UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Supervisión del tratamiento de templado y revenido de cilindros corrugados mediante el uso de cámara Dahua IR EYEBALL controlada por PLC SIMATIC ET 200SP Open Controller CPU 1505SP F.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: García Navarro, Carlos

Tutor/a: Pitarch Pérez, José Luis

CURSO ACADÉMICO: 2023/2024

Índice

1. Introducción

1.1 Contexto empresarial

El grupo GH es un líder mundial en el campo del calentamiento por inducción para aplicaciones industriales. La empresa matriz, GH Electrotermia, S.A.U., tiene su sede en Valencia, España, y cuenta con más de 50 años de experiencia y conocimientos en esta tecnología innovadora. Con una base sólida de más de 4000 clientes en todo el mundo, el Grupo GH se ha ganado una reputación de confiabilidad y excelencia en el sector.

El grupo se destaca por su amplia experiencia y conocimientos en el calentamiento por inducción. Su historial de más de cinco décadas ha establecido un sólido know-how en el campo, lo que lo convierte en un pionero y líder en esta tecnología.

La empresa presta sus servicios a diversas industrias, lo que le permite generar sinergias beneficiosas para sus clientes. Desde la automotriz hasta la aeroespacial y más allá, el Grupo GH atiende a una amplia gama de aplicaciones industriales.

No solo diseña y fabrica equipos de calentamiento por inducción de alta calidad, sino que también ofrece soluciones completas y personalizadas llave en mano. Además, proporciona un sólido soporte y asistencia técnica durante toda la vida útil de sus sistemas.

La capacidad de I+D en electrónica, aplicaciones e ingeniería le permite desarrollar soluciones específicas para las necesidades de sus clientes. La empresa se esfuerza por convertir las buenas ideas en realidades que aumenten la rentabilidad de sus clientes.

Tiene una presencia global con oficinas y representantes en España, Alemania, Francia, Estados Unidos, India, China, Brasil y México. Además, colabora con licenciatarios y oficinas de servicio técnico en otros países para garantizar tiempos de respuesta rápidos y eficientes.

Un compromiso de la empresa es la innovación constante, trabajando en estrecha colaboración con clientes, instituciones académicas y centros de investigación. Esta colaboración ha llevado a avances tecnológicos notables, como la introducción de generadores en 1986, que ha sido ampliamente adoptada en todo el mundo.

El Grupo GH está comprometido con la calidad y opera bajo la certificación 9001 desde 1995, lo que garantiza que cumple con los estándares más exigentes en sus operaciones.

Es un líder global en tecnología de calentamiento por inducción con una rica historia de innovación y excelencia. Su enfoque en la personalización, la atención al cliente y la calidad ha contribuido a su éxito continuo en una amplia variedad de industrias en todo el mundo.

1.2 Necesidad del cliente

Tratamientos Industriales Iruña es una empresa con una sólida trayectoria en el sector industrial. Fundada en 1973, se ha especializado en el Tratamiento de Temple por Inducción de piezas de acero y fundición, además de proporcionar recubrimientos metálicos, como el carburo de tungsteno, a rodillos utilizados por empresas en diversas industrias, incluyendo la del cemento, papel, minería, química y energía eólica.

El cliente ha comprado una máquina de temple vertical de cilindros corrugados de una longitud de 4500 mm y un peso máximo de 5000 kg, concretamente en la oferta se ha acordado un proceso de "Temple Scanning".

El proceso metalúrgico de templado es una técnica de tratamiento térmico que se utiliza para mejorar las propiedades mecánicas de los materiales metálicos, como el acero. Su objetivo principal es aumentar la dureza y la resistencia del material, mientras se mantiene una cierta tenacidad. Este proceso se logra calentando el material a una temperatura crítica y luego enfriándolo rápidamente en un medio, agua en este caso. El proceso implica las siguientes etapas:

- **Calentamiento:** El material se calienta a una temperatura crítica, que es específica para cada tipo de acero o aleación.
- **Mantenimiento de temperatura:** El material se mantiene a esta temperatura durante un período de tiempo determinado para asegurarse de que toda la pieza alcance la temperatura crítica de manera uniforme.
- **Enfriamiento rápido:** Después del período de mantenimiento, el material se enfría rápidamente sumergiéndolo en un medio de enfriamiento.

En este caso en particular, se aplica un proceso de templado, pero combinándolo con un movimiento de escaneo del cilindro. Debido a que las piezas son de una gran envergadura y peso, su movimiento es altamente complejo, por lo que, en este caso, el movimiento a lo largo de la pieza lo realiza el inductor.

La oferta aceptada, contiene los siguientes elementos de supervisión del proceso de tratamiento de piezas:

- Sistema de control de temperatura en bucle abierto mediante el sistema de dos pirómetros.
- Graficado, control y guardado de los siguientes parámetros operacionales:
    - Frecuencia de salida
    - Potencia de salida
    - Corriente
    - Tensión
    - Energía entregada durante el calentamiento
    - Tiempo de calentamiento
    - Flujo del líquido de temple
    - Temperatura del líquido de temple
    - Tiempo de enfriamiento
    - Rotación de piezas
    - Temperatura de la pieza
    - Posición y velocidad de la pieza
    - Posición y velocidad de la bobina

- Dos cámaras para procesamiento de imagen industrial, de forma que se puedan controlar y grabar los puntos de la estación solicitados por TI Iruña, junto con sus soportes y material necesario para la instalación.

Como se puede observar, el cliente requiere tres métodos de supervisión del proceso de calentamiento de sus piezas.

Esta memoria de TFG se centrará en el tercer elemento mencionado, las cámaras. Cada una de ellas tiene un objetivo distinto, una se encargará de vigilar el contrapunto que mantiene el cilindro asegurado y la otra vigilará el trabajo del inductor, en concreto y para el objetivo de este documento, esta segunda cámara será la razón de este proyecto.

1.3 Antecedentes

Por último, este proyecto está estrechamente relacionado con la formación académica del grado en Ingeniería Electrónica Industrial y Automática. A través del programa de estudio, se aplican y amplían los conocimientos adquiridos en electrónica, automatización, sistemas de control y tecnologías de la información. Estos conocimientos proporcionan la base necesaria para abordar con éxito la implementación de un sistema de grabación y documentación en un entorno industrial complejo.

Por ejemplo:

- Comprender y trabajar con sistemas de control, como el PLC Siemens, que son fundamentales para la automatización de procesos industriales.
- Desarrollar aplicaciones y sistemas de software, lo cual es esencial para crear la interfaz que permita la comunicación entre PLC y cámara.
- Evaluar y seleccionar componentes electrónicos y tecnologías de grabación de video adecuadas para este proyecto.
- Diseñar soluciones técnicas sólidas que cumplan con los objetivos y requisitos específicos de la empresa.

2. Objeto

El objeto de este proyecto es el desarrollo e implementación de un sistema integral de grabación de video y documentación que permita supervisar de manera efectiva el proceso de tratamiento de cilindros corrugados en la empresa "Tratamientos Industriales Iruña" (TI Iruña). Este sistema se diseñará para garantizar la trazabilidad, calidad y transparencia del proceso de tratamiento de piezas de gran envergadura.

El proyecto tiene como objetivo central proporcionar a TI Iruña una solución tecnológica que permita la grabación y documentación de cada etapa del proceso de tratamiento de cilindros corrugados.

La implementación de este sistema abordará las necesidades específicas de la empresa, considerando su inversión reciente en una máquina vertical de templado de cilindros corrugados de gran capacidad. Lo fundamental es optimizar la supervisión y la gestión del proceso de tratamiento, así como ofrecer un registro detallado y organizado de las operaciones propósito para mejorar la trazabilidad y satisfacer las expectativas de los clientes.

El TFG se enfocará en el desarrollo de un programa de grabación de vídeo que sea compatible con las máquinas de tratamiento y su integración con el sistema de control basado en PLC Siemens de la empresa. Además, se establecerán comunicaciones efectivas entre el PLC y la cámara Dahua que permitirán el control y la documentación del proceso. La organización y almacenamiento eficiente de las grabaciones, identificando cada pieza de manera única según su número de pieza, fecha y estado, es componente crucial de la solución.

