchamiana co 200 ms
Begin
  // wszystkie rozkazy które znajdują się pomiędzy tymi słowami rozkazami początku i końca bloku
  // wykonywane będą co 200 ms
```

```
end;

Begin // początek programu
  // wszystkie rozkazy które znajdują się pomiędzy tymi słowami rozkazami początku i końca bloku
  // wykonywane zostaną jeden raz po uruchomieniu programu
end.
```

Procedura TASK może zawierać dowolny kod realizujący wiele czynności ale pamiętajmy że nie może c
Kompilator przetłumaczy nam i pozwoli wykonać poniższą procedurę :

```
procedure TASK;
begin
 repeat
 until not KIN1;
end;
```

Ale konstrukcja taka jest niedopuszczalna gdyż może zatrzymać pracę skryptu - jeżeli nie będzie z
koncentratora to procedura TASK zamknie się w "martwej" pętli.

UWAGA
Pod żadnym pozorem nie należy zmieniać nazwy procedury TASK

W starszych bazach może znaleźć się zapis innego formatu albo edytor po uruchomieniu może być pus
W takim przypadku należy w pierwszej kolejności utworzyć procedurę TASK i główną procedurę zwraca

3 Zmienne i flagi

Rodzaje zmiennych

- zmienne systemu Golem, np. KIN1 – pierwsze wejście koncentratora czy SVStatus1 – numer statusu nadzorcy SV1
- Flagi - specjalne zmienne binarne ustawiane wewnątrz programu i kasowane po każdym wykonaniu skryptu
- zmienne pomocnicze globalne – zmienne pomocnicze które możemy użyć w naszym skrypcie takie jak np. Vbit1 czy Vint1
- zmienne globalne zadeklarowane w kodzie skryptu na jego początku, przed słowami procedure task; poprzedzone słowem kluczowym Var,
- zmienne lokalne dla procedury Task

Zmienne zadeklarowane w skrypcie przydać się mogą bardziej zaawansowanym użytkownikom. Dodatkowo można też skorzystać ze zmiennych systemowych takich jak np. zmienna NOW która reprezentuje aktualny czas (typ TDateTime)

Zmienne dzielimy też (szczególnie zmienne systemu golem) na takie które są do odczytu i zapisu oraz takie które są tylko do odczytu co zostanie wyraźnie zaznaczone przy ich opisie.

WAŻNE:
Zmienne systemu golem muszą mieścić się w ramach zmiennych obsługiwanych przez stacje zbierania danych.
Jeżeli w skrypcie dla stacji nr 1 użyjemy np. zmiennej INSV130 to kompilator zasygnalizuje błąd. Zmienna INSV130 reprezentuje stan wejścia niefiltrowanego przypisanego do nadzorcy SV130 a ten jak wiadomo należy do zakresu nadzorców stacji numer 2.

Niektóre zmienne systemu Golem które są tylko do odczytu możemy zmienić za pomocą odpowiednich procedur, np. zmienna order_cp zwraca stan licznika produktu i jest zmienną tylko do odczytu – możemy jednak przypisać jej nową wartość za pomocą procedury Set_Order_CP.

3.1 Zmienne

xx w opisie zmiennej oznacza numer nadzorcy SV. np. INSV**xx** oznacza że zmienna może mieć nazwę od INSV1 do INSV512.
jeśli w opisie nie zaznaczono inaczej to zmienne są tylko do odczytu.
Niektóre zmienne które są do odczytu zapisywać można za pomocą procedur.

ZMIENNA	***Typ***	***opis***
KIN1 do K**IN128**	boolean	Wejścia koncentratora. Wejścia 1 do 64 reprezentują fizyczne wejścia poszczególnych modułów, wejścia 65 do 128 to wejścia logiczne. **Zmienne KIN1 do KIN64 i KIN112 do KIN127 należy traktować jako zmienne tylko do odczytu.** Przypisanie im wartości nie spowoduje błędu ale zostanie zignorowane.

INSVxx	boolean	Wejście filtrowane nadzorcy SV. Zmienna reprezentująca stan wejścia skojarzonego z nadzorcą. Kiedy konfigurujemy nadzorcę SV1 to wcale nie musimy mu przypisać wejścia koncentratora nr 1 a możemy przypisać inne – np. wejście 10. Zmienna INSV1 będzie wiec reprezentować stan wejścia skojarzonego z 1 nadzorcą, czyli wejścia 1
INFSVxx	boolean	Jak zmienne INSV ale reprezentują stan wejść filtrowanych.
WORKxx	boolean	Specjalna zmienna informująca o tym że nadzorca nalicza czas pracy
STATUSxx	byte	Zmienna zwraca numer aktualnego statusu nadzorcy : 0-Postój Planowany 1-Postój Nieplanowany 2-Przezbrajanie 3-Ustawianie 4-Awaria 5-Praca
STATUSEXxx	byte	Numer statusu rozszerzonego nadzorcy – numer odczytujemy w konstruktorze w edytorze statusów
DelayOnxx		Czas filtru opóźnienia załączenia
DelayOffxx		Czas filtru opóźnienia wyłączenia
DelayENxx		Czas filtru blokady
ORDER_CCxx	integer	Licznik cykli aktualnego zlecenia
ORDER_CPxx	integer	Licznik produktu aktualnego zlecenia zaokrąglony do wartości całkowitej
ORDER_CPFxx	Double	Licznik produktu aktualnego zlecenia jako liczba rzeczywista
ORDER_TARGETxx	Integer	Ilość zamówiona dla aktualnego zlecenia
ORDER_MULTIPLExx	Integer	Parametr gniazdo zaokrąglony do liczby całkowitej
ORDER_MULTIPLEFxx	Double	Parametr gniazdo jako liczba rzeczywista
ORDER_IDxx	Integer	Numer ID aktualnego zlecenia. 0 jeśli nie ma aktywnego zlecenia
ORDER_OCCxx	Integer	Optymalny czas cyklu zlecenia
ORDER_DEFECTxx	Integer	Licznik braków od początku zlecenia
CCxx	integer	Aktualny czas cyklu w 10 częściach sekundy (10 to 1 sekunda)
UserName	String	Zmienna wykorzystywana przez te procedury które dokonują operacji w systemie znakowanych operatorem. Np. procedura SetStatus zmienia status a jak wiadomo operacja ta jest logowana w rejestrze zdarzeń. W kolumnie Operator zobaczymy wartość zmiennej UserName
		Zmienne pomocnicze i zmienna timer są zmiennymi do zapisu i odczytu
Vbit1 do **Vbit64**	Boolean	Zmienne pomocnicze typu binarnego
Vint1 do **Vint64**	Integer	Zmienne pomocnicze typu integer (liczba całkowita)
Vr1 do **Vr64**	Real	Zmienne pomocnicze typu Real (rzeczywiste)
Timer1 do **Timer64**	Integer	Zmienna timer co 1 sekundę jest pomniejszana jeśli jest większa od zera zobacz Timery
RestartTime	Integer	Czas (w sekundach) jaki upłyną od zakończenia pracy stacji do ponownego uruchomienia

3.2 Flagi

Flagi to specjalne zmienne które są ustawiane wewnątrz programu stacji w wyniku jakichś zdarzeń a kasowane każdorazowo po wykonaniu skryptu.

Dzięki flagom możemy wykonać jakieś polecenie tylko jeden raz po jakimś zdarzeniu np. polecenie:

