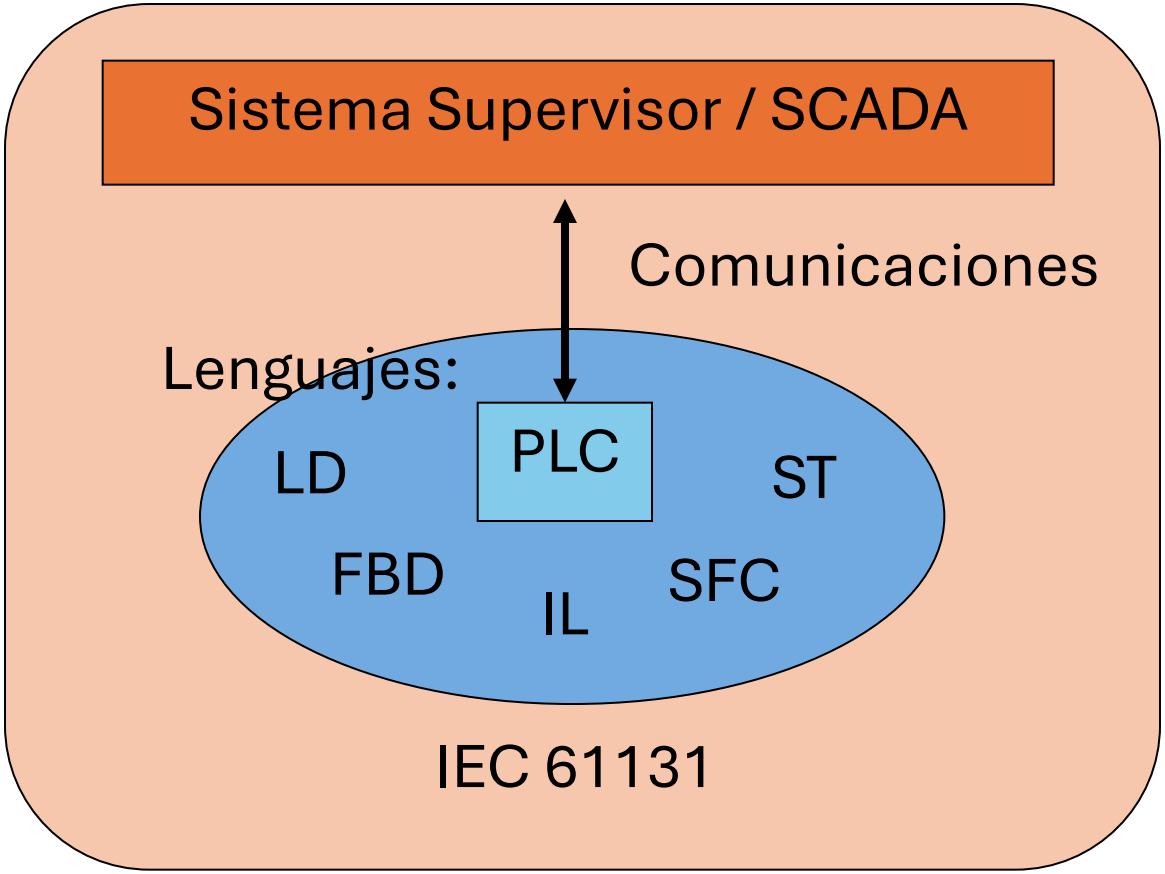
Estándar IEC 61131

Controladores Lógicos Programables

Programa del Curso



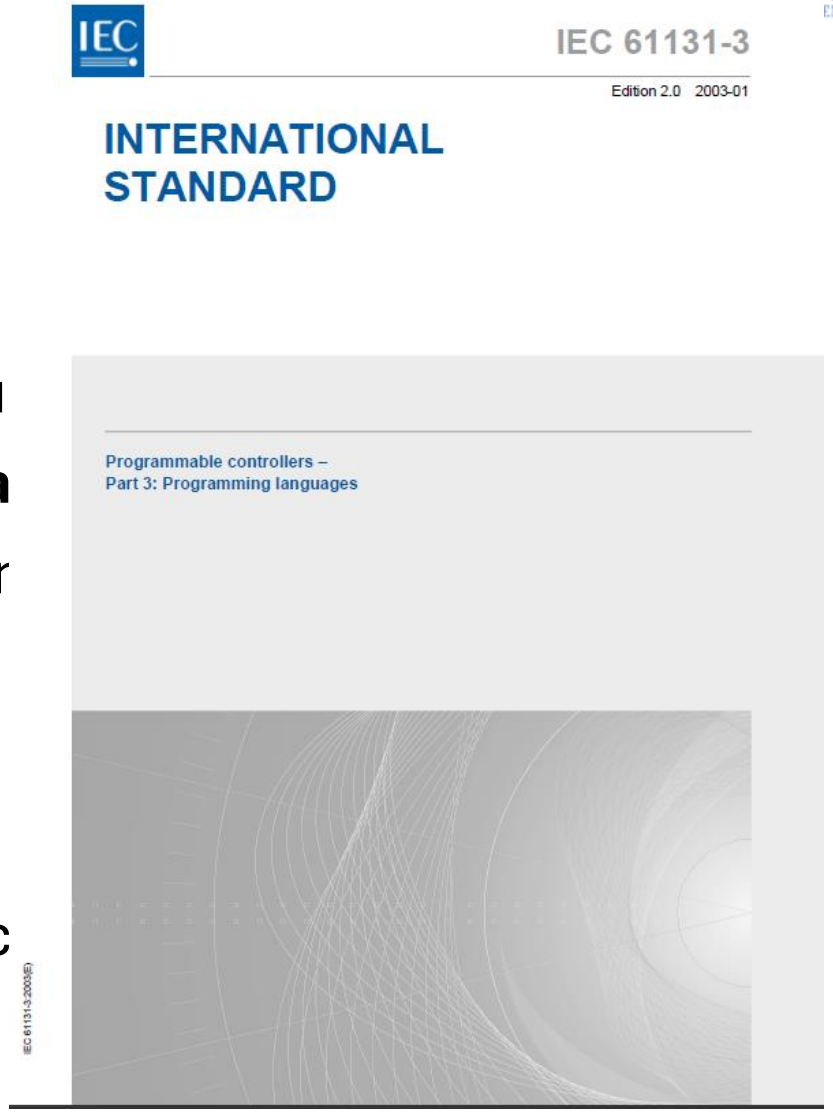
Proyectos / Aplicaciones

Estándar IEC 61131

- Normativa sobre PLCs y su aplicación a procesos industriales de medición y control
- Primera edición en 1993
- Uniformiza forma de programar PLCs
- Ingeniería de software aplicado a la industria
- Tendencia a sistemas abiertos, interoperabilidad, etc.
- Sistemas Abiertos:
  - Adpota estándares y técnicas industriales actuales
  - Facilita la integración con otros sistemas abiertos (interoperabilidad)
  - Foco en comunicaciones y programación

IEC 61131

- Parte 1 – Información General
- Parte 2 – Requerimientos y pru
- **Parte 3 – Lenguajes de Progra**
- Parte 4 – Guías de selección, ir
- Parte 5 – Comunicación
- Parte 6 – Seguridad Funcional
- Parte 7 – Lógica Borrosa
- Parte 8 – Guía de implementac



Deficiencias previo a IEC 61131

- Programación clásica en LADDER:
  - Estructura de software débil (soporte limitado para bloques de función, encapsulamiento, etc.)
  - Poco re-uso de software
  - No soporta datos estructurados
  - Soporte limitado para secuencias
  - Control de ejecución limitado (único tiempo de ciclo)
  - Cálculos aritméticos engorrosos

Calidad de Software

- Objetivo principal de IEC 61131-3: mejorar la calidad de software
- Atributos:

Capacidad
- Tiempo de respuesta
- Capacidad de procesamiento
- Capacidad de almacenamiento

Disponibilidad
- Confiabilidad (MTBF) – tiempo entre fallas
- Mantenibilidad (MTTR) – tiempo en reparar
- Integridad – robustez

Usabilidad
- Requerimientos previos para usarlo
- Esfuerzo de aprendizaje
- Productividad
- Amigabilidad

Adaptabilidad
- Posibilidad de mejoras en capacidad, disponibilidad, usabilidad
- Extensibilidad – nuevas funcionalidades
- Portabilidad
- Reuso

IEC 61131-3 => mejorar estos atributos de calidad de software

Características IEC 61131-3

1. Software estructurado (top-down/bottom-up)
2. Chequeo de tipos de datos
3. Control de ejecución (tareas)
4. Control secuencial
5. Datos estructurados
6. Elección de lenguajes
7. Software independiente del fabricante