Los objetivos específicos del proyecto están diseñados para lograr el propósito general y abordar las necesidades particulares de TI Iruña en la implementación del sistema de grabación:

- Desarrollar un sistema de grabación de video compatible con las máquinas de tratamiento de cilindros corrugados. Esto implica seleccionar las cámaras adecuadas, configurar sus parámetros y garantizar una grabación precisa de las operaciones.
- Integrar el sistema con el PLC siemens. Este objetivo busca permitir la supervisión y control del sistema de grabación a través del PLC, lo que garantizará una coordinación eficiente en el proceso de tratamiento. También seleccionar el modelo de PLC que más se ajuste a las necesidades del proyecto.
- Establecer comunicaciones efectivas entre el PLC y la cámara Dahua, esencial para el control y la documentación del proceso. Este objetivo implica desarrollar una aplicación que funcione como intermediario y que sea capaz de enviar comandos "web service" a la cámara para activar la grabación y gestionar la documentación.
- Organizar y almacenar las grabaciones de manera eficiente, creando carpetas específicas para cada pieza, identificadas, como ya se ha mencionado, por número de pieza, fecha y estado de esta.

La implementación exitosa de los objetivos específicos se traducirá en una serie de beneficios esperados para el cliente.

- La documentación completa y precisa del proceso de tratamiento garantiza una mejora significativa en la calidad de los cilindros corrugados y la trazabilidad de las operaciones realizadas a cada pieza.

- Capacidad de identificar áreas de mejora y optimización en el proceso de tratamiento conducirá a una mayor eficiencia operativa, lo que a su vez puede reducir costos y tiempos de producción.
- Con una capacidad mejorada para satisfacer las demandas de los clientes y ofrecer servicios de alta calidad, TI Iruña reforzará su competitividad en un mercado industrial altamente competitivo.

Por otro lado, la implementación exitosa de los objetivos específicos también se traducirá en una mejora en el punto 9 de los Objetivos de Desarrollo Sostenible "Industria, innovación e infraestructura" en los siguientes términos.

- **Promoción de la innovación tecnológica:** La aplicación de tecnologías como la comunicación por sockets TCP/IP y el futuro uso de servidores OPC UA demuestran un enfoque innovador para mejorar los procesos industriales.
- **Desarrollo de infraestructuras digitales:** Al integrar sistemas de control basados en PLC y pantallas HMI, el proyecto está contribuyendo al desarrollo de infraestructuras digitales en entornos industriales. Estas infraestructuras son esenciales para mejorar la eficiencia, productividad y la capacidad de respuesta en las operaciones industriales.
- **Mejora de la eficiencia y la sostenibilidad:** Al optimizar los procesos de control y monitoreo en el contexto industrial, el proyecto contribuye a la mejora de la eficiencia operativa de reducción del consumo de recursos y, por tanto, reducciones de desperdicios y emisiones.

3. Análisis de alternativas y selección de componentes

La elección del PLC y de la cámara Dahua dependerá de diversos factores, incluyendo las necesidades específicas de TI Iruña, las características técnicas requeridas y las opciones disponibles de mercado.

Selección de PLC.

En primer lugar, debido a lo mencionado en el punto 1.2, el cliente requiere un graficado, control y guardado de una serie de parámetros. La aplicación/función desarrollada por GH que lo permite realizar se llama IPM (Induction Process Monitoring).

Esta aplicación/función (se denomina de esta forma ya que puede estar integrada dentro del PLC o externamente) se encarga de evaluar la calidad de un proceso de calentamiento en base a los valores muestreados durante el proceso y los parámetros especificados. El IPM muestrea las señales cada 10 ms, completando un vector con sus valores, una vez terminado el proceso de calentamiento, *plotea* estos puntos y dibuja una gráfica, independiente para cada parámetro. Previamente, se le deben introducir una serie de tolerancias para que así pueda calcular los límites superiores e inferiores de las tendencias representadas en las gráficas. De esta forma, una vez se ha hecho un calentamiento de prueba, para que pueda calcular los límites (este proceso se denomina patronaje), la forma de determinar si una pieza es buena o mala es comprobar que cada punto de la gráfica entra dentro de los límites calculados. Por último, el IPM guarda un registro de cada pieza, indicando si ha sido buena o mala, la fecha y el valor medio o valor de cada punto del *array* de cada parámetro.

El IPM requiere un alto nivel de computación y, por lo tanto, de consumo de CPU. Esto es fundamental a la hora de escoger el PLC ya que, si se sobrecarga su CPU, es posible que un ciclo de "scan" se extienda de forma inaceptable y se pierda parte de la información. Teniendo en cuenta esto, hay una clara necesidad de integrar un PC con el PLC.

Dentro de las opciones disponibles en el catálogo de siemens, existen dos posibilidades: escoger un "Open Controller" (PLC con PC Integrado), o un PLC de la serie 1500 con un PC industrial. Ambas opciones cumplen con el requisito computacional del IPM, pero existen diferencias sustanciales.

Open Controller	**PLC + PC Industrial**
Debido a que integran computadoras industriales con un sistema operativo completo, son altamente flexibles y versátiles.	Ofrecen un rendimiento fiable y más robusto.
Más variedad de aplicaciones y programas.	Implementación más fluida.
Más económicos en términos de hardware y software.	

El objetivo es buscar una solución económica, versátil y escalable para el proyecto. La flexibilidad y la capacidad de adaptación del controlador debe brindar una base sólida para diseñar un sistema que cumpla con los requisitos actuales y futuros de TI Iruña. Esta elección se debe alinear con la optimización de recursos y la capacidad de adaptarse a las necesidades cambiantes de la empresa en un entorno industrial dinámico. Por todas estas razones, la opción más adecuada para el proyecto es la de un "Open Controller".

Por razones ajenas a este proyecto (cantidad de servos, motores, seguridades, etc), se ha escogido un "ET 200SP Open Controller CPU 1515SP PC", referencia 6ES7 677-2DBx2-0xx0 y versión 21.9. Esta versión incluye también una pantalla HMI con la que el operario podrá manejar las funcionalidades de la máquina.

Hay otras empresas que también ofrecen PLCs con capacidad de PC. Por ejemplo, estos PLCs tienen una partición Linux donde se puede ejecutar algoritmos programados incluso en Matlab. [1]

Selección cámara Dahua.

Dentro del catálogo de cámaras que ofrece la empresa Dahua, existen una serie de características relevantes para el objetivo de este proyecto. [2]

- **Tipo de cámara:** Para supervisar un proceso de tratamiento industrial, es importante que la cámara sea robusta y resistente, tipo bala o domo.
- **Resolución de video:** En el entorno industrial, una resolución de al menos 1080p o superior es lo recomendable.
- **Resistencia a condiciones ambientales:** Deben ser resistentes a condiciones ambientales adversas, polvo, humo y cambios de temperatura.
- **Compatibilidad de protocolos:** Debe ser compatible con los protocolos de comunicación que se utilizarán para controlar la grabación y otras funciones a través del PLC.

Una opción viable económicamente y que cumple con todos los requisitos es la cámara "IR EYEBALL NETWORK CAMERA H.265+", robusta y de tipo domo, con una resolución de 1080p o superior ya que es configurable, resistente a temperaturas elevadas y altas concentraciones de humo y que además cuenta con servicios web para poder comunicarse con ella desde una aplicación externa.

Incluir un sistema de almacenamiento externo en la selección de componentes es una consideración importante, especialmente para la aplicación diseñada que debe almacenar grandes cantidades de datos. Por lo tanto, se deben tener en cuenta las siguientes características:

- **Capacidad de almacenamiento adecuada:** Debe ser un sistema que tenga suficiente capacidad de almacenamiento. TI Iruña trabaja con una producción baja en cuanto a número de piezas, pero de proceso largo. Por lo que los videos que se almacenarán, a pesar de ser pocos a lo largo del día, serán de gran envergadura.
- **Velocidad de transferencia de datos:** Un sistema con una velocidad de transferencia de datos rápida para garantizar una transferencia eficiente de archivos entre el Open Controller y el sistema de almacenamiento externo.

- **Conectividad:** El sistema de almacenamiento debe ser compatible con la conectividad del entorno de trabajo, el Open Controller. Conexión USB, Ethernet u otras opciones.
- **Fiabilidad y durabilidad:** El sistema de almacenamiento debe ser confiable y duradero, capaz de mantener la integridad de los datos en entornos industriales o condiciones adversas (calor, humo, altas tensiones, etc).

Teniendo en cuenta todas estas especificaciones, una opción viable y económica dentro del mercado es el disco duro "SSD SanDisk Extreme Portable de 1TB", una solución robusta y confiable para almacenar videos y otros datos. Con 1 TB de almacenamiento, la tecnología SSD que ofrece lecturas y escrituras rápidas, compatible con USB 3.1 Gen 2 para una conexión rápida y fiable y construido para resistir condiciones adversas como golpes, vibraciones, temperaturas extremas y agua.