```
if f_order1 then Reset(3)
```

spowoduje skasowanie wszystkich liczników kasowalnych nadzorcy SV3 po zmianie zlecenia w nadzorcy SV1.

Dostępne są następujące flagi:

Flaga	*Typ*	*opis*

F_INSVxx	boolean	Flaga ustawiana zmianą stanu (załączeniem) wejścia filtrowanego przypisanego do wskazanego nadzorcy SV.
		jeżeli zastosujemy zapis:
		if F_INSV1 then
		`begin`
		`inc(Vint1); //zwiększ o jeden zmienną pomocniczą Vint1`
		`end;`
		to zmienna Vint1 będzie zwiększana (rozkaz wykonywany) przy każdym zboczu narastającym odfiltrowanego sygnału wejścia ustalonego w konfiguracji jako wejście główne nadzorcy SV1

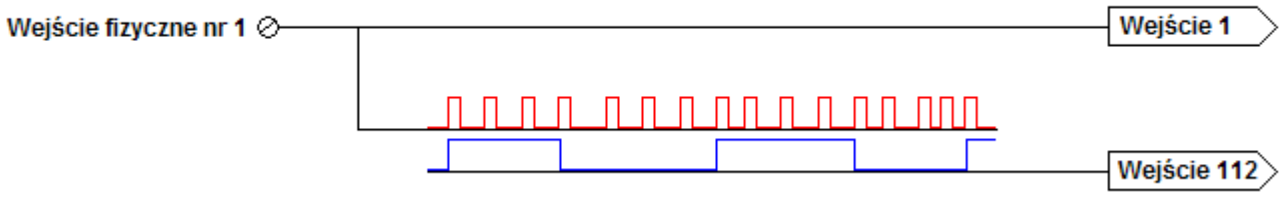
F_ORDERxx	boolean	Flaga sygnalizuje zmianę zlecenia we wskazanym nadzorcy (lub skasowanie liczników kasowalnych jeżeli nadzorca jest skonfigurowany do pracy bez zlecenia)
F_ENDORDERxx	boolean	Flaga sygnalizuje zakończenie zlecenia
F_STATUSxx	boolean	Flaga sygnalizuje zmianę statusu
F_OPERATORxx	boolean	Flaga sygnalizuje zmianę operatora
F_TIMER1 do F_TIMER64	boolean	Flaga ustawiana po zakończeniu odliczania czasu przez timer zobacz Timery
F_SEC	boolean	Flaga ustawiana co 1 sekundę

3.3 Wejścia logiczne

Uważny czytelnik zauważy że zmienne KIN reprezentujące wejścia koncentratora mają numerację od 1 do 127 a przecież koncentrator ma maksymalnie 64 wejścia.

Wejścia 65 do 127 są wejściami logicznymi z tym że wejścia o numerach 112 do 127 są odpowiednikami wejść 1 do 16 z podziałem przez 4 a wejścia 65 do 111 możemy wykorzystać w dowolny sposób.

Pracę i zastosowanie w wejść 112 do 127 wytłumaczymy na przykładzie wejścia 112 - pozostałe działają analogicznie.



Sygnał który zmienia się na 1 wejściu koncentratora widoczny jest w całym systemie jako wejście 1. Czasami mamy do czynienia z bardzo krótkimi sygnałami,np z wyjść liczników energii elektrycznej. Sygnały takie mogą zostać potraktowane przez

filtry przeciw zakłóceniowe koncentratora jako sygnały nieporządkowane i mogą zostać niepoliczone albo niektóre sygnały mogą zostać zagubione.
Dlatego koncentrator dzieli częstotliwość sygnału wejściowego przez 4 i podaje ten stan na wejście logiczne 112. Jeżeli zamiast wejścia 1 ustawimy wejście 112 jako główne wejście sterujące nadzorcą to nadzorca taki będzie widział co 4 sygnał względem wejścia fizycznego.

Jak wykorzystać pozostałe wejścia logiczne ?
Możemy np. ustawić w nadzorcy wejście 65 i spowodować aby sygnał z wejścia 10 koncentratora pojawiał się na nim tylko wtedy gdy maszyna reprezentowana przez nadzorcę numer 5 ma status praca:

```
procedure TASK;
begin
 KIN65 :=  KIN10 and ( status5 =5 );
end;
```

Można też stworzyć maszynę wirtualną która będzie pracowała wtedy gdy będzie pracować którąkolwiek z maszyn reprezentowanych przez nadzorców sv1 do sv5

```
procedure TASK;
begin
 KIN65 :=  Work1 or Work2 or Work3 or Work4 or Work5;
end;
```

Zmienna Work ma wartość True wtedy gdy nadzorca nalicza czas pracy.

3.4 Specjalny dostęp do zmiennych

Powiedzmy że mamy następujący skrypt

```
procedure Task;
Begin
 KIN80:= (Status1 = 5) and (order_id1 <>0) and INSV1;
 KIN81:= (Status2 = 5) and (order_id2 <>0) and INSV2;
 KIN82:= (Status3 = 5) and (order_id3 <>0) and INSV3;
 KIN83:= (Status4 = 5) and (order_id4 <>0) and INSV4;
 KIN84:= (Status5 = 5) and (order_id5 <>0) and INSV5;
 KIN85:= (Status6 = 5) and (order_id6 <>0) and INSV6;
end;
```

Nie jest istotne co ten skrypt robi, istotne jest to że robi to 6 razy dla 6 różnych nadzorców. A jest to bardzo prosta logiczna formuła.

A gdybyśmy mieli bardziej złożony program a maszyn nie 6 a 15 ?
Na taką właśnie okoliczność poza odwołaniem się do zmiennych po ich pełnej nazwie wraz z numerem zmiennej który najczęściej jest numerem
nadzorcy dodano specjalne funkcje i procedury do obsługi zmiennych gdzie parametrem jest jej numer.

Ten sam skrypt mógłby wyglądać tak

```
procedure Ustaw(nrv:integer);
var b:boolean;
Begin
 b:= ( GetTabInt(T_Status,nrv)=5) and ((GetTabInt(T_Order_Id,nrv)<>0) and GetTabBit(T_INSV,nrv);
 SetTabBin( T_KIN, nrv +79, b);
end;