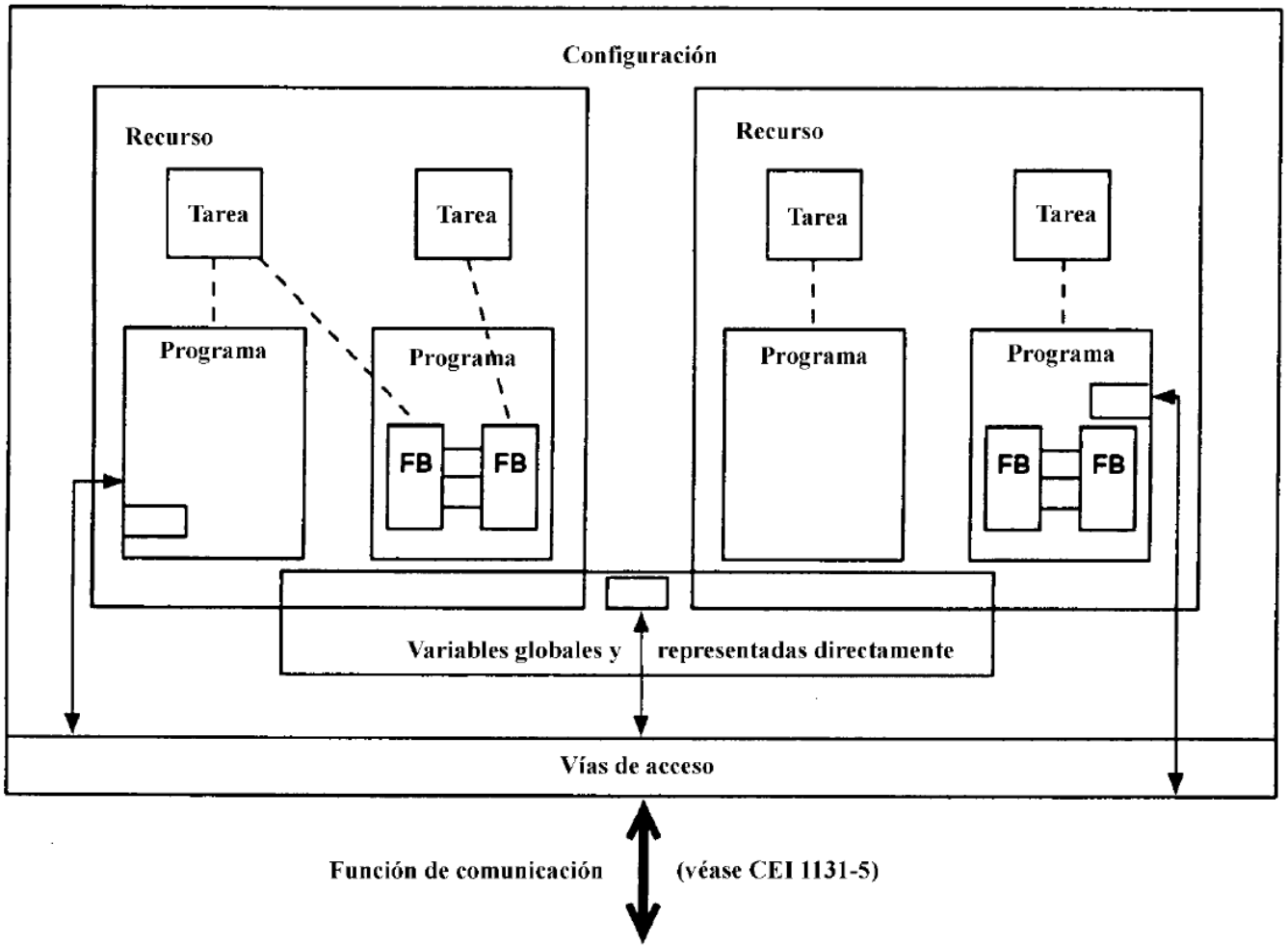
Arquitectura Clásica del PLC

- Arquitectura clásica consiste en ejecución cíclica de programa, con tres fases:
  - Lectura de entradas
  - Ejecución de programa
  - Actualización de salidas
- La ejecución del programa puede alterarse sólo por una interrupción

Arquitectura bajo IEC 61131

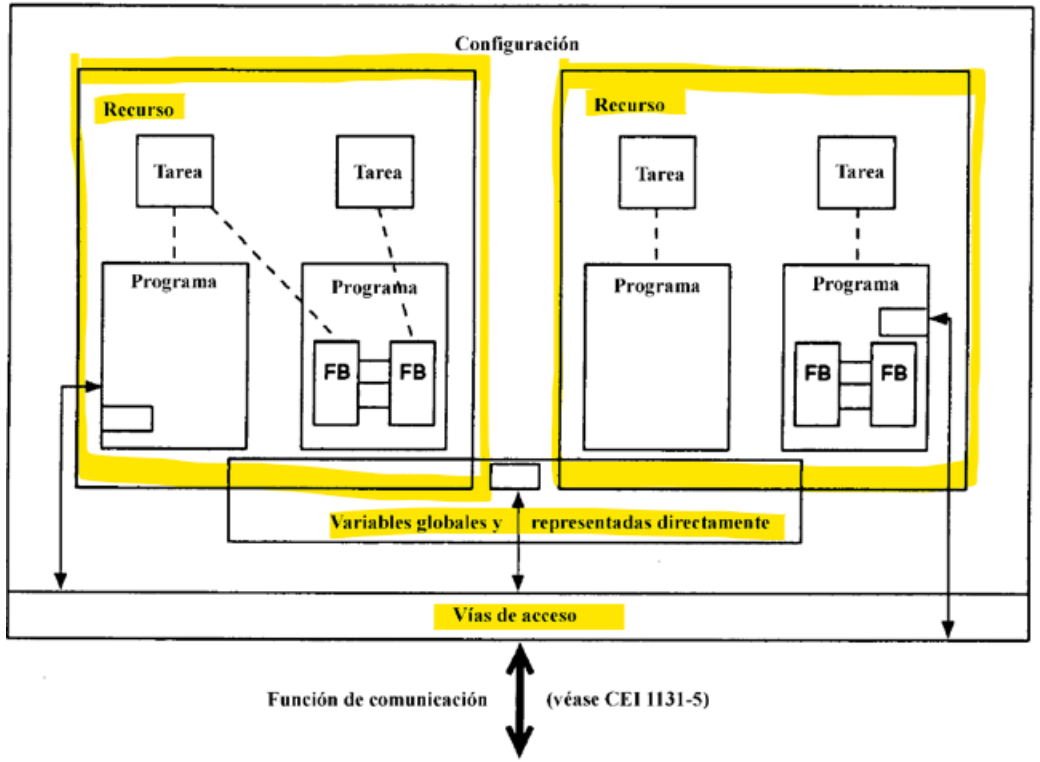
- Por avance continuo del hardware, la IEC-61131 define una arquitectura más avanzada
- La arquitectura se basa en la programación jerárquica, con 4 niveles:
  - Configuración
  - Recursos
  - Tareas
  - Programas