La selección de los componentes para este proyecto representa una decisión clave en el diseño de un sistema de control industrial eficiente y versátil. Al concluir este proceso, se ha establecido una base sólida para el éxito de la implementación del cliente.

En conjunto, esta selección refleja una cuidadosa consideración de las necesidades técnicas y presupuestarias del cliente, y sienta las bases para la implementación exitosa de un sistema de control estricto y documentación eficiente.

4. Descripción detallada de la solución adoptada

El esquema de flujo de información ofrece una representación visual de cómo se intercambia y comunica la información entre los diversos elementos hardware y software del proyecto. Este esquema destaca las relaciones y conexiones clave entre el Open Controller, la máquina de tratamiento de cilindros corrugados y la cámara Dahua.

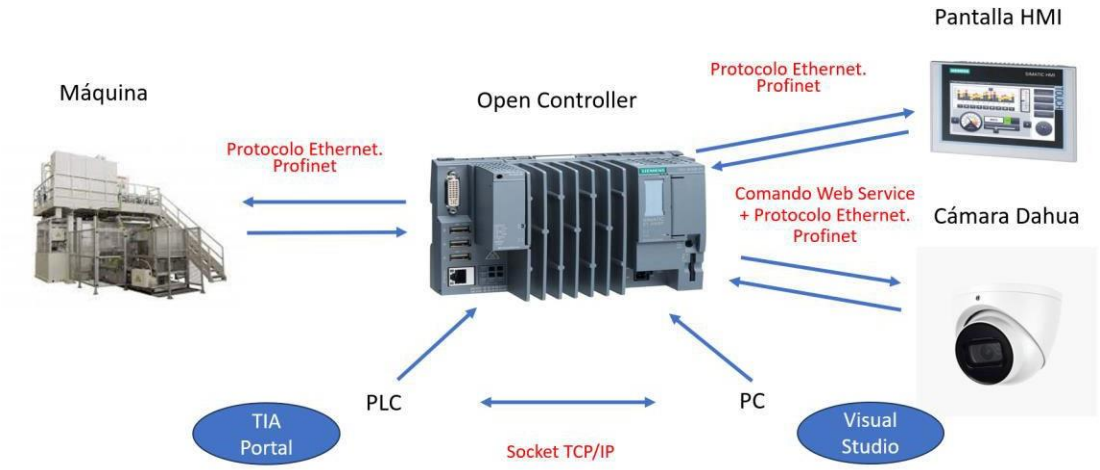


Figura 1: Esquema hardware y software del sistema.

En el esquema de la figura 1 se identifican los siguientes elementos y sus interacciones:

Open Controller: Este componente es el núcleo del sistema de control y supervisión. Actúa como el controlador principal que coordina y ejecuta las operaciones del proceso de templado y revenido de cilindros corrugados. El PLC intercambia información tanto con el PC como con la máquina de tratamiento, permitiendo la sincronización y supervisión efectivas del proceso. Gracias al *Socket TCP/IP*, que se explicará a continuación, como un intermediario entre el PLC y el PC, se permite la comunicación bidireccional entre estos componentes. Debido a los programas desarrollados en Visual Studio y TIA Portal, se intercambian información PC y PLC a través de paquetes de datos, para recibir comandos y datos relacionados con el proceso de tratamiento, y a su vez, el PC envía comandos a la cámara Dahua para controlar la grabación de video y la adquisición de datos.

Paquetes de datos relevantes	**Descripción**
Pieza OK/NOK	Indica si la pieza se ha tratado correctamente o no.
Fecha y hora inicial	Indica la fecha y la hora en la que se empezó el tratado de la pieza.
Fecha y hora final	Indica la fecha y la hora en la que se terminó el tratado de la pieza.
Número de pieza	Es un contador que indica el número de piezas tratadas, particular para cada pieza.

Máquina de Tratamiento de Cilindros Corrugados: Es el equipo físico encargado de llevar a cabo el proceso de templado y revenido de los cilindros corrugados. El PLC se comunica directamente con la máquina de tratamiento, a través de Profinet protocolo de comunicación Ethernet, para enviar comandos de control y supervisar el estado del proceso.

Cámara Dahua: La cámara Dahua se conecta al PC para recibir comandos de inicio y detención de grabación, así como para transmitir datos de video en tiempo real. Esta comunicación se establece a través de los comandos *web service* creados en el programa de Visual Studio y de manera física a través de Profinet.

Pantalla HMI: Interfaz de usuario para el sistema de control y monitoreo. Los comandos de aviso diseñados en el programa de Visual Studio se verán reflejados en esta pantalla. Se comunica con el Open Controller a través de Profinet.

La manera en la que se han realizado las comunicaciones es mediante el uso de un *socket*, un mecanismo de comunicación bidireccional que permite que dos procesos en distintos dispositivos intercambien datos. En el contexto de desarrollo software, un socket es una interfaz de programación de aplicaciones (API) que facilita la comunicación entre aplicaciones a través de una red, como Internet o una red local.

En un sistema de automatización industrial como el descrito, el uso de *sockets* es fundamental por varias razones:

1. **Comunicación entre dispositivos:** En este contexto, el PLC actúa como el cliente, mientras que el programa desarrollado en Visual Studio funciona como el servidor. El PLC, como cliente, establece una conexión con el servidor (programa en Visual Studio) para enviar y recibir datos.
2. **Control remoto:** El PLC, en su rol de cliente, envía solicitudes al servidor (programa en Visual Studio) a través del socket para controlar el proceso de tratamiento de cilindros corrugados. Por ejemplo, el cliente (PLC) puede enviar comandos al servidor para iniciar o detener la grabación de video.
3. **Transmisión de datos en tiempo real:** Los sockets permiten la transmisión de datos en tiempo real entre el cliente (PLC) y el servidor (programa en Visual Studio). Esto asegura que la supervisión del proceso de tratamiento de cilindros corrugados sea precisa y oportuna, ya que los datos se transmiten de manera rápida y eficiente entre los dispositivos.
4. **Flexibilidad y escalabilidad:** El uso de sockets proporciona una solución flexible y escalable para la comunicación entre dispositivos. Esto permite la integración de nuevos componentes o la expansión del sistema en el futuro sin necesidad de reescribir gran parte del código.

Por lo tanto, el objetivo dentro del desarrollo de ambos programas (TIA Portal y Visual Studio), es el de crear un cliente y un servidor funcionales, que puedan controlar las prestaciones de la cámara. [3]

4.1 Solución en TIA Portal.

La solución adoptada en TIA Portal se centra en la configuración del Open Controller ET 200SP 1515SP PC, aprovechando las potentes capacidades de programación y personalización que ofrece este entorno de desarrollo. Este proceso incluye la definición de las E/S, la configuración de la red, y la programación de la lógica de control que coordinará el tratamiento de cilindros corrugados y la grabación de video a través de la cámara Dahua.

En esta fase inicial, nos sumergiremos en la interfaz intuitiva de TIA Portal para establecer una comunicación efectiva entre el PLC y los dispositivos periféricos. La estructura modular del programa se diseña cuidadosamente, garantizando la coherencia y la eficacia en la ejecución de las operaciones programadas.

Asimismo, aprovecharemos las funcionalidades avanzadas de TIA Portal para implementar protocolos de comunicación eficientes, permitiendo que el PLC interactúe sin inconvenientes con la cámara Dahua.

Para facilitar una compresión clara y visual de la solución implementada en TIA, se emplearán diagramas de flujo. Estas representaciones gráficas permitirán desglosar de manera detallada la secuencia de operaciones. A través de estos esquemas, se ofrecerá una visión paso a paso de cómo el Open Controller coordina eficientemente las acciones de la máquina y la cámara Dahua, destacando la lógica de control y las conexiones clave que facilitan el proceso integrado. Este enfoque visual no solo simplificará la comprensión técnica de la solución, sino que también proporcionará una referencia clara para el diseño, la implementación y el mantenimiento continuo del sistema de automatización industrial propuesto.

Estos son los bloques funcionales más utilizados en el programa desarrollado. [4]

OB (Bloque Organizativo):

- Este bloque se utiliza para la programación del ciclo de usuario en el PLC. Contiene la lógica de control principal del programa y se ejecuta cíclicamente.

FC (Bloque de función):

- Los bloques de función contienen funciones reutilizables que pueden ser llamadas desde otros bloques. Son útiles para modular el código y facilitar la reutilización.