procedure Task;
var n:integer;
Begin
 for n:= 1 to 6 do Ustaw(n);
end;
```

Jak widać zdefiniowano dodatkową procedurę Ustaw która wywoływana jest 6 razy w pętli z kolejnym parametrem.
W procedurze Ustaw zastosowano funkcje i procedury które pozwalają na dostęp do zmiennych po ich NUMERZE.

Na dostęp do zmiennych z wykorzystaniem ich numerów pozwalają nam:

- funkcja **GetTabBit** (nazwa_zmiennej, numer_zmienej) zwracająca stan wskazanej zmiennej typu boolean
- funkcja **GetTabInt** (nazwa_zmiennej, numer_zmienej) zwracająca stan wskazanej zmiennej typu integer
- funkcja **GetTabR** (nazwa_zmiennej, numer_zmienej) zwracająca stan wskazanej zmiennej typu real (double)
- procedura **SetTabBit** (nazwa_zmiennej, numer_zmienej, wartość_do zapsiu) która ustawia wskazaną zmienną typu boolean
- procedura **SetTabInt** (nazwa_zmiennej, numer_zmienej, wartość_do zapsiu) która ustawia wskazaną zmienną typu integer

Nazwa zmiennej jest taka sama jak nazwa zmiennej z numerem ale zamiast numeru na końcu ma przedrostek **T_** na początku, np.
T_Vbit jako substytut zmiennej Vbitxx, T_StatusEx jako substytut zmiennej StatusEXxx, T_Order_Target jako substytut zmiennej Order_Targetxx itd.

Należy pamiętać o właściwym doborze funkcji do odczytu zmiennej względem jej typu i o tym że procedury SetTab mogą ustawić tylko te zmienne które mogą być zapisywane.
Za pomocą funkcji GetTabBit można też odczytać wartość flag np

```
Function WejscieLogiczne( nsv, nwl:integer) : boolean;
begin
result:=false;
 if GetTabBit(T_F_Order, nsv) then   // Jeśli zmiana zlecenia nadzorcy o numerze wskazanym zmienn
   begin
      SetTabBit( T_KIN, nwl,true);   // Ustaw logiczne wejście koncentratora o numerze wskazanym
      result:= true;                 // i ustaw wartośc zwracana przez funkcję na true
   end;
 if GetTabBit(T_F_EndOrder,nsv) then // Jeśli koniec zlecenia nadzorcy o numerze wskazanym zmienn
   begin
      SetTabBit( T_KIN, nwl,false);  // Skasuj logiczne wejście koncentratora o numerze wskazanym
   end;
end;
```

Zwróćmy uwagę na rozkaz ustawiający zmienną wejścia koncentratora. Poza 64 fizycznymi wejściami które oczywiście są tylko do odczytu mamy do dyspozycji wejścia
logiczne (wirtualne) którym możemy przypisać jakąś wartość, zobacz Zmienne

4 Timery

W skrypcie można użyć 64 timery.

Timer działa tak że gdy do zmiennej timer np. Timer1 wpiszemy jakąś wartość to program co sekundę będzie ją pomniejszał tak długo aż osiągnie wartość 0.
Gdy ją osiągnie ustawi flagę F_Timer1.
Jeśli chcemy przerwać pracę timera ale bez ustawiania flagi wywołujemy procedurę ResetTimer(nr);

Przykład użycia timera dla zmiany statusu na postój nieplanowany po 120 sekundach braku aktywności maszyny

```
procedure Task;
Begin
  if Work1 then Timer1:= 120; // jeśli maszyna pracuje cały czas wpisuj 120 sekund
  if F_Timer1 then            // jeśli skończył się czas (upłyneło 120 sekund)
    then SetStatus(1,1);      // zmień status na postój nieplanowany
end;
```

5 Procedury i Funkcje

Dla obsługi systemu zdefiniowano kilkanaście procedur i funkcji które wykonują określone czynności lub zwracają określone dane.
Możemy też zdefiniować własne procedury i funkcje lokalne zgodnie z zasadami języka Pascal

5.1 Blokada czasowa

Niektóre z procedur takie jak np. SetStatus posiadają wbudowany mechanizm blokowania ich kolejnego wykonania przez czas 5 sekund.
Działa to tak że po wywołaniu procedury przez 5 sekund nie można jej wywołać ponownie, to znaczy wywołać można ale nie będzie żadnej reakcji.
Oczywiście możemy wywołać w tym czasie tę samą procedurę z innym parametrem (dla innego nadzorcy).
Mechanizm ten ma zabezpieczyć program przed wielokrotnym wywołaniem procedury która ma wykonać np wpis do rejestru zdarzeń.

Rozważmy taką sytuację

Chcemy aby po naciśnięciu przycisku podłączonego do wejścia koncentratora, do rejestru zdarzeń dodać komunikat "naciśnięto przycisk"
Napiszemy więc skrypt:

```
Procedure TASK;
Begin
 if KIN10 then ToLog(1,'naciśnięto przycisk);
End;
```

Pamiętajmy jednak że procedura TASK wykonywana jest ok 10 razy na sekundę i podczas naciśnięcia przycisku do rejestru zdarzeń dodano by kilka
wpisów.

Dlatego procedura ToLog ma zabezpieczenie czasowe które do tego nie dopuści - nawet jeśli będziemy trzymać cały czas przycisk naciśnięty to zapis do rejestru nie nastąpi częściej jak co 5 sekund.

5.2 Zmiana statusu

Za pomocą skryptu możemy zmienić status lub status rozszerzony
Dostępne są trzy procedury zmiany statusu z poziomu skryptu sterującego:SetNoStatus, SetStatus I SetStatusEx.

SetNoStatus

Procedura zmienia tylko numer głównego statusu bez logowania (dodawania informacji do rejestru zdarzeń) tej operacji.
Numer aktualnego statusu możemy odczytać ze zmiennej **STATUSxx**
Zmiana statusu tym rozkazem nie powoduje zmiany flagi **F_STATUSxx.**
Jest aktywna tylko wtedy gdy w modelu ustawiony jest sposób starowania statusem na zmianę w skrypcie.

