

State MACHine

Qui sono raccolte le state machine in termini di file origine (*.rul) e file di configurazione delle state machine ingressi dei vari **IOB**

- [State MACHine](#)
 - [Standard Tecnici](#)
 - [Definizione ed acronimi](#)
 - [Descrizione Impiego](#)
 - [Sintassi file RULES](#)
 - [Commento iniziale](#)
 - [Definitions](#)
 - [BIT](#)
 - [STATES](#)
 - [EVENTS](#)
 - [RULES](#)

Standard Tecnici

Il progetto impiega i seguenti standard tecnici:

- http call
- Python 2.7 (R-IOB)
- asp.net / C# (W-IOB)

Definizione ed acronimi

Di seguito una legenda degli acronimi usati in seguito (per Componenti, Sistemi, Attori, ...):

Acronimo	Definizione
MapoState	Applicazione epr compilazione rul --> stati
*.rul	RULES: File regole generazione macchina a stati
*.csv	CSV: File compilato macchina a stati

Descrizione Impiego

Il sw principale (che deve tipicamente trovarsi installato in c:\lavori) permette di compilare un file rul per generare la macchina a stati come file csv che può venire caricato nel DB nella tabella delle **TransizioniIngressi** ovvero quella che descrive la macchina a stati degli ingressi e per il riconoscimento degli **EVENTI MACCHINA** del sistema Mapo

Sintassi file RULES

Il file dei rules prevede la seguente struttura riportata e commentata a blocchi di seguito. I commenti iniziano col carattere #.

Commento iniziale

L'area iniziale prevede i commenti descrittivi tra cui

- nome macchina
- ragionamento globale macchina a stati dei segnali **S.IOB**
- modalità di implementazione sistema contapezzo
- ...

```
#
# Donati Macchine SM05 , SM06 , SM09 , SM10 ( PLC Stefano )
# Macchine Ghidini
#
```

```
# v.1 31-X-2013 prima versione compilata : sostituisce versione scritta a mano
# v.2 9-XII-2013 rimessa regola : se segnale contapezzo, vale per tutti gli stati
#
# volendo fare i figli , bit 4 sulla SM10 ⚡ fine nastro OR fine Spazzola
#
# nella macchina manuale era definito uno stato 8 "Scarico pieno" ma su queste macchine non c'⚡ il
segnale
#
# segnale contapezzo : non rimanda evento se gi⚡ nello stato "Cycle end" <<<< no dalla versione 2
# si faceva fottere da 3 o + campioni a 7 in ingresso
# primo input a 7 --> stato end cycle
# secondo input a 7 --> stato run !!! ( mancava la regola )
# terzo input a 7 --> di nuovo stato end cycle ( e quindi un pezzo in pi⚡ )
```

Definitions

La seconda area è quella delle definizioni degli "ingredienti" del file di generazione della macchina a stati, con alcune aree di seguito definite

- In primis sono definiti
 - nome del ruolo
 - **IDX** ovvero chiave univoca della **Famiglia Ingresso**
 - numero Stati previsti
 - numero bit di ingresso gestiti

\$DEFINITIONS

```
$NAME      : stefano
$IDX       : 11
$N_STATES  : 8
$N_BITS    : 8
```

BIT

Successivamente vanno definiti i singoli bit (zero based) in quantità pari a quanto definito prima, attenzione al nome perché poi sarà chiave per le fasi successive

#definizione bit : obbligatorio iniziare da 0

```
$BIT      : 0      : power_on
$BIT      : 1      : run
$BIT      : 2      : end_cycle
$BIT      : 3      : alarm
$BIT      : 4      : fine nastro
$BIT      : 5      : manual
$BIT      : 6      : spare1
$BIT      : 7      : spare2
```

STATES

In modo analogo ai bit vanno definiti tutti gli stati previsti e configurati:

- nome status deve iniziare (per convenzione) per ST_
- vanno previsti MINIMO gli stati di init, power on, run, alarm, manual