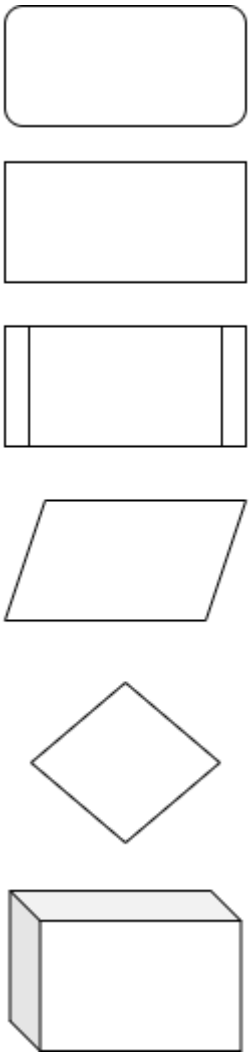
FB (Bloque de función)

- Similar a los bloques de función (FC), los bloques de función (FB) también contienen funciones reutilizables, pero pueden tener variables internas. Esto permite que los FBs mantengan un estado interno entre las llamadas.

DB (Bloque de datos):

- Los bloques de datos se utilizan para definir estructuras de datos y variables. Pueden contener tipos de datos definidos por el usuario y se utilizan para almacenar información utilizada por el programa.

Los bloques que se verán a continuación en los diagramas de flujo corresponden a lo siguiente:



-Inicio o final de la ejecución de un conjunto de segmentos de TIA Portal.

-Segmento de TIA Portal.

-Conjunto de segmentos de TIA Portal.

-Función propia de TIA Portal.

-Conjunto de segmentos en TIA Portal en los que dependiendo de una variable, se ejecutará un segmento u otro.

-Función creada externamente de TIA Portal (FC o FB).

Figura 2: Leyenda de bloques de diagrama de flujo.

4.1.1 Bloque OB1 (Main)

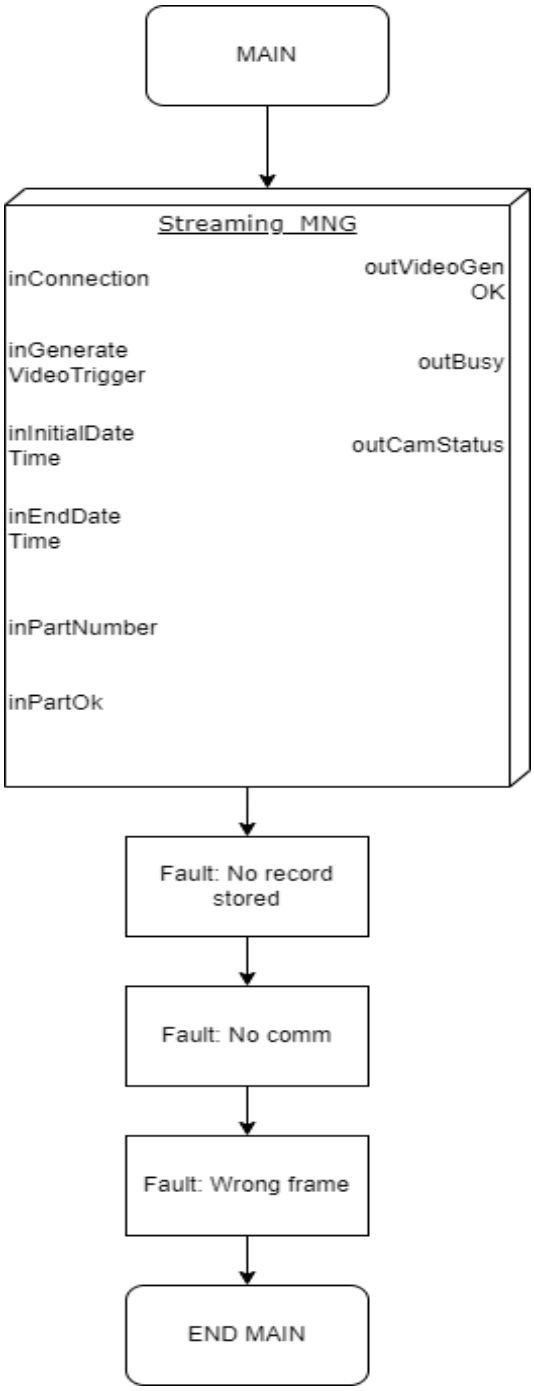


Figura 3: Estructura general del programa implementado en TIA Portal.

1. **Incio del Bloque OB1 (Main):**

Este es el bloque organizativo principal (OB1). Este es el punto de incio de la ejecución del programa completo y, que, por tanto, iniciará la parte de supervisión de la cámara.

2. **Llamada a la FC Streaming_MNG:**

Dentro del bloque OB1, se realiza una llamada a la función "Streaming_MNG". Esto indica que el flujo de ejecución del programa pasa a la función específica.

La función se encarga de la gestión completa de la cámara y tiene unas entradas (inputs) y salidas (outputs) específicas. Estas entradas y salidas determinan el comportamiento y la interacción de la función con otras partes del programa. La función realiza operaciones como la gestión de transmisiones de datos, procesamiento de información y control del dispositivo relacionado con el video.

3. **Retorno de 3 Fallos:**

Después de ejecutar la función, esta indicará como ha resultado el proceso gracias a sus salidas.

- outVideoGenOK (1=Sí;0=NO): Indica el estado del video, si ha sido creado correctamente o no.
- outBusy (1=Sí;0=NO): Indica si la función se está ejecutando o no.
- outCamStatus: Indica si ha ocurrido algún tipo de problema durante la ejecución de la función, contempla 3 posibles fallos explicados a continuación.

  -outCamStatus = 1: No se ha guardado ningún video.

  -outCamStatus = 2: No hay comunicación con la cámara.

  -outCamStatus = 3: La trama de datos enviada no tiene la estructura correcta.

4. **Final de bloque OB1 (Main):**

Después de la ejecución de la función y la gestión de fallos, se finaliza este bloque principal.

4.1.1.1 FC “Streaming_MNG”

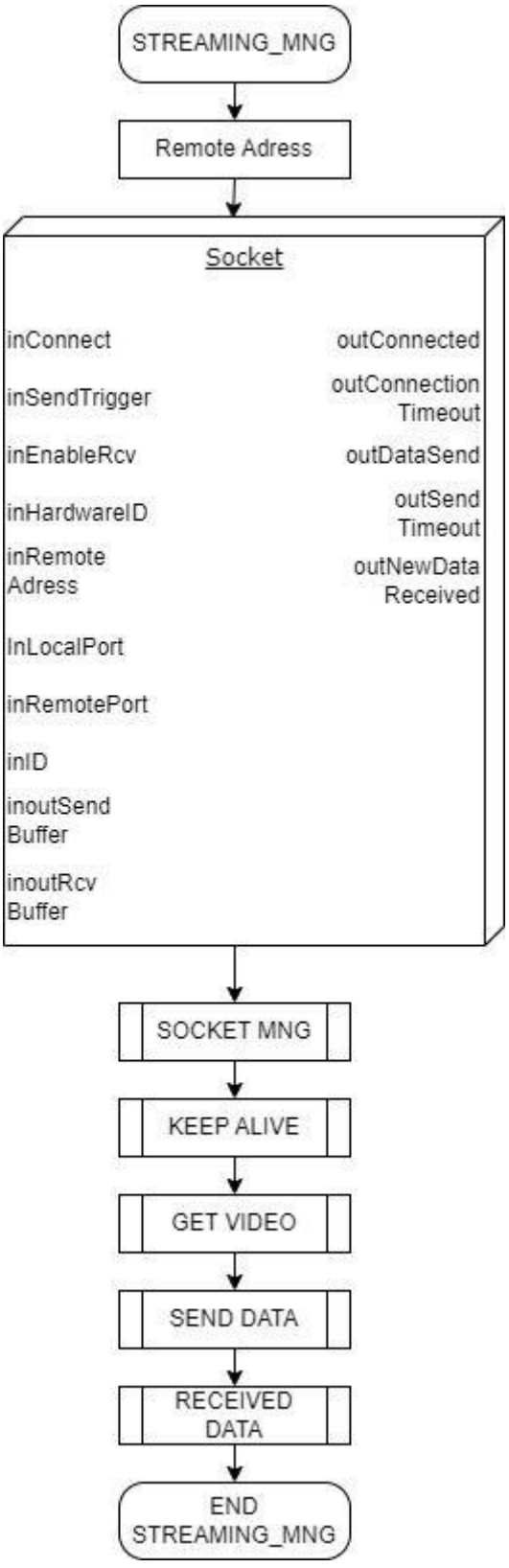


Figura 4: Estructura de la FC “Streaming_MNG” implementada en TIA Portal.

