Golem OEE MES – Skrypt sterujący stacją

Poniższy dokument jest rozszerzeniem dokumentacji systemu **Golem OEE MES** dostępnej na stronie neuron.com.pl

W skrócie

- *skrypt sterujący pozwala na zwiększenie funkcjonalności stacji*
- *skrypt jest prostym programem który piszemy w języku podobnym do Pascala*
- *skrypt edytujemy za pomocą edytora wbudowanego w program konstruktor*
- *Nieumiejętne posługiwanie się skryptem może zdestabilizować pracę systemu*

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.

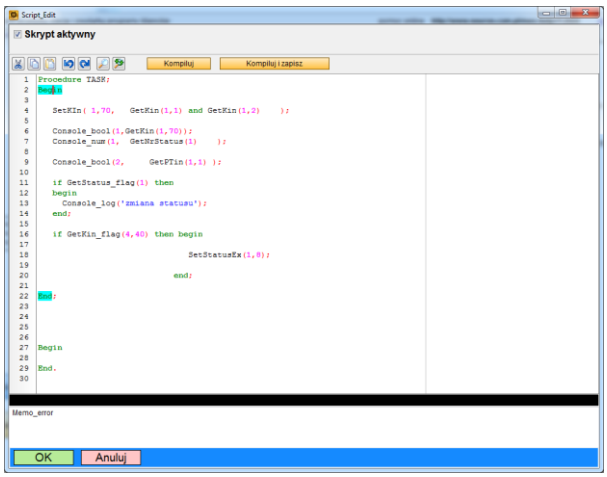
Ważne ! Konfigurując system, ustalając poszczególne parametry w definicji modelu czy nadzorcy mamy prawo oczekiwać aby możliwe ustawienia były LOGICZNE. Pisząc skrypt sterujący to my jesteśmy odpowiedzialni za logikę programu i musimy mieć pełną świadomość tego co robimy.
Nieumiejętne posługiwanie się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

Skrypt ładowany jest i kompilowany w momencie uruchomienia stacji. Jeżeli skrypt będzie zawierał błędy to stacja się uruchomi ale skrypt 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.

Edycja i aktywacja skryptu

Domyślnie skrypt jest wyłączony i trzeba go aktywować w edytorze skryptu.



Skrypt sterujący stacji edytujemy w programie konstruktor.

Podczas pisania skryptu możemy się wspomóc lista procedur i funkcji otwierana za pomocą kombinacji CTRL + SPACE

Podczas kompilacji skryptu (kompilacja dokonywana jest również podczas zamknięcia edytora) sprawdzana jest składnia a błędy są oznaczana i opisane w dolnym polu.

Budowa skryptu

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 20 ms.

```
procedure Task; // procedura uruchamiana co 20 ms
Begin
  // wszystkie rozkazy które znajdują się pomiędzy tymi rozkazami początku i końca bloku (begin / end)
  // wykonywane będą co 20 ms
end;

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

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

```
procedure TASK;
begin
 repeat
 until not GetKin(1,1);
end;
```

Ale konstrukcja taka jest niedopuszczalna gdyż może zatrzymać pracę skryptu - jeżeli nie będzie załączone wejście 1 koncentratora to procedura TASK zamknie się w "martwej" pętli.

Poza procedurą Task która być musi i której nazwy zmieniać nie wolno oraz kodem pomiędzy Begin i end z „kropką” możemy stosować dowolne, własne procedury i funkcje zgodnie ze specyfikacją języka Pascal.

Zmienne i flagi

Zmienne możemy podzielić ogólnie na zmienne systemu i zmienne skryptu. Zmienne systemu to stany wejść, stany liczników, numery aktualnych statusów poszczególnych nadzorców itp.
Zmienne skryptu to zmienne pomocnicze zdefiniowane w skrypcie. Te z kolei podzielić możemy na zmienne globalne i zmienne lokalne.