Arquitectura



Configuración

- Más alto nivel del modelo
- Se corresponde con el sistema del PLC
- Generalmente se corresponde con el software necesario para un PLC
- Se comunica con otras configuraciones
- La configuración define:
  - recursos
  - datos compartidos por los recursos
  - datos accesibles desde exterior del PLC

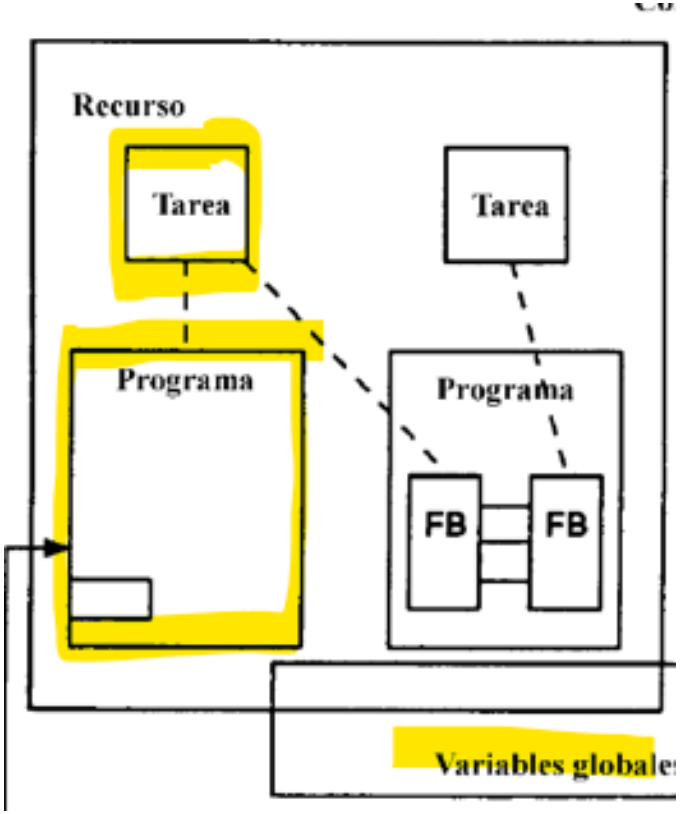


Recursos

- Dentro de una configuración existen uno o más recursos
- Un Recurso proporciona el soporte para ejecutar un programa IEC (símil “máquina virtual”)
- Para que un programa se ejecute, debe ser cargado en un recurso
- Pueden existir en un PLC o en un software para PC por ejemplo
- Un PLC con múltiples procesadores => un recurso por procesador
- Dentro de la configuración, cada recurso debe poder correr en forma independiente

Recursos

- Función principal: interface entre programas y entradas/salidas físicas del PLC
- Recurso define:
  - tareas
  - datos compartidos por todos los programas
  - datos de recurso accesibles desde exterior del PLC
  - programas que ejecutan las tareas del recurso



Tareas

- Configurada para ejecutar un programa o un bloque de función
- Los programas/bloques asociados a una tarea se ejecutan cada vez que se dispara la tarea
- Condición de disparo puede ser:
  - Intervalo periódico de tiempo expresado en mseg (tarea tiene un ciclo asociado, análogo al del PLC)
  - Evento

Tareas

- Un programa sin una tarea asociada no se ejecuta
- Un bloque de función sin una tarea asociada se ejecuta junto con el programa donde reside
- Tarea define:
  - Programas asociados a la tarea
  - Condición de disparo
  - Prioridad