#definizione stati : obbligatorio iniziare da 0

```
$STATE     : 0      : ST_Init
```

```

$STATE      : 1      : ST_Power off
$STATE      : 2      : ST_Machine ready
$STATE      : 3      : ST_Run
$STATE      : 4      : ST_Cycle end
$STATE      : 5      : ST_Alarm
$STATE      : 6      : ST_Broken belt
$STATE      : 7      : ST_Manual
#####STATE    : 8      : Output full

```

EVENTS

A questo punto sono indicati gli eventi che si potranno inviare a valle. Questi DEVONO corrispondere agli eventi configurati nel DB di MAPO

- gli eventi sono zero-based come indice interno
- i primi eventi riportati (EV_00 .. EV_12) NON saranno poi gestiti dalla macchina a stati, sono infatti eventi MANUALI gestiti dal DB centrale tramite interfaccia **TAB**
- vanno inseriti (e poi gestiti) quindi tutti gli eventi HW previsti e nel caso siano necessari nuovi eventi vanno configurati in modo analogo anche sul DB centraleS

#definizione eventi : obbligatorio iniziare da 0

```

$EVENT      : 00      : EV_00
$EVENT      : 01      : EV_01
$EVENT      : 02      : EV_02
$EVENT      : 03      : EV_03
$EVENT      : 04      : EV_04
$EVENT      : 05      : EV_05
$EVENT      : 06      : EV_06
$EVENT      : 07      : EV_07
$EVENT      : 08      : EV_08
$EVENT      : 09      : EV_09
$EVENT      : 10      : EV_10
$EVENT      : 11      : EV_11
$EVENT      : 12      : EV_12
$EVENT      : 13      : HW - init
$EVENT      : 14      : HW - power off
$EVENT      : 15      : HW - power on
$EVENT      : 16      : HW - machining
$EVENT      : 17      : HW - end machining
$EVENT      : 18      : HW - error
$EVENT      : 19      : Barcode - cambio operatore
$EVENT      : 20      : Contapezzi
$EVENT      : 21      : HW - start pallet
$EVENT      : 22      : HW - end pallet
$EVENT      : 23      : HW rottura nastro abrasivo
$EVENT      : 24      : HW manuale
$EVENT      : 25      : HW nastro scarico pieno
$EVENT      : 26      : Barcode - Manca Riforn. MPD
$EVENT      : 27      : Timer - timeout tempo ciclo
$EVENT      : 28      : Timer - timeout TURNO by tempo ciclo
$EVENT      : 29      : HW - magazzino grezzi vuoto
$EVENT      : 30      : HW - emergenza

```

RULES

Infine arrivano i veri RULES con i quali verranno costruite le macchine a stati

- la regola è nel formato stato in cui va applicata (con la keyword **ALL_STATES** che indica tutti gli stati)
- l'input definisce quale bit genera un evento di stato (non come numero ma come nome descrittivo / 3° colonna della conf precedente)
- next state rappresenta il nuovo (micro)stato per la state machine ingressi (utile ad esempio epr definire loop/microstati transitori)

- event rappresenta l'evento che deve venire generato alla ricezione del valore in input quando ci si trova nello stato corrente
- è importante l'ordine: sarà eseguita la regola valida trovata per prima partendo dall'alto

```

$RULES

# state      : input      : next state      : event -----
-----

ALL_STATES   : NOT power_on      : ST_Power off    : HW - power off
ALL_STATES   : manual              : ST_Manual        : HW manuale
ALL_STATES   : fine nastro        : ST_Broken belt   : HW rottura nastro abrasivo
ALL_STATES   : alarm           : ST_Alarm         : HW - error

# rimetta a posto la candela !

ALL_STATES   : end_cycle      : ST_Cycle end     : HW - end pallet

ALL_STATES   : run           : ST_Run           : HW - machining
ALL_STATES   : power_on      : ST_Machine ready : HW - power on

$DO

```

NB: il file deve chiudersi con il tag **\$DO**