```
var licznik1:integer; // zmienna globalna

Procedure TASK;     // procedura task wywoływana cyklicznie
var in1,in2,inl:boolean;
Begin
   in1:= GetKin(1,1);
   in2:= GetKin(1,2);
   inl:= in1 and in2;
   SetKIn( 1,70, inl   );

   if GetStatus_flag(1) then
   begin
     Console_log('zmiana statusu nadzorcy 1');
   end;
End;

Begin
 // instrukcje w tym bloku wykonywane są jeden raz po inicjacji stacji
 licznik1:=100;
End.
```

W powyższym przykładzie widzimy zmienną globalną licznik1 oraz trzy zmienne lokalne procedury task (widoczne są tylko w jej wnętrzu)

Dostęp do zmiennych i flag systemu

Skrypt nie miał by sensu gdyby jego kod nie mógłby „sięgnąć” do zmiennych systemu. Odbywa się to jednak za pomocą funkcji i procedur. Ten fragment kodu:

```
   in1:= GetKin(1,1);
   in2:= GetKin(1,2);
   inl:= in1 and in2;
   SetKIn( 1,70, inl   );
```

odczytuje stan wejść koncentratora za pomocą funkcji GetKin do zmiennych lokalnych in1 i in2 i zapisuje zmienną inl do wejścia logicznego koncentratora (wejścia 65 do 111) za pomocą procedury SetKin
Dostęp do zmiennych za pomocą funkcji daje możliwość ich używania w procedurach i pętlach co znacznie upraszcza kod dla wielu maszyn co pokażemy w przykładach.

Wejścia logiczne koncentratora

Kiedy wybieramy wejście koncentratora to możemy ustawić je w zakresie od 1 do 128. A przecież koncentrator ma tylko 64 wejścia. Pozostałe wejścia to wejścia logiczne.
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 ustawiając ich stan za pomocą procedury SetKin

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.

procedury symulujące czynności operatora

Użycie niektórych procedur ma skutek identyczny jak operacja wykonana przez operatora. Przykładowo procedura SetNoStatus zmienia numer statusu wybranego nadzorcy, ale bez żadnych dodatkowych czynności, bez tworzenia wpisu w rejestrze powiadomień, wysyłania maili itp.
Z kolei procedura SetStatus zmienia status podstawowy wskazanego nadzorcy dokładnie tak samo, z takimi samymi konsekwencjami, jak zmiana statusu przez operatora. dlatego mówimy że procedury te symulują czynności operatora.

Przegląd procedur i funkcji z przykładami użycia

Wejścia koncentratora

Funkcje i procedury do odczytu i zapisu stanu różnych wejść systemu.

function GetKin(NrK,NrIn:integer):boolean	NrK – numer koncentratora NrIn – numer wejścia	Zwraca stan wybranego wejścia wybranego koncentratora
function GetSVin(SV:integer; filtr:boolean):boolean	SV – numer nadzorcy filtr –true to wejście filtrowane	Zwraca stan wejścia (filtrowanego lub nie filtrowanego) przypisanego do wskazanego nadzorcy
function GetASin(AS:integer; filtr:boolean):boolean	SV – numer nadzorcy filtr –true to wejście filtrowane	Zwraca stan wejścia (filtrowanego lub nie filtrowanego) przypisanego do wskazanego asystenta
function GetPTin(SV,NR:integer):boolean	SV – numer nadzorcy NR – numer pt1 lub pt2	Zwraca stan wejścia przypisanego do sterowania przerwą technologiczną wskazanego nadzorcy
procedure SetKIn(Nrk,NrIn:integer; state:boolean)	NrK – numer koncentratora NrIn – numer wejścia state - stan	Przypisuje stan wejścia logicznego wskazanego nadzorcy. Uwaga: wejścia z zakresu 65 do 111

Flagi

Flagi to specjalne zmienne systemowe czytane przez funkcje. Flaga ustawiana jest podczas jakiegoś zdarzenia w systemie, np. podczas zmiany statusu a kasowana wykonaniu całego skryptu.
Dzięki temu stan flagi odczytany zostanie tylko raz po zdarzeniu

function GetSVin_flag(sv:integer):boolean	SV – numer nadzorcy	Zwraca stan flagi wejścia wskazanego nadzorcy. Flaga zmienia się podczas zbocza narastającego wejścia filtrowanego
function GetStatus_flag(sv:integer):boolean	SV – numer nadzorcy	Zwraca stan flagi sygnalizującej zmianę statusu
function GetOrder_Flag(sv:integer):boolean	SV – numer nadzorcy	Zwraca stan flagi zmiany zlecenia wybranego nadzorcy
function GetEndOrder_Flag(sv:integer):boolean	SV – numer nadzorcy	Zwraca stan flagi końca zlecenia wybranego nadzorcy
function GetOperator_Flag(sv:integer):boolean	SV – numer nadzorcy	Zwraca stan flagi zmiany operatora wybranego nadzorcy
function Get1s_flag:boolean		Zwraca stan flagi ustawianej co sekundę

W przykładzie poniżej funkcja GetStatus_Flag czyta stan flagi statusu 1 nadzorcy a kod pomiędzy Begin / end zostanie wykonany raz po zmianie statusu.