```
procedure SetNoStatus(sv,nr:integer);   przykład składni: SetNoStatus(1,2);
```

parametry: sv – numer nadzorcy, nr – numer statusu który ma zostać ustawiony

SetStatus

Procedura SetStatus zmienia status w taki sam sposób jak by to operator zmienił status. Może ona zostać użyta gdy w modelu jest ustawiony sposób sterowania statusu z panelu operatora lub ze skryptu. W pierwszym przypadku możliwa jest wymiennie zmiana statusu przez operatora i ze skryptu a drugim przypadku tylko skrypt może status zmienić.
Po zmianie statusu zostanie dodana informacja do rejestru zdarzeń.
Procedura posiada blokadę ponownego wykonania przez 5 sekund.

```
procedure SetStatus(sv,nr:integer);   przykład składni: SetStatus(1,2);
```

parametry: sv – numer nadzorcy, nr – numer statusu który ma zostać ustawiony

SetStatusEx

Procedura zmienia status rozszerzony w analogiczny sposób jakby to wykonał operator. Procedura SetStatusEx wykonana zostanie tylko wtedy w konstruktorze ustawiono status rozszerzony.
Procedura posiada blokadę ponownego wykonania przez 5 sekund.

```
procedure SetStatusEx(sv,nr:integer); przykład składni: SetStatusEx(1,11);
```

parametry: sv – numer nadzorcy, nr – numer statusu rozszerzonego (odczytany w konstruktorze) który ma zostać ustawiony

Procedury SetStatus i SetStatusEx zapisują informacje o zmianie statusu do rejestru zdarzeń. Jednym z parametrów którym opatrywany jest taki zapis jest nazwisko operatora. W przeciwieństwie do manualnej zmiany statusu z terminala skrypt nie zna

operatora który np nacisną przycisk zmiany statusu. Możemy wiec przypisać zawartość kolumny operator w zmiennej UserName np. `UserName:= 'Zmiana przyciskiem;`

5.2.1 Przykładowy skrypt

```
procedure Task;
Begin

 // Zmiana status przyciskami
 UserName:='Panel przyciaków';

 if KIN40 then          // Jeśli naciśnięto przycisk podłączony do wejścia 40 koncentratora
   SetStatus(7,4);      // zmień statusu nadzorcy nr 7 na AWARIE
 if KIN40 then          // Jeśli naciśnięto przycisk podłączony do wejścia 41 koncentratora
   SetStatus(7,5);      // zmień statusu nadzorcy nr 7 na PRACĘ

 if (Status8 <> 4) and KIN42 then    // Jeśli status nadzorcy nr 8 jest inny niż AWARIA i naciśni
                                     // do wejścia 42 koncentratora
   SetStatusEx (8,11);              // zmień statusu rozszerzony nadzorcy nr 8 na
                                     // status rozszerzony odpowiadający 11 pozycji z listy statu


// Zmiana status jednego nadzorcy przy zmianie status innego nadzorcy
// UWAGA - nadzorcy 1 I 2 musza mieć identyczna konfiguracje statusów
 UserName:='Skrypt sterujący';
  if F_Status1 then
     Begin                                 // Ten blok wykonywany jest raz po zmianie flagi F_Stat
       if StatusEx2 <> StatusEx1 then      //Jeżeli status rozszerzony nadzorcy 2 inny niż nadzorc
            SetStatusEx(2,StatusEx1);      //Zmień status nadzorcy 2 na taki jak nadzorca 1
     end;

end;
```

5.3 Kasowanie liczników

Procedura Reset kasuje wszystkie liczniki kasowalne (liczniki zlecenia dla trybów uwzględniających zlecenie)

`procedure Reset(sv:integer)  przykład składni: Reset(1)`
parametry: sv – numer nadzorcy.
Procedura nie loguje czynności w rejestrze zdarzeń i nie zmienia żadnych ustawień.

Przykładowe zastosowanie:
Mamy trzy maszyny spięte w jedną linię produkcyjną. Pierwsza maszyna jest maszyną główną której nadzorca (SV1) obsługuje zlecenia produkcyjne
i dwie maszyny pomocnicze których nadzorcy (SV2 i SV3) skonfigurowano dla prostych trybów.
Zmiana zlecenia w pierwszym nadzorcy spowoduje skasowanie liczników w innych nadzorcach.