1. **Inicio de la Función Streaming_MNG:**

Este es el punto de entrada de la función Streaming_MNG. Aquí comienza la ejecución de la lógica específica que se ha definido para gestionar el streaming.

2. **Remote Address:**

En este segmento de código se incluye la IP de la cámara Dahua.

3. **Función General Socket (FB):**

Esta función contiene la lógica necesaria para manejar la apertura, cierre y configuración de las comunicaciones entre el PLC y la aplicación en Visual Studio.

4. **Llamada al subconjunto Socket_MNG:**

Aquí se realiza la solicitud de inicio y la confirmación del inicio de la FB anterior.

5. **Llamada al subconjunto Keep Alive:**

Este subconjunto se encarga de enviar señales periódicas para mantener viva la conexión, evitando la desconexión por inactividad.

6. **Llamada al subconjunto Get Video:**

Este subconjunto involucra la lógica necesaria para obtener el video.

7. **Llamada al subconjunto Send Data:**

Este subconjunto involucra la lógica de enviar las tramas de datos a través de la conexión establecida.

8. **Llama al subconjunto Received Data:**

Este subconjunto maneja la lógica asociada con la recepción y procesamiento de las tramas de datos recividas.

9. **Final de la función Streaming_MNG:**

En este punto concluye la ejecución de la función.

La función creada “Socket”, así como los subconjuntos, serán explicados a continuación.

4.1.1.1.1 “Socket”

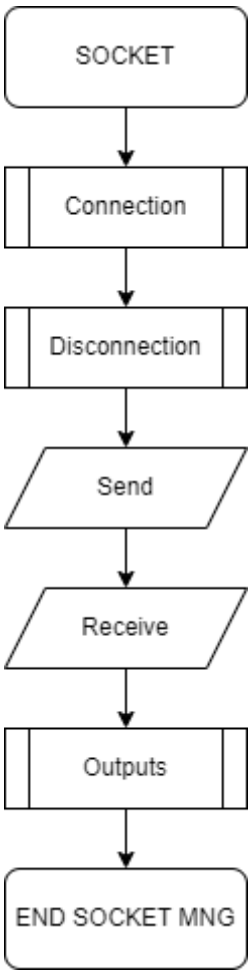


Figura 5: Estructura de la FB “socket” perteneciente a la FC “Streaming_MNG”.

1. **Inicio de la función Socket:** Este es el punto de inicio de la función Socket, donde comienza la ejecución de la lógica para establecer y gestionar la conexión a través del socket.

2. **Llamada al subconjunto Connection:** Se efectúa una llamada al subconjunto que maneja la conexión. En este punto, se establece la conexión con el servidor para iniciar la comunicación.

3. **Llamada al subconjunto Disconnection:** Se realiza una llamada al subconjunto encargado de la desconexión. Aquí se cierra la conexión establecida previamente con el servidor.

4. **Bloque de TIA Portal Send:** Este bloque representa la acción de enviar datos a través del socket. Es utilizado para enviar comandos, solicitudes u otra información al servidor.

5. **Bloque de TIA Portal Receive:** Este bloque indica la recepción de datos a través del socket. Se encarga de recibir las respuestas o los datos enviados por el servidor en respuesta a las solicitudes del cliente.

6. **Llamada al subconjunto Outputs:** Se efectúa una llamada al subconjunto que gestiona los datos de salida. Aquí se procesan los datos recibidos del servidor y se utilizan para la lógica de control.

4.1.1.1.1.1 “Connection”

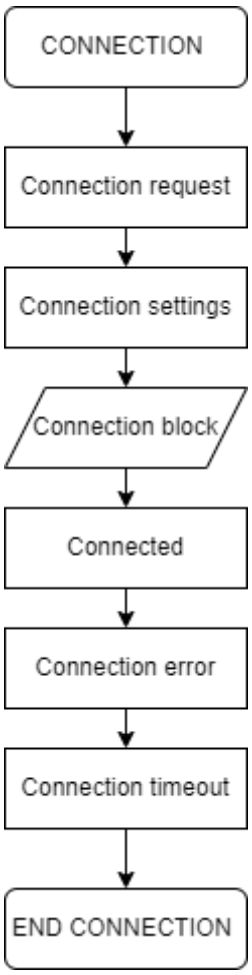


Figura 6: Estructura del subconjunto “CONNECTION” perteneciente a la FB “Socket”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Connection, donde comienza la ejecución de la lógica para establecer la conexión con el servidor.

2. **Ejecución del Segmento Connection Request:** En esta fase, se ejecuta el segmento encargado de enviar una solicitud de conexión al servidor. Esta solicitud es el primer paso para iniciar el proceso de establecimiento de la conexión.

3. **Ejecución del Segmento Connection Settings:** Aquí se lleva a cabo la configuración de los parámetros de conexión, como la dirección IP del servidor, el número de puerto, el protocolo de comunicación, etc. Estos ajustes son necesarios para establecer una conexión exitosa.

4. **Ejecución del Bloque de TIA Portal Connection Block:** En este punto, se ejecuta el bloque de TIA Portal dedicado a establecer la conexión con el servidor. Este bloque

contiene la lógica específica para configurar y abrir la conexión utilizando los parámetros previamente establecidos.

5. **Ejecución del Segmento Connected:** Si la conexión se establece con éxito, se ejecuta este segmento. Aquí se realizan las acciones necesarias una vez que se confirma que la conexión está activa y lista para la comunicación.

6. **Ejecución del Segmento Connection Error:** Si ocurre algún error durante el proceso de conexión, se ejecuta este segmento. Aquí se manejan los errores de conexión y se toman las medidas adecuadas para notificar al usuario o para intentar reconectar.

7. **Ejecución del Segmento Connection Timeout:** Este segmento se ejecuta si se produce un tiempo de espera durante el proceso de conexión. Esto puede suceder si el servidor no responde dentro del tiempo esperado. En este caso, se ejecutan las acciones correspondientes para manejar la situación de tiempo de espera.

8. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto Connection.

4.1.1.1.1.2 “Disconection”

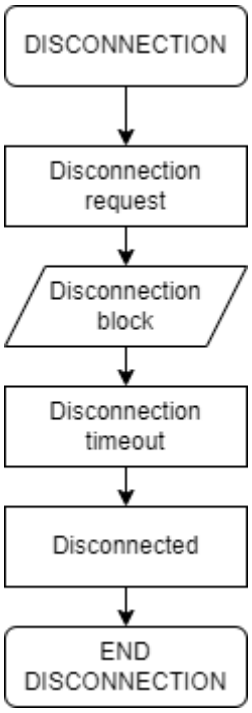


Figura 7: Estructura del subconjunto “DISCONNECTION” perteneciente a la FB “Socket”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Disconnection, donde comienza la ejecución de la lógica para desconectar del servidor.

2. **Ejecución del Segmento Disconnection Request:** En esta fase, se ejecuta el segmento encargado de enviar una solicitud de desconexión al servidor. Esta solicitud es el primer paso para iniciar el proceso de desconexión.

3. **Ejecución del Bloque TIA Portal Disconnection Block:** Aquí se lleva a cabo la desconexión real del servidor utilizando el bloque de TIA Portal diseñado para este propósito. Este bloque contiene la lógica específica para cerrar la conexión de manera adecuada y ordenada.

4. **Ejecución del Segmento Disconnection Timeout:** Si se produce un tiempo de espera durante el proceso de desconexión, se ejecuta este segmento. Esto puede suceder si el servidor no responde a la solicitud de desconexión dentro del tiempo esperado.

5. **Ejecución del Segmento Disconnected:** Una vez que se completa el proceso de desconexión con éxito, se ejecuta este segmento. Aquí se realizan las acciones necesarias una vez que se confirma que la conexión se ha cerrado correctamente.

6. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto Disconnection.