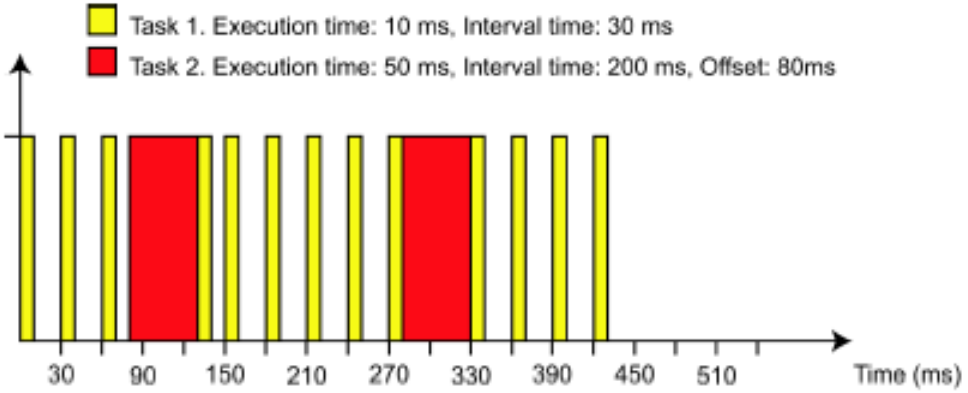
Tareas

- Esquema sin preferencias (Non-preemptive Scheduling)
  - Las tareas no interrumpen a otras tareas
  - Se ejecutan completamente los programas o bloques de una tarea previo a pasar a la siguiente
  - La tarea en espera de mayor prioridad continua la ejecución
  - No recomendado para aplicaciones críticas

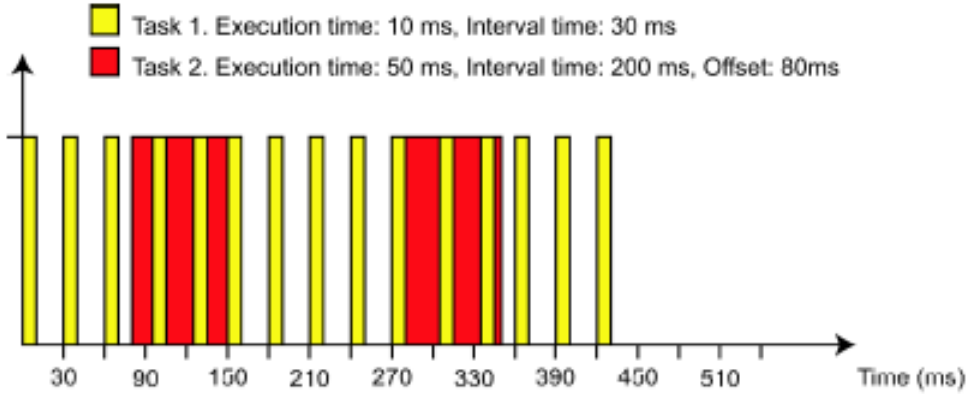
- Esquema con preferencias (Preemptive Scheduling)
  - Las tareas de mayor prioridad interrumpen a las de menor prioridad
  - Para sistemas determinísticos, aplicaciones críticas

Tareas

Ejecución de 2 tareas con misma prioridad o “Non-preemptive”



Ejecución de 2 tareas con diferente prioridad (“Preemptive”)



Tareas

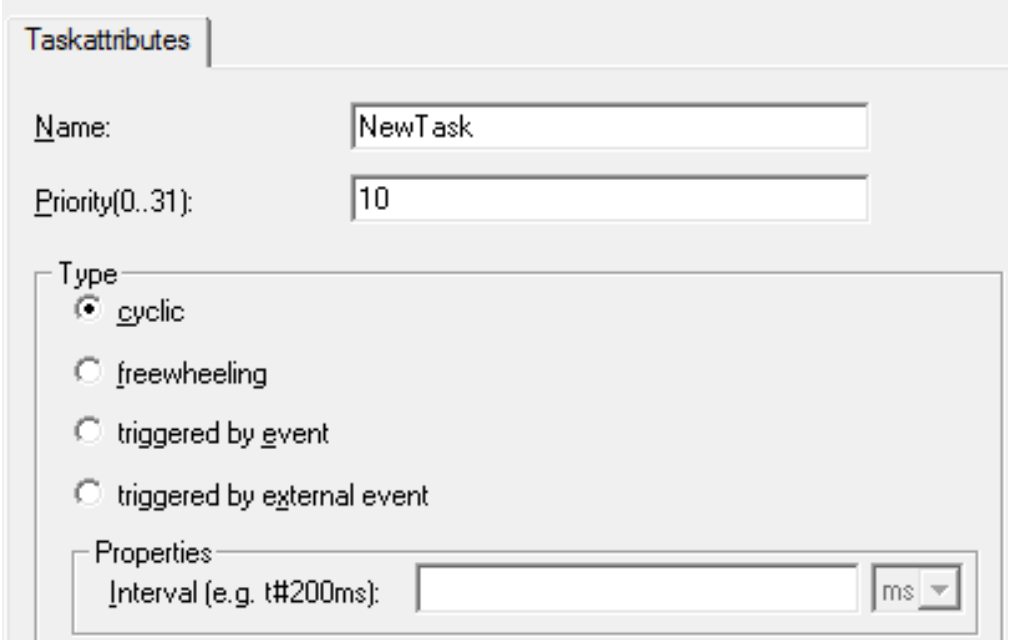
- Ejemplo: declaración de tareas en lab 2 de control de temperatura (se observa la simplificación de la programación):

```
Task Configuration:
  TAREA1(PRIORITY:=1,INTERVAL:=T#100ms)
        PWM
  TAREA2(PRIORITY:=2,INTERVAL:=T#1sec);
        CONTROL
```