```
if GetStatus_flag(1) then
begin
  Console_log('zmiana statusu nadzorcy 1');
end;
```

Status

Funkcje i procedury związane z odczytem i sterowaniem statusu

function GetNrStatus(SV:integer):integer	SV – numer nadzorcy	Zwraca numer aktualnego statusu wskazanego nadzorcy 0 – postój planowany 1 – konserwacja 2 – przezbrajanie 3 - ustawianie 4 – postój nieplanowany 5 – awaria 6 - PRACA
function GetNrStatusEx(SV:integer):integer	SV – numer nadzorcy	Zwraca numer aktualnego statusu rozszerzonego wskazanego nadzorcy
procedure SetStatus(sv,nr:integer)	SV – numer nadzorcy NR – numer statusu	Symuluje zmianę statusu podstawowego na NR przez operatora dla nadzorcy SV
procedure SetStatusEx(sv,nr:integer)	SV – numer nadzorcy NR – numer statusu	Symuluje zmianę statusu rozszerzonego na NR przez operatora dla nadzorcy SV

Liczniki, stan, braki

function GetCount(sv,c,r:integer):integer	SV – numer nadzorcy C – rodzaj licznika R – zakres licznika	Odczytuje stan wybranego licznika wybranego nadzorcy dla wybranego zakresu czasu rodzaj licznika C: 0 – licznik cykli 1 – licznik produktu 2 – licznik braków 3 – licznik odpadu 4 – licznik energii zakres R 0 = bieżący miesiąc 1 – bieżąca zmiana robocza 2 – bieżące zlecenie
function GetParams(sv,p:integer):single	SV – numer nadzorcy P – rodzaj parametru	Odczytuje stan wybranego parametru wybranego nadzorcy rodzaj parametru P 0 – optymalny czas cyklu (lub Tt) 1 – krotność 2 – zlecenie – ilość zamówiona 3 – zlecenie – ID
function GetState(sv,nr:integer):boolean	SV – numer nadzorcy NR – numer parametru	Odczytuje stan wybranego, logicznego parametru nadzorcy SV parametr nr: 0 – biegnie czas pracy 1 – biegnie czas pracy lub mikro postoju 2 – biegnie czas nieoznaczony
procedurę SVReset(sv:integer);	SV – numer nadzorcy	kasuje wszystkie liczniki kasowane nadzorcy w trybie pseudo – zlecenia lub bez zlecenia
procedure AddDefect(sv,x:integer)	sv – numer nadzorcy x – ilośc braków	Zwiększa liczniki braków nadzorcy o wartość x
procedure SetDefect(sv,x:integer)	sv – numer nadzorcy x – ilośc braków	procedura symuluje dodanie braków przez operatora

function GetAndonsv:integer):integer	sv – numer nadzorcy	Zwraca stan wezwania andon: 0 – brak wezwania 1 – wezwanie pomocy 2 - wezwanie pomocy technicznej 3 – wezwanie pomocy logistycznej 4 i 5 – wezwanie pomocu U1, U2
procedure SetAndon(sv,nr:integer);	sv – numer nadzorcy nr – numer wezwania	Zwraca stan flagi określającej koniec odliczania czasu przez wybrany timer
procedure AddComment(sv,mode:integer; txt:string)	sv – numer nadzorcy mode – rodzaj komentarza txt – tekst komentarza	dodaje komentarz użytkownika do rejestru powiadomień

Uwaga – procedury takie jak SetAndon czy SetStatus oprócz blokady czasowej ustalają wymaganą wartość i nie wykonują się jeśli wartość taj jest aktywna. Można więc bezpiecznie użyć konstrukcji w stylu :
jeśli wejście to SetStatus
Procedura AddComment w takim przypadku będzie wykonywana co 5 sekund dlatego należy ją używać tylko z flagami które zagwarantują 1 wywołanie.

Timery

W skrypcie możemy użyć pomocnicze timery do odmierzania czasu.

function GetTimer(nr:byte):integer	nr – numer timera numer timera od 0 do 512	Zwraca licznika czasu (w sekundach) wybranego timera
function GetTimer_Flag(nr:byte):boolean	nr – numer timera numer timera od 0 do 512	Zwraca stan flagi określającej koniec odliczania czasu przez wybrany timer
procedurę SetTimer(nr,czas:integer)	nr – numer timera numer timera od 0 do 512 czas – czas w sekundach	Ustawia czas wybranego timera. Po doliczeniu czasu do zera aktywowana zostanie flaga.
function Get1s_flag:boolean		Zwraca stan flagi ustawianej co sekundę

Timer odlicza wpisany czas co sekundę do zera. Gdy osiągnie wartość zero to na jeden cykl ustawi flagę.
Poniższy fragment skryptu sprawdza czy maszyna pracuje (biegnie czas efektywnej pracy testowany funkcją GetState) a jeśli tak to cały czas wpisuje wartość 120 sekund do licznika timera.
Gdy maszyna przestaje pracować timer po 120 sekundach doliczy do zera i ustawi flagę. W 3 wierszu funkcja GetTimer_flag testuje stan flagi i gdy timer doliczy do zera wywoła polecenie zmiany statusu