4.1.1.1.1.3 “Outputs”

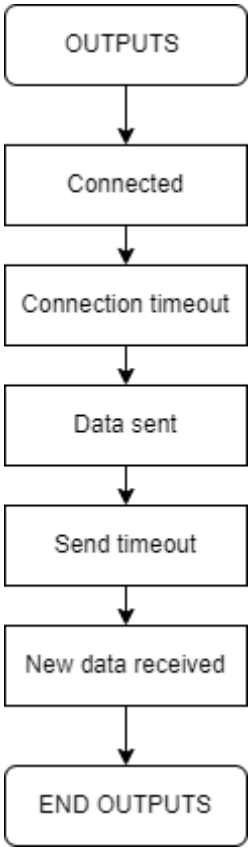


Figura 8: Estructura del subconjunto “OUTPUTS” perteneciente a la FB “Socket”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Outputs, donde comienza la ejecución de la lógica para gestionar las salidas de la función socket.

2. **Ejecución del Segmento Connected:** Este segmento se encarga de manejar la situación cuando la conexión está activa y establecida con el servidor. Proporciona las salidas correspondientes para indicar que la conexión está establecida y lista para la comunicación.

3. **Ejecución del Segmento Connection Timeout:** En caso de que se produzca un tiempo de espera durante el proceso de conexión, este segmento proporciona las salidas necesarias para manejar esta situación.

4. **Ejecución del Segmento Data Sent:** Cuando los datos se han enviado con éxito al servidor, este segmento proporciona las salidas adecuadas para indicar que los datos se han transmitido correctamente.

5. **Ejecución del Segmento Send Timeout:** Si se produce un tiempo de espera durante el proceso de activación del pulsador, este segmento proporciona las salidas correspondientes para manejar este escenario.
6. **Ejecución del Segmento New Data Received:** Cuando se reciben nuevos datos del servidor, este segmento proporciona las salidas necesarias para procesar y gestionar adecuadamente los datos recibidos.

7. **Final del Subconjunto:** En este punto concluye la ejecución del subconjunto Outputs.

4.1.1.1.2 "Socket MNG"

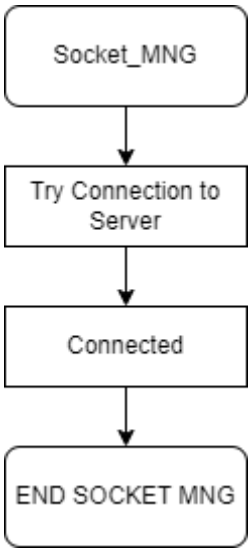


Figura 9: Estructura del subconjunto "Socket_MNG" perteneciente a la FC "Streaming_MNG".

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto.

2. **Ejecución del Segmento Try Connection to Server:** En esta fase, se ejecuta el segmento encargado de intentar establecer la conexión con el servidor. Se encarga de activar la señal de entrada inConnect de la función Socket.

3. **Ejecución del Segmento Connected:** Si la conexión se establece con éxito, se ejecuta este segmento. Se encarga de desactivar la señal de entrada inConnect de la función Socket una vez que se confirma que la conexión está activa y lista para la comunicación.

4. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto.

4.1.1.1.3 “Keep alive”

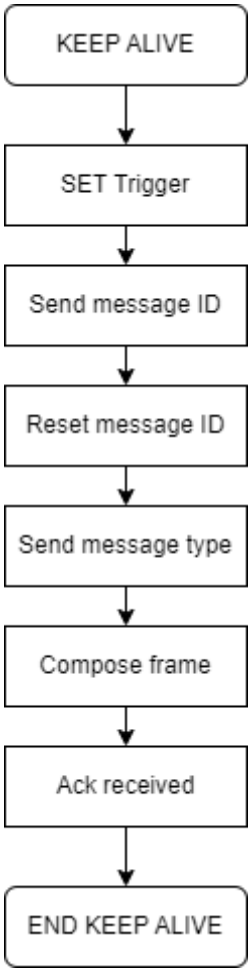


Figura 10: Estructura del subconjunto “KEEP ALIVE” perteneciente a la FC “Streaming_MNG”.

1. **Inicio del Subconjunto:** Este es el punto de partida del subconjunto Keep Alive, donde comienza la ejecución de la lógica para mantener activa la comunicación con el servidor.

2. **Ejecución del Segmento SET Trigger:** Aquí se configura el disparador para el envío periódico de mensajes de mantenimiento al servidor. Este paso es fundamental para garantizar que la conexión permanezca activa y evitar su desconexión por inactividad.

3. **Ejecución del Segmento Send Message ID:** En esta fase, se envía al servidor el identificador del mensaje, el cual es utilizado para identificar los mensajes de mantenimiento enviados desde el cliente.

4. **Ejecución del Segmento Reset Message ID:** Este segmento restablece el identificador del mensaje, asegurando que cada mensaje de mantenimiento tenga un identificador único y evitando conflictos con mensajes anteriores.
5. **Ejecución del Segmento Send Message Type:** Aquí se envía al servidor el tipo de mensaje, indicando que se trata de un mensaje de mantenimiento y no de un mensaje de datos regular.

6. **Ejecución del Segmento Compose Frame:** En esta fase se compone el marco de datos que se enviará al servidor como parte del mensaje de mantenimiento, el cual sigue una estructura de datos específica.

7. **Ejecución del Segmento Ack Received:** Este segmento maneja la recepción de un acuse de recibo (ACK) del servidor después de enviar el mensaje de mantenimiento, confirmando que el servidor ha recibido correctamente el mensaje y que la conexión sigue activa.

8. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto.

4.1.1.1.4 “Get video”

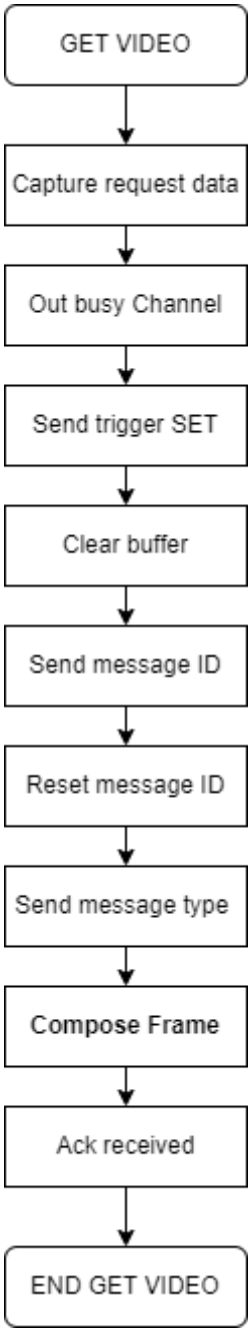


Figura 11: Estructura del subconjunto “GET VIDEO” perteneciente a la FC “Streaming_MNG”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Get Video, donde comienza la ejecución de la lógica para obtener el video desde la cámara.

2. **Ejecución del Segmento Capture Data Request:** En esta fase, se ejecuta el segmento encargado de capturar los datos necesarios para solicitar el video a la cámara.

3. **Ejecución del Segmento Out Busy Channel:** Aquí se maneja la situación cuando el canal de comunicación con la cámara está ocupado. Este segmento se encarga de gestionar esta condición y tomar las acciones necesarias.

4. **Ejecución del Segmento Send Trigger Set:** En esta fase se envía la señal para activar el disparador que inicia la captura del video en la cámara.

5. **Ejecución del Segmento Clear Buffer:** Aquí se realiza la limpieza del buffer de datos para prepararlo para recibir el video capturado.

6. **Ejecución del Segmento Send Message ID:** En esta fase, se envía al servidor el identificador del mensaje, el cual es utilizado para identificar los mensajes de mantenimiento enviados desde el cliente.

7. **Ejecución del Segmento Reset Message ID:** Este segmento restablece el identificador del mensaje, asegurando que cada mensaje de mantenimiento tenga un identificador único y evitando conflictos con mensajes anteriores.

8. **Ejecución del Segmento Send Message Type:** Aquí se envía al servidor el tipo de mensaje, indicando que se trata de un mensaje relacionado con la obtención del video.

9. **Ejecución del Segmento Compose Frame:** En esta fase se compone el marco de datos que se enviará al servidor como parte del mensaje relacionado con la obtención del video.

10. **Ejecución del Segmento Ack Received:** Este segmento maneja la recepción de un acuse de recibo (ACK) del servidor después de enviar el mensaje relacionado con la obtención del video, confirmando que el servidor ha recibido correctamente el mensaje.

11. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto.

