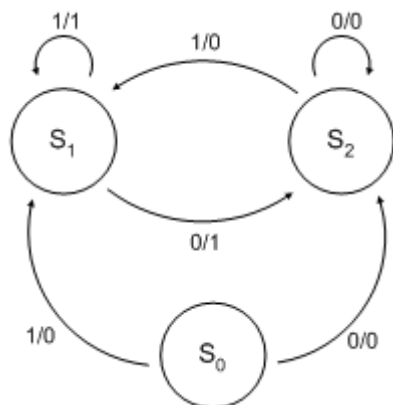


# Macchine a Stati: Stringhe o Enumeratori?

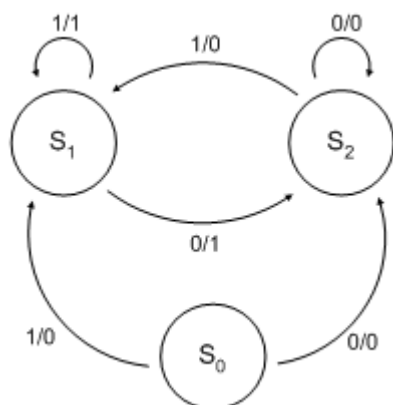
Category: Insight

scritto da Lorenzo Borghi | Settembre 19, 2025



In questo articolo analizziamo due approcci alla costruzione di **State Machine** in LabVIEW, confrontando pro e contro e scoprendo in quali scenari ciascuno risulta più adatto.

## LabVIEW State Machine



Una **State Machine** descrive un'applicazione come sequenza di **stati** con regole di transizione.

In LabVIEW si realizza tipicamente con:

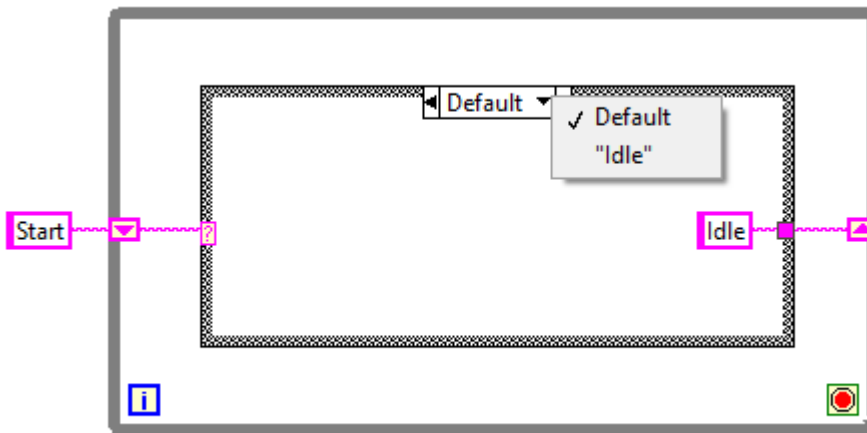
- un **While Loop** che gestisce il ciclo,
- una **Case Structure** che rappresenta gli stati,
- uno **shift register** che memorizza il prossimo stato.
- Il messaggio solitamente un **enum** o una **stringa** (altri tipi di dato sarebbero troppo poco descrittivi per leggibilità e integrazione)

Il vero nodo sta nel **tipo di dato usato come messaggio**, che può essere **stringa**, **ring** o **enum**.

Il **ring** equivale a gestire numeri: limitato e poco utile per rappresentare un

tipo "messaggio" per la State Machine. La community apre la discussione quindi, tra **stringhe** ed **enumeratori**.

## Messaggi String : Fessibilità



L'uso delle **stringhe** come message type, in una State Machine di [LabVIEW](#) è il metodo più flessibile e meno vincolante.

Ogni stato è rappresentato dal valore della stringa, passato nello shift register e richiamato dalla Case Structure.

Questo approccio consente di aggiungere **stati rapidamente**, non obbliga alla dichiarazione di una typedef, ed è adatto a prototipi o logiche ridotte, anche quando i messaggi sono raccolti per esempio da strumentazione.

Il limite principale è l'assenza di **controllo in edit-mode**: un errore di battitura genera uno stato non previsto, da curare con uno statpo default per gestire eccezioni di messaggi non interpretati (come per esempio suggerito nei 5 stati fondamentali Idle, Init, **Default**, Error, Exit).

Le stringhe sono inoltre **case sensitive** (superabile abilitando [insensitive match](#) sulla stuttura del case).

Il debug, se non si applicano strategie preventive, diventa complesso, in particolare quando si devono gestire molti messaggi (pensiamo alle architetture piu complesse tra più [produttori e consumatori](#)).

Per riassumere:

### PRO

- Creazione immediata di stati.
- Massima flessibilità in sviluppo iniziale
- Messaggi da Hardware (i.e. ASCII)
- Manipolazione estesa, Serializzazione, Protocolli Custom

### CONTRO

- Nessun controllo a compile-time
- Rischio di duplicati ed errori di battitura
- Debug complesso senza strategie preventive

L'uso delle stringhe è molto diffuso tra i programmatori della community e le motivazioni sono verso la flessibilità e l'esperienza che permette strategie di prevenzione errori.