```
procedure Task;
Begin
  if F_ORDER1 then
      Begin // Ten blok wykonywany jest raz po zmianie flagi F_ORDER1 (zmianie zlecenia w SV1)
        Reset(2) //Kasowanie wszystkich liczników kasowalnych nadzorcy 2
        Reset(3) //Kasowanie wszystkich liczników kasowalnych nadzorcy 3
      end;
end;
```

5.4 Braki

Braki możemy odczytać ze zmiennej Order_Defectxx (stan licznika braków od zmiany zlecenia lub skasowania liczników w prostszych trybach).
Możemy też dodać (lub podając dodając wartość ujemną) określoną ilość braków do czego służą poniższe procedury:

```
procedure AddDefect(sv,x:integer)   przykład składni: AddDefect(1, 7)
procedure SetDefect(sv,x:integer)   przykład składni: SetDefect(1, 7)
```

parametry: sv – numer nadzorcy, x – ilość braków która zostanie dodana do liczników braków.

Procedura SetDefect różni się od procedury AddDefect tym że działa tak jak dodawanie braków przez operatora – po dodaniu braków zostaje dopisany komunikat do rejestru zdarzeń.
Procedura SetDefect korzysta z mechanizmu czasowej blokady wykonania procedur.

5.5 Display, SetLamp

```
procedure DISPLAY(sv,nr:integer;x:single); przykład składni: Display(5,2,Vint1);
```
parametry: sv – numer nadzorcy, nr – numer wyświetlacza (1 lub 2), x – wartość do wyświetlenia

Procedura DISPLAY wysyła daną do wyświetlenia w przeglądarce. Procedura ta jest dokładnie opisana w rozdziale Wyświetlacz

```
procedure SetLamp(sv,nr_color:integer;stat:boolean) przykład składni: SetLamp(1,1,true);
```
parametry: sv – numer nadzorcy, nr_color - numer koloru lampy: 1-zielony, 2- żółty, 3- czerwony, stat - stan wybranej lampy

Procedura SetLamp zapala/gasi odpowiedni segment lampy sygnalizacyjnej wybranego nadzorcy.
Tryb pracy lampy dla danego zespołu musi być ustalony na sterowanie skryptem.

Przykład zastosowania
```
procedure Task;
Begin
 // sterowanie lampą 7 nadzorcy
 // wejście 15 zapala segment zielony, wejście 17 segment żółty
 // jeśli wyłączone są wejścia 16 i 17 świeci segment czerwony
 SetLamp(7,1,KIn16);
 SetLamp (7,2,KIn17);
 SetLamp(7,3, (not KIn16) and (not KIn17);
end;
```

5.6 ToLog

Procedura ToLog pozwala dodać własny komunikat do rejestru zdarzeń.

```
procedure ToLog(sv:integer;ev:string); przykład składni: ToLog(1,'Naciśnięty przycisk START');
```
parametry: sv – numer nadzorcy, ev – tekst który zostanie wpisany do rejestru zdarzeń.

Jako operator podawany jest tekst przypisany do zmiennej *operator w zmiennej* *UserName*

Procedura posiada blokadę ponownego wykonania przez 5 sekund.

5.7 Andon

Skrypt pozwala na obsługę systemu Andon.

```
procedure SetAndon(sv,nr:integer); przykład składni: SetAndon(1,1);
procedure SetAndonVar(sv,nr:integer); przykład składni: SetAndonVar(1,1);
```

parametry: sv – numer nadzorcy, nr – Numer komendy (0 - odwołanie wezwania, 1 - wezwanie pomocy, 2- wezwanie pomocy technicznej)

Procedura SetAndon pozwala na programowe wywołanie lub odwołanie pomocy analogicznie jak wykonanie tego polecenia za pomocą panelu operatora.
Jako operator podawany jest tekst przypisany do zmiennej *operator w zmiennej* *UserName*
Procedura posiada blokadę ponownego wykonania przez 5 sekund.

Procedura SetAndonVar zmienia tylko wartość stanu wewnętrznej zmiennej zgłoszenia. Nie zostanie wygenerowane zdarzenie w rejestrze zdarzeń i rejestrze zdarzeń Andon.
Nazwa nadzorcy nie pojawi się na liście maszyn na zakładce andon przeglądarki. Jedynym efektem działania procedury SetAndonVar jest pojawienie pulsującej ikony
na panelach maszyn.

5.8 SetFiltr

Za pomocą zmiennych DelayOnxx, DelayOffxx i DelayENxx można odczytać ustawienia odpowiednio filtru opóźnienia załączenia, opóźnienia wyłączenia i blokady.
Ustawić wartość filtru można za pomocą procedury SetFiltr