4.1.1.1.5 “Send data”

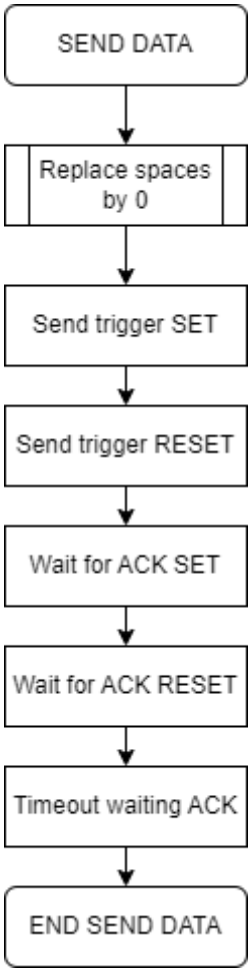


Figura 12: Estructura del subconjunto “SEND DATA” perteneciente a la FC “Streaming_MNG”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Send Data, donde comienza la ejecución de la lógica para enviar los datos al servidor.

2. **Ejecución del Subconjunto Replace Spaces by 0:** En esta fase, se ejecuta el subconjunto encargado de reemplazar los espacios por 0 en los datos a enviar. Esto puede ser necesario para garantizar la integridad de los datos antes de su transmisión.

3. **Ejecución del Segmento Send Trigger Set:** Aquí se envía la señal para activar el disparador que inicia el envío de los datos al servidor.

4. **Ejecución del Segmento Send Trigger Reset:** Este segmento se encarga de enviar la señal para restablecer el disparador después de completar el envío de los datos al servidor.

5. **Ejecución del Segmento Wait for ACK SET:** Este segmento activa la señal de espera al ACK.

6. **Ejecución del Segmento Wait for ACK RESET:** Este segmento desactiva la señal de espera al ACK si se ha recibido el ACK.

7. **Ejecución del Segmento Timeout Waiting ACK:** Si transcurre el tiempo de espera sin recibir el ACK del servidor, el segmento proporciona una señal de timeout.

8. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto.

4.1.1.1.6 “Received data”

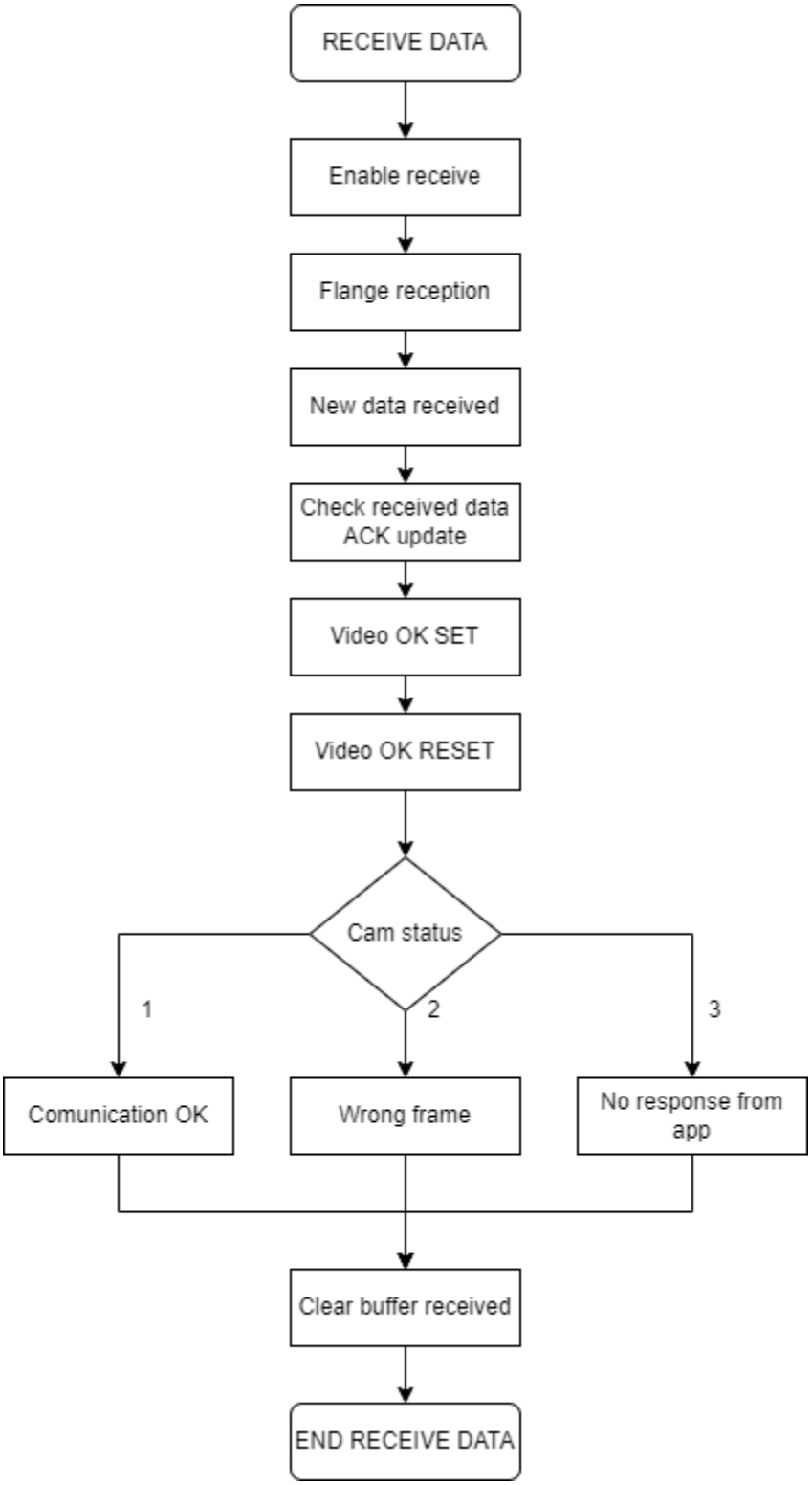


Figura 13: Estructura del subconjunto “Received Data” perteneciente a la FC “Streaming_MNG”.

1. **Inicio del Subconjunto:** Este es el punto de inicio del subconjunto Received Data, donde comienza la ejecución de la lógica para recibir y procesar los datos del servidor.

2. **Ejecución del Segmento Enable Receive:** En esta fase, se activa la recepción de datos desde el servidor para que el PLC esté preparado para recibir la información entrante.

3. **Ejecución del Segmento Flange Reception:** Aquí se maneja la recepción de datos para que sea única por cada ciclo del PLC.

4. **Ejecución del Segmento New Data Received:** En esta fase se maneja la recepción de nuevos datos desde el servidor. Este segmento se activa cada vez que se reciben datos nuevos y se encarga de procesarlos.

5. **Ejecución del Segmento Check Received Data ACK Update:** Este segmento se encarga de verificar si se ha recibido un acuse de recibo (ACK) del servidor en respuesta a los datos enviados previamente. Si se recibe el ACK, se actualiza el estado de la recepción de datos.

6. **Ejecución del Segmento Video OK SET:** Aquí se activa la señal para indicar que el video ha sido creado correctamente.

7. **Ejecución del Segmento Video OK RESET:** Este segmento se encarga de restablecer la señal de estado del video recibido después de que se haya procesado correctamente.

8. **Ejecución del Conjunto de Segmentos Dependientes de la Variable Cam Status:** Este conjunto de segmentos se ejecuta en función del estado actual de la cámara (Cam Status). Dependiendo de este estado, se ejecutarán diferentes acciones para procesar los datos recibidos de manera adecuada.

9. **Ejecución del Segmento Clear Buffer Received:** En esta fase se realiza la limpieza del buffer de datos recibidos para prepararlo para la recepción de nuevos datos.

10. **Final del Subconjunto:** Este es el punto en el que concluye la ejecución del subconjunto.

4.2 Solución en Visual Studio

Este programa desempeña un papel crucial al actuar como el puente de comunicación entre el PLC Siemens y la cámara Dahua, integrando eficazmente el hardware y el software para garantizar un control preciso y una supervisión efectiva de todo el proceso.

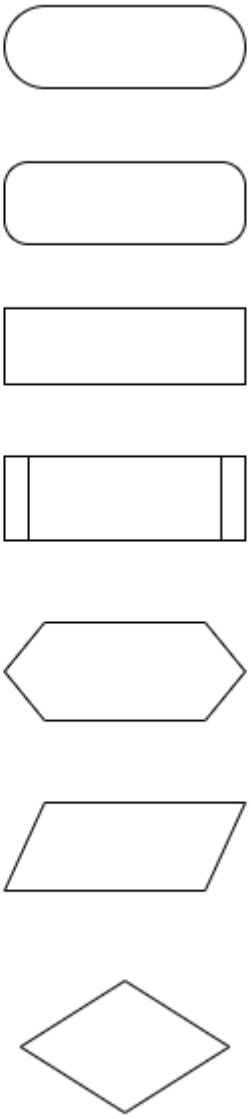
El servidor, en este contexto, desempeñará un papel fundamental al recibir las tramas enviadas por el cliente (PLC) a través del *socket*, interpretarlas y responder de manera adecuada. La cámara, actuando como servidor, procesará las tramas recibidas y ejecutará las acciones correspondientes, como iniciar o detener la grabación de video, ajustar la hora de la cámara, entre otras.