```
if GetState(1,0) then SetTimer(1,120);  // jeśli maszyna 1 pracuje cały czas wpisuj 120 sekund
if GetTimer_flag(1) then // jesli skończył się czas (upłynelo 120 sekund)
     SetStatus(1,4);     // zmień statusu na postój nieplanowany
```

Konsola kontroli

W programie stacji zbierania danych dostępne jest okno pozwalające na podgląd skryptu i jego poprawności. Dostępna jest w nim konsola przydatna podczas uruchamiania i testowania skryptu.
Do dyspozycji mamy pole tekstowe gdzie dopisywane są dowolne teksty za pomocą rozkazu console_log oraz po 8 pól numerycznych i logicznych które ustawiane są rozkazami console_num i console_bool.

Rozkazy są aktywne jeśli kosola jest aktywna – została aktywowana ręcznie.

procedure Console_log(txt:string)	txt – tekst komunikatu	dopisuje komunikat do konsoli Uwaga; procedura TASK wykonuje się co ok 20ms więc jeśli użyjemy procedurę conslole_log w jej ciele bez ograniczenia flagami to komunikat będzie dopisywany w takim tempie
procedure Console_num(nr:byte; value:single)	nr – numer punktu kontrolnego value – wartość liczbowa	Wyświetla we wskazanym numerycznym punkcie kontrolnym wartość numeryczną
procedure Console_bool(nr:byte; value:boolean)	nr – numer punktu kontrolnego value – wartość logiczna	Wyświetla we wskazanym logicznym punkcie kontrolnym wartość logiczną true lub false

Przykłady skryptów i technik

dodatkowe sterowanie statusem rozszerzonym za pomocą wejść

Polecenie SetStatusEx pozwala na zmianę statusu rozszerzonego. Procedura ta symuluje czynność operatora – jej wywołanie daje identyczny efekt jak zmiana statusu przez operatora za pomocą panelu operatora lub za pomocą aplikacji mobilnej.
Wyobraźmy sobie maszynę dla której zdefiniowano kilkanaście statusów, w tym status postój nieplanowany: zablokowana maszyna. Status ten ma numer 11 (lista statusów w konstruktorze) i chcemy przy maszynie umieścić dwa przyciski – jeden, podłączony do 1 wejścia 8 koncentratora ma zmienić status na postój nieplanowany: zablokowana maszyna, ale tylko jeśli aktualnym statusem jest praca.
Drugi przycisk podłączony do 2 wejścia 8 koncentratora zmieni status na pracę.
Skrypt mógłby wyglądać tak:

```
if (GetNrStatus(1)=6) // jeśli aktualny status to praca
and GetKin(8,1) then  // i wejście 1 8'koncentratora
  SetStatusEx(1,11);  // zmień status rozszerzony na nr 11 - a postój nieplanowany

if GetKin(8,2) then   // jeśli aktualny status to praca i wejście 2 8'koncentratora
  SetStatusEx(1,1);   // zmień status rozszerzony  na nr 1 - praca

Console_num(1,  GetNrStatus(1)    );  // w polu num1 wyświetl nr statusu podstawowego
Console_num(2,  GetNrStatusEx(1)  );  // w polu num2 wyświetl nr statusu rozszerzonego
```

W dwu ostatnich wierszach dodano podgląd numerów statusu i statusu rozszerzonego w trybie debugowania.

Pamiętajmy jednak o tym że operator musi widzieć zmianę statusu czyli mieć w zasięgu wzroku monitor albo telewizor aby wiedział czy naciśnięcie przycisków dało jakiś efekt.

Sterowanie statusu ustawionego w tryb sterowania wejściami

W skrypcie możemy operować statusem „udając” operatora. W określonych warunkach zmieniać stan statusu wołając odpowiednie procedury.
Ale jest inna technika. Można ustawić status sterowany za pomocą wejść i przypisać wejścia logiczne.
Przykładowo ustawić status na sterowanie wejściami AB a do wejść przypisać wejścia np. 70 i 71.
Następnie możemy w skrypcie manipulować stanem tych wejść a ich zmiana spowoduje zmianę statusu przez mechanizmy stacji.

synchronizacja maszyn w linii produkcyjnej

Jednym z zastosowań skryptów jest synchronizacja maszyn wchodzącej w skład jednej linii produkcyjnej polegająca. Poniższy przykład wykrywa zmianę statusu i ustawia status w dwu maszynach pomocniczych oraz wykrywa zmianę zlecenia maszyny głównej i kasuje liczniki maszyn pomocniczych.
Zadanie to realizuje procedura LiniaPP z trzema parametrami – numerem nadzorcy maszyny głównej i numerami dwu maszyn pomocniczych.
Pierwszy blok kodu bada stan flagi zmiany zlecenia i jeśli ją wykryje wykonuje dwie komendy SVReset które kasują liczniki kasowane lub liczniki psedozlecenia
Drugi blok wykrywa zmianę statusu, odczytuje nowy status i wywołuje procedury SetStatus ustawiający identyczny status na maszynach a i b.

```
// procedura zmiany statusu i kasowania linii PP
Procedure LiniaPP(nr,nra,nrb:integer);
var status:integer;
begin
  // po zmianie zlecenia na maszynie głównej kasuj zlecenia maszyn pomocniczych
  if GetOrder_flag(nr ) then
   begin
    SVReset(nra);
    SVReset(nrb);
    Console_log('reset maszyn '+inttostr(nra)+', '+inttostr(nrb));
   end;
  // po zmianie statusu maszyny głównej zmień status maszyn pomocniczych
  if GetStatus_flag(nr) then
   begin
     status:= GetNrStatus(nr);
     SetStatus( nra, status);
     SetStatus( nrb, status);
```

```
     Console_log('nowy status maszyn '+inttostr(nra)+', '+inttostr(nrb));
   end;
end;
```

Zwróćmy uwagę na to że kod nie znajduje się w obrębie procedury TASK a w oddzielnej procedurze LiniaPP. Aby został on wykonany musimy wywołać LiniaPP w procedurze TASK. Zastosowanie dodatkowej procedury porządkuje kod ale przede wszystkim pozwala na obsługę większej ilości podobnych zestawów:

```
Procedure TASK;
Begin

   LiniaPP(51,52,53);  // Linia nr 1
   LiniaPP(54,55,56);  // Linia nr 2
   LiniaPP(60,61,62);  // linia nr 3

End;
```