```
procedure SetFiltr(sv,nr,time:integer)  przykład składni SetFiltr(1,1,10)
```
parametry: sv – numer nadzorcy, nr – numer filtru (0 – opóźnienie załączenia, 1 – opóźnienie wyłączenia, 2 – blokada), time – nowa wartość czasu.

5.9 SetTimeTT

W systemie dostępne są dwie metody wyznaczania czasu pracy. Pierwszy prosty gdzie czas pracy "biegnie" kiedy jest załączone wejście koncentratora i drugi który polega na doliczaniu ustalonego czasu po każdym impulsie na wejściu.
Czas ten możemy ustalić w dwojaki sposób: albo ustawić na stałe w konfiguracji modelu lub nadzorcy (czas Tt) albo ustalić jako parametr zlecenia (optymalny czas cyklu)

Za pomocą rozkazu SetTimeTT możemy zmienić czas Tt co ma oczywiście sens wtedy gdy jest on podstawą czasu danego nadzorcy.

```
procedure SetTimeTT(sv,time:integer)  przykład składni SetTimeTT(1,30)
```
parametry: sv – numer nadzorcy, time – nowa wartość czasu Tt

Poniższy przykład pokazuje jak możemy wybrać jedna z dwu podstaw czasu za pomocą wejścia:
gdy załączone jest wejście 50 koncentratora to czas Tt nadzorcy o numerze 25 będzie wynosił 5 sekund a gdy wejście jest wyłączone 20 sekund.

```
procedure TASK;
begin
  if kin50 then SetTimeTT(25,5) else SetTimeTT(25,20) ;
end;
```

```
Następny przykład pokazuje kodowanie dwujkowo 4 czasów za pomocą 2 wejść:
```

```
procedure TASK;
var tx:byte;
begin
if ( not kin50) and (not kin51) then tx:= 8;
if  kin50 and (not kin51) then tx:= 15;
if ( not kin50) and  kin51 then tx:= 23;
if kin50 and  kin51 then tx:= 35;

  SetTimeTT(25,tx) ;
end;
```

5.10 NextOrder

```
procedure NextOrder(sv:integer)  przykład składni NextOrder(1)
```
parametry: sv – numer nadzorcy

Specjalny rozkaz którego wywołanie realizuje następujące czynności

- zamyka zlecenie jeśli jest otwarte
- wyszukuje z listy przygotowanych zleceń znajdujące się najwyżej (zależnie od pola sort) nowe zlecenie
- Zmienia zlecenie

Warunkiem wykonania operacji jest oczywiście właściwa konfiguracja nadzorcy - wybór zlecenia z kolejki zleceń.
Dodatkowo rozkaz zabezpieczony jest czasowo pomiędzy wykonaniami musi upłynąć co najmniej 25 sekund

Przykład skryptu

```
if ORDER_ID1 then   // jest jakieś zlecenie
   if F_INSV20 then    // po zmianie wejścia przypisanego do nadzorcy 20
       NextOrder(1)      // zmień zlecenie na następne
```

Proszę zwrócić uwagę że użyto flagi wejścia nadzorcy 20 co pozwala na odpowiednie ukształtowanie sygnału wejściowego filtrami.

Rozkaz ten stworzono dla obsługi konkretnej sytuacji - na nocnej zmianie roboczej pracownik po zakończeniu zlecenia może za pomocą wejścia
(np stacyjki z kluczykiem podłączonym do odpowiedniego wejścia) może wybrać następne zlecenie z listy przygotowanych zleceń.

5.11 czas

Pisząc skrypt możemy skorzystać z dostępnych w pascalu funkcji związanych z czasem takich jak DecodeDate, DecodeTime, DATE, NOW itp.
Nie będziemy ich tu opisywać gdyż są one typowymi funkcjami języka pascal - podamy za to przykład ich użycia w skrypcie.

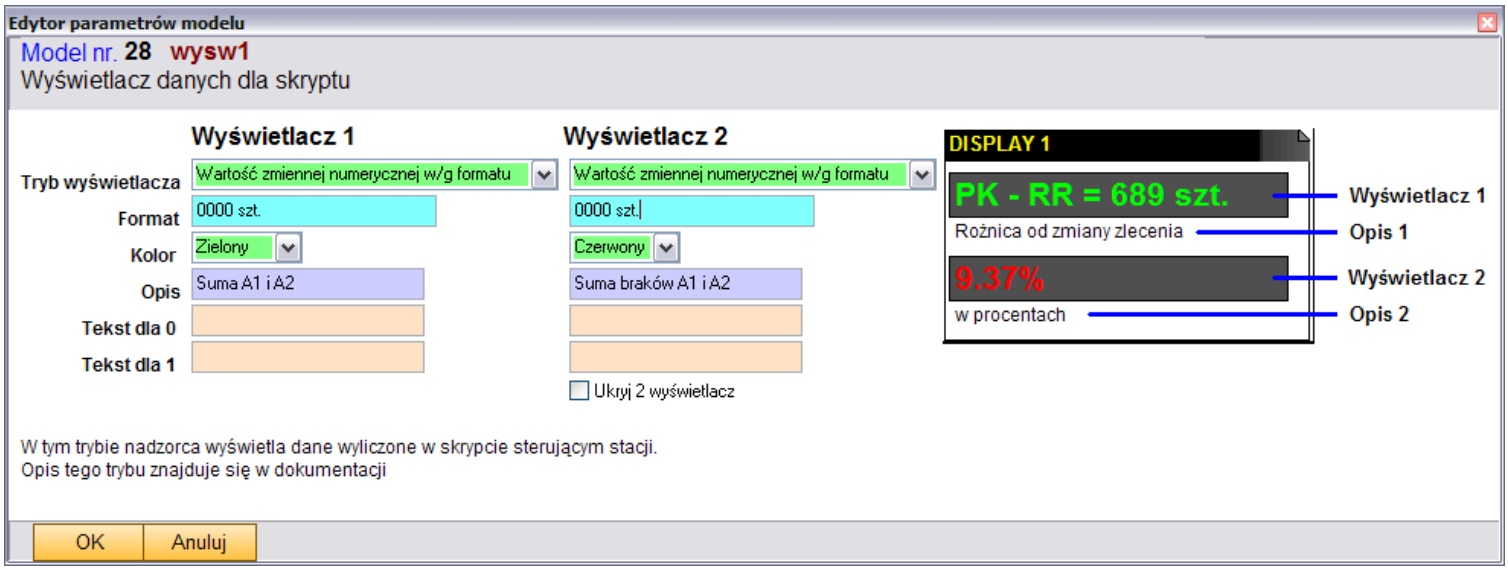
Nadzorca SV10 skonfigurowany jest jako rejestrator zdarzeń który ma rejestrować fakt otwarcia drzwi do magazynu.
Jako wejście główne w nadzorcy SV10 ustawiono wejście logiczne nr 70
Do wejścia numer 10 koncentratora podłączono czujnik który podaje na wejście stan wtedy gdy drzwi są zamknięte.