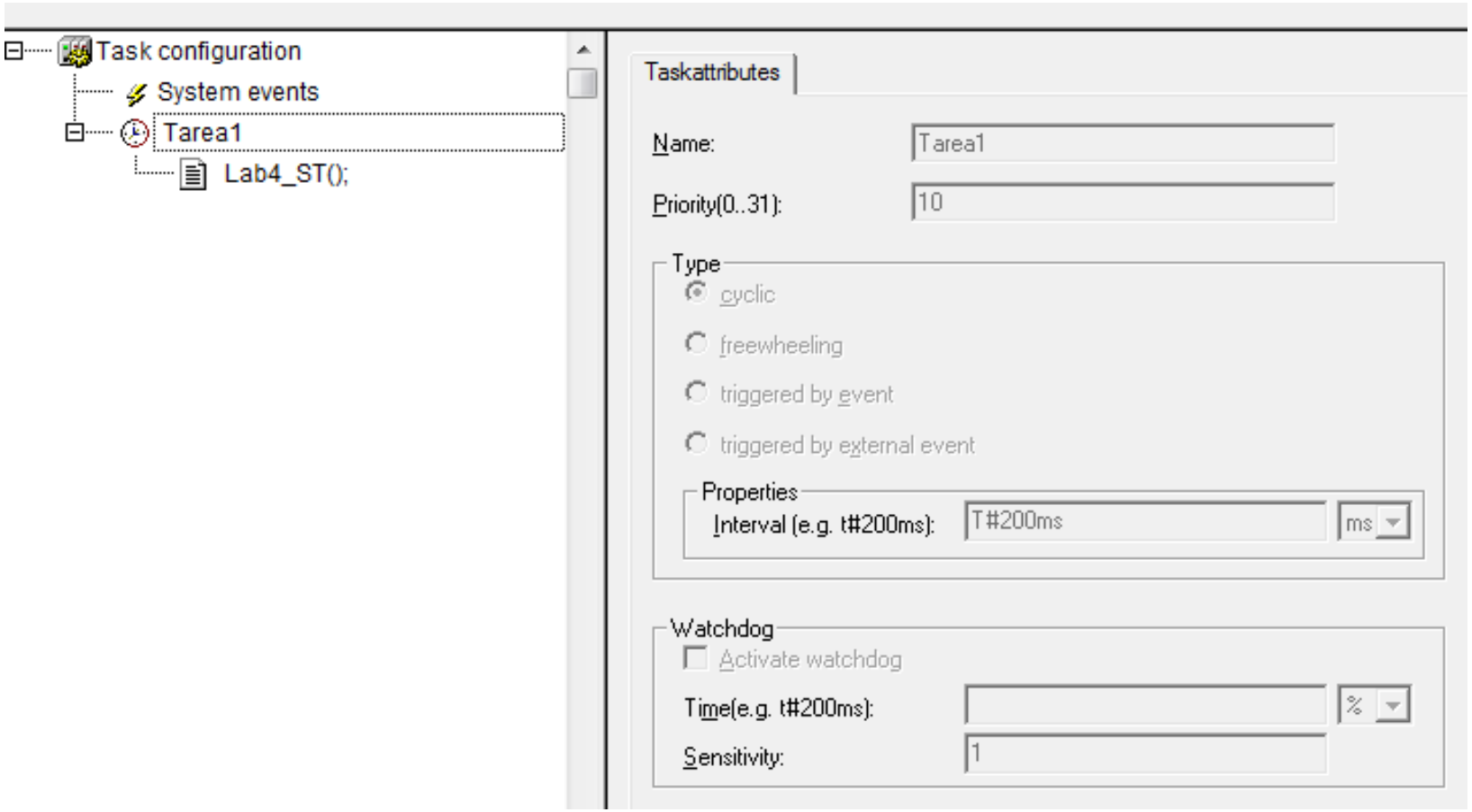
Tareas en PLC de Lab

- Por defecto:
  - Type = cyclic
    Priority = 10
    Cycle time = t#10ms
    Program call= PLC_PRG