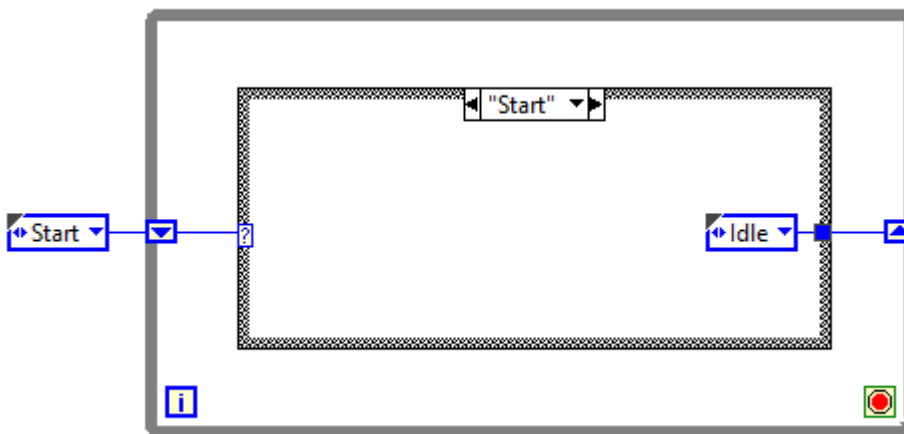
È sempre necessario un **Default case** (non una stringa "Default"), che intercetta valori non mappati.

Oltre a garantire la compilazione, è utile in architetture con più State Machine che condividono un dispatcher: gestisce stati non utilizzati senza replicare logiche e mantiene l'architettura leggera.

Un esempio di utilizzo dei messaggi arriva dalla [JKI State Machine](#) un esempio di implementazione molto versatile e dove i vantaggi dei msg come stringhe è completamente utilizzato.

Osserviamo come può essere possibile creare stati "segnaposto" che possono essere utilizzati anche per organizzare meglio tutti i messaggi nel case selector e formare anche macro command, per esempio concatenando messaggi e scodandoli estraendo una linea concatenata.

## Enumeratori come messaggi: Meno Errori e Rigore



L'uso degli **enumeratori (enum)** come case selector in una SM di LabVIEW è la soluzione che possiamo definire più scolastica.

I messaggi sono dichiarati in una type-def che garantisce che ogni nuovo messaggio da gestire deve essere prima DICHIARATO (in uno stile più C-like) e si distribuisce direttamente ai subVI che usano la typedef di messaggio.

Ogni stato è rappresentato da un valore simbolico definito nell'enum e, se tipizzato come **typedef**, ogni modifica si propaga automaticamente a tutto il progetto.

Questo garantisce coerenza e rende più semplice l'aggiornamento delle Case Structure.

Il principale vantaggio rispetto alle stringhe è il **controllo a compile-time**: se uno stato non è gestito nella Case Structure, LabVIEW lo segnala subito,

evitando comportamenti imprevisti a runtime. Gli enum sono inoltre *strongly typed* e non possono essere modificati a runtime, caratteristica che aumenta stabilità e riduce il rischio di errori.

Possono essere poi convertiti in TypeDef, generando coerenza tra tutte le copie di se stessi sparse per il codice, garantendo una replicazione della stessa struttura SM ovunque sia posizionata.

La comodità sta nella capacità di aggiornare il typedef quando si aggiungono o modificano stati, riflettendo il cambiamento ovunque sia utilizzata.

#### PRO

#### CONTRO

Controllo a compile-time

Richiede gestione del typedef

Aggiornamento automatico della Case Structure tramite typedef

Meno immediati nella prototipazione rapida

Maggiore leggibilità e coerenza del codice

Non necessita del caso di Default

L'uso degli enum è consigliato nelle State Machine destinate a sistemi complessi e manutenibili: il rigore imposto dal typedef previene errori e rende l'architettura più scalabile sul lungo periodo.

## Conclusione

La scelta del case selector in una State Machine di LabVIEW determina stabilità e manutenibilità del progetto.

- Le **stringhe** sono pratiche per prototipi e logiche ridotte, ma espongono a errori di battitura, duplicati e mancanza di controlli a compile-time.
- Gli **enumeratori** impongono disciplina progettuale attraverso typedef e garantiscono coerenza grazie al controllo a compile-time. Sono *strongly typed*: gli stati sono definiti a priori e non mutano a runtime.

Per applicazioni professionali e scalabili, gli **enum** rappresentano la scelta consigliata. Le stringhe restano utili solo in contesti sperimentali o di test rapido, dove la velocità conta più della robustezza.

La scelta e la discussione rimane accesa, anche in Bytelabs ci sono diverse scuole di pensiero e questo [post](#) dal forum della community NI fa capire quanto non ci possa essere una scelta migliore.

Secondo noi non importa più se usare TypeDef enum o String, importa piuttosto stabilire uno standard di lavoro condiviso tra il team.