```
procedure TASK;
var g,m,s,ss :word;
b :boolean;
begin
 DecodeTime( NOW, g,m,s,ss);  // rozbij aktualny czas na godzinę, minutę, sekundę i setną sekundy
 b:= ( g > 5 ) and ( g < 16);  // zmienna b przyjmie wartość true jeśli aktualny czas jest w zakr
 KIn70 := ( not b ) and ( not KIn10 ); // wejście logiczne 70 ustawi się jeśli drzwi są otwarte a
end;
```

Jaki efekt uzyskamy? Stan wejścia logicznego, a co za tym idzie rejestracja otwarcia drzwi magazynu odbywać się będzie tylko między 15:00 a 5:59 rano.

6 Wyświetlacz

Jednym z trybów modelu jest tryb 12 – tryb wyświetlacza dla skryptu. W trybie tym wyświetlany jest pojedynczy lub podwójny wyświetlacz.



Wyświetlaną wartość ustala procedura DISPLAY a ustawienie modelu decyduje o formie w jakim ta dana będzie wyświetlana.
Możemy ustalić czy widoczny będzie 1 czy dwa wyświetlacze, zdefiniować kolor czerwony lub zielony zdefiniować nazwę oraz format dla liczby.
Określamy też sposób wyświetlania

- wyświetlany jest parametr X wg formatu
- Wyświetlany jest tekst 0 lub tekst 1 w zależności czy x=0 czy x=1
- Wyświetlany jest x jako liczba sekund w przyjętym w golemie formacie czasu xD xx:xx:xx

Format to tekst w którym program wymieni liczbę zdefiniowaną w postaci zer lub zer i znaków # na odpowiednią wartość. np. dla wartości x = 34.8705 zobaczymy:

Format	Zobaczymy
0	35
0.00	34.87
0.##	34.87 ale jeśli x będzie 74.00 to zobaczymy tylko 74
0.00%	34.87%
00000.00	00034.87
A+B=0.00	A+B=34.87

Przykład użycia wyświetlacza do wyświetlania sumy produktów i braków dwu maszyn (konfiguracja widoczna na ilustracji wyżej)

```
procedure Task;
Begin
 vint1 := order_cp1 + order_cp2;
 vint2 := order_defect1 + order_defect2;
  display(28,1,vint1);
  display(28,2,vint2);
end;
```

gdzie górny wyświetlacz wyświetla sumę produktu a dolny sumę braków dla bieżącego zlecenia nadzorców SV1 i SV2. Wyświetlacz obsługiwany jest przez nadzorcę SV28.

Przykład ilustruje też użycie zmiennych Vint. Ten sam skrypt można zapisać tak:

```
procedure Task;
Begin
  display(28,1,order_cp1 + order_cp2);
  display(28,2,order_defect1 + order_defect2);
end;
```