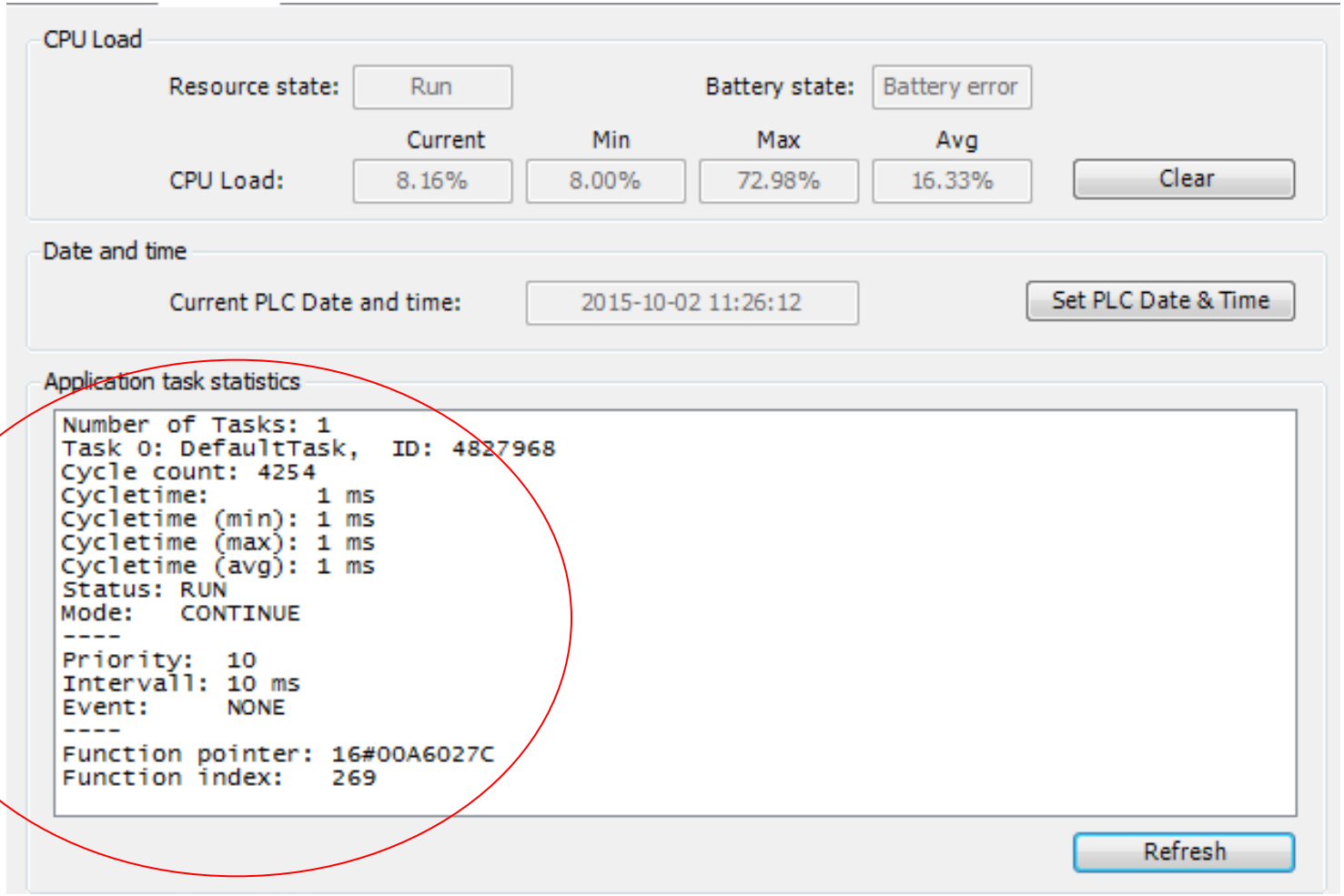
- Nueva Tarea:



Tareas en PLC de Lab



Tarea en PLC de Lab

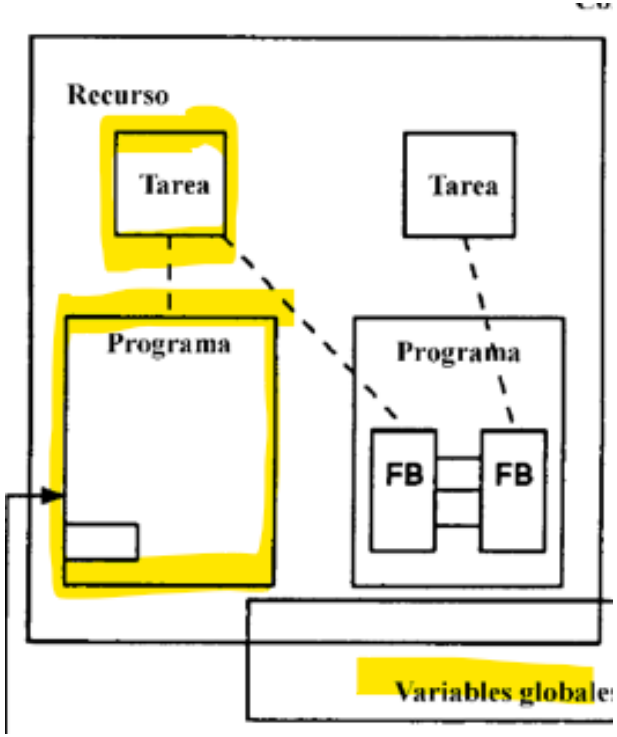


POU (Program Organizational Unit)

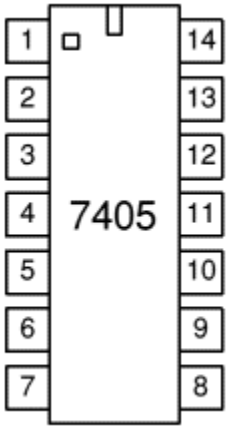
- Hay tres tipos de POU (Program Organizational Unit):
  - Programas
  - Bloques funcionales
  - Funciones

Programas

- Conjunto de elementos de software cada uno definido en base los lenguajes de la norma
- Típicamente, bloques de función interconectados que pueden intercambiar datos
- La ejecución de diferentes partes del programa se puede definir mediante tareas



Bloque Funcional



- Característica más importante de la norma IEC
- Base para diseño jerárquico de software
- Permite descomponer un programa complejo en partes más simples
- Se definen sobre la base de “templates” de bloques
- Para utilizar un template de bloque, la tarea declara una instancia del template
- Bibliotecas propias de usuario (re-uso)

Bloque Funcional

- Se compone de:
  - Datos: parámetros de entrada/salida
  - Código interno
  - Variables internas que representan su estado
- Ejemplos: contadores, PIDs, etc.

Función

- Diferencia entre bloque funcional y función:
  - Número de salidas:
    - Función permite sólo una salida
    - Bloque funcional permite más de una
  - Variables permanentes (conservan valor entre ejecuciones):
    - Función no
    - Bloque funcional sí
- Ejemplos: and, sen, cos, etc.

Modelo vs Sistema Real

- Pequeños PLCs:
  - 1 configuración, 1 recurso, 1 programa
- PLCs más grandes:
  - Configuración = PLC
  - Recursos = procesadores del PLC
  - Cada recurso podrá tener uno o más programas
- Finalmente dependerá de cada implementación

Variables locales/globales

- Variables locales: definidas solo para el programa/bloque de función donde es declarada
- Variables globales:
  - Definidas a nivel de programa: accesible por todos los bloques internos
  - Definidas a nivel de recurso/configuración: accesible por todos los programas incluidos

Variables Directas

- Variables representadas en forma directa: definen su ubicación en memoria en forma explícita
- NombreVar [AT %DirecciónVar] : TipoDeDato [::= ValorInicial];

Variables de Acceso

- Variables designadas para intercambio de datos entre configuraciones
- La norma no define el protocolo de comunicación

Variables

- [Código letras] [Código números]
  - Código de letras: distingue el tipo de dato
  - Primer letra:
    - I/E: Dirección E
    - Q/O: Dirección S
    - M: Dirección dato interno
  - Segunda letra:
    - X: bit
    - B: byte (8 bits)
    - W: word (16 bits)
    - D: double word (32 bits)
    - L: long word (64 bits)

Variables

- Tipo de dato: uno de los tipos de dato reconocido por IEC 61131-3:
  - INT: entero con signo de 16 bits (prefijo S, D, L cambia tamaño)
  - BOOL: Bit
  - BYTE, WORD, DWORD, LWORD
  - REAL, LREAL: punto flotante (32 bit, 64 bit)
  - TIME, DATE, TIME_OF_DAY, DATE_AND_TIME
  - STRING
  - Tipo de datos derivado (STRUCT, ARRAY)

Variables

- Ejemplos tipos derivados por el usuario:

```
TYPE my_array
    ARRAY [1..3] OF INT;
  END_TYPE
```

Variables Estructuradas

```
TYPE motor
  STRUCT
        estado: BOOL
        falla: BOOL
        velocidad: REAL
        corriente: REAL
  END_STRUCT
END_TYPE
```

Uso: M1.estado, M1.falla, etc.

Variables Enumeradas

```
TYPE estado_operativo
  (inicio, ejecución, espera, falla);
END_TYPE
```

Uso: estado1 := falla;