La necesidad de utilizar un entorno como Visual Studio se justifica por la complejidad de la comunicación y la interacción requerida entre el PLC Siemens y la cámara mediante tramas codificadas. Visual Studio proporciona un entorno de desarrollo integrado (IDE) que ofrece las herramientas necesarias para crear aplicaciones robustas y sofisticadas, permitiendo así la implementación eficiente de la lógica de comunicación y control entre los dispositivos.

Al aprovechar las capacidades de Visual Studio, se puede implementar fácilmente la lógica necesaria para enviar y recibir tramas codificadas entre el PLC y la cámara, así como el uso de comandos **"***web service***",** facilitando así la integración de la funcionalidad de control y supervisión en el sistema global.

El código implementado en Visual Studio para la comunicación entre el PLC Siemens y la cámara se organiza en una serie de funciones y subrutinas que se ejecutan secuencialmente para lograr la interacción deseada.

Los bloques que se verán a continuación en los diagramas de flujo corresponden a lo siguiente:



-Inicio y final del bucle principal.

-Inicio y final de funciones, regiones, clases y subconjuntos.

-Ejecución de una o más líneas de código

-Llamada de función o subconjuntos de código.

-Llamada de región de métodos.

-Llamada de clase.

-Condición necesaria para la ejecución del código.

Figura 14: Leyenda de bloques de diagrama de flujo.

A continuación, se presenta el flujo de ejecución del programa completo:

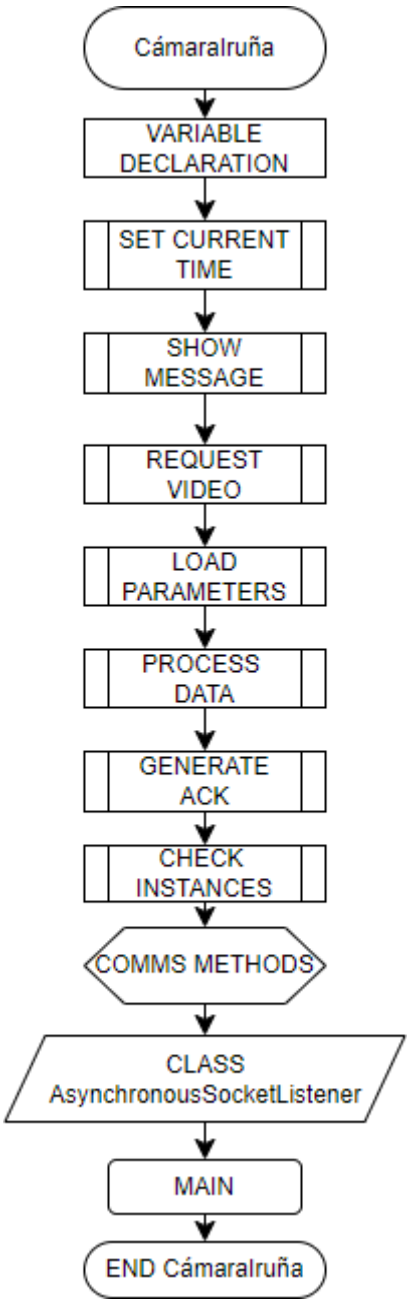


Figura 15: Estructura general del programa implementado en Visual Studio.

Este diagrama representa el flujo de ejecución del programa completo "CamaraIruña". Comienza con la inicialización del programa y la declaración de las variables necesarias. Luego, se crean una serie de funciones que cumplen diferentes propósitos, como establecer la hora actual, mostrar mensajes, solicitar video, cargar parámetros, procesar datos y generar respuestas de confirmación. Posteriormente, se incluyen regiones específicas de código para el manejo de temporizadores y métodos de comunicación. Finalmente, el programa entra en un bucle principal infinito donde se ejecutan las funciones necesarias y se mantiene la comunicación con los dispositivos relevantes.

4.2.1 Función SET CURRENT TIME

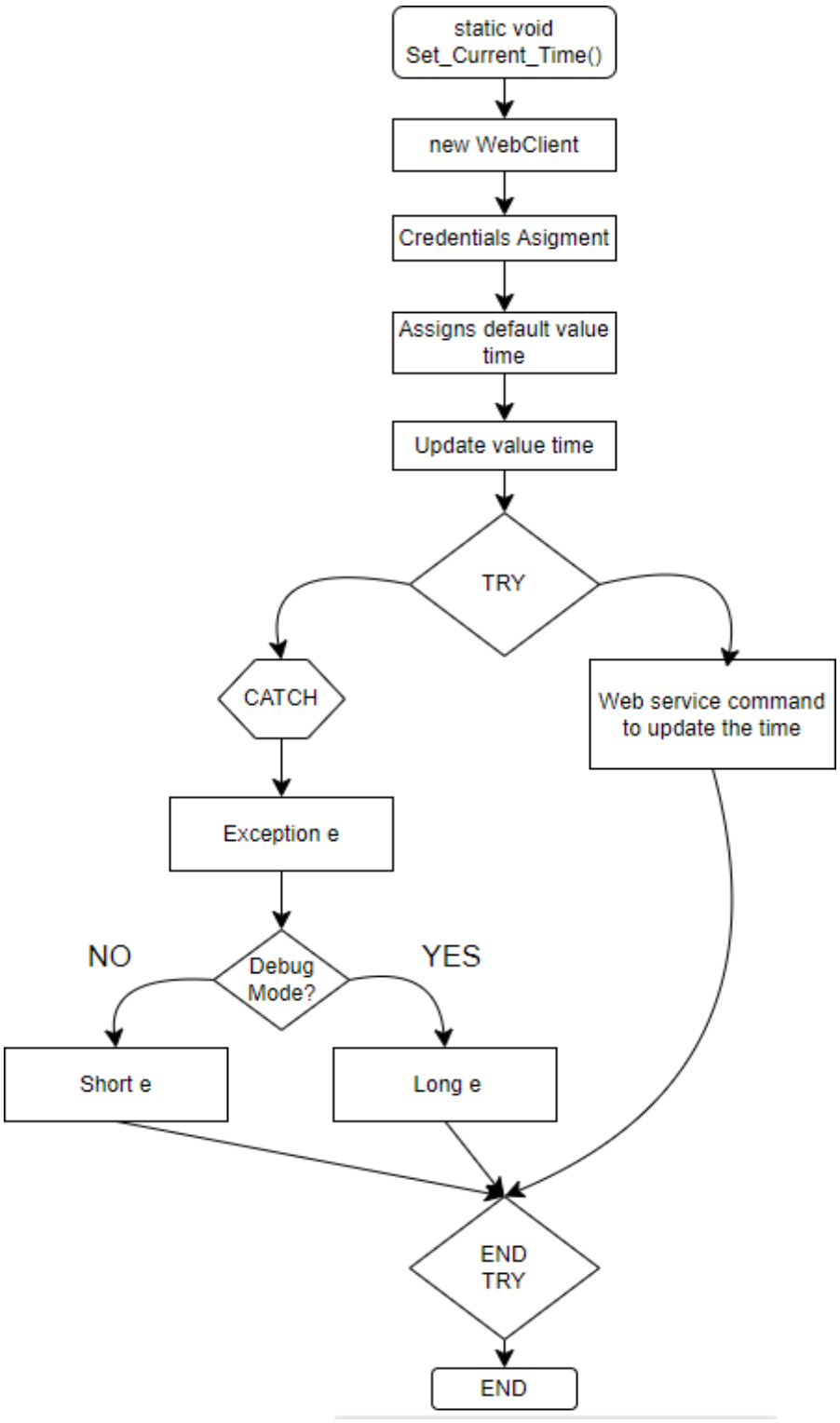


Figura 16: Estructura de la función estática "Set_Current_Time".

Es una función estática (no es necesario crear una instancia de la clase para llamarla) y no devuelve ningún valor. En ella se crea una nueva instancia de la clase "**WebClient**", que permite

la descarga o carga de contenido desde o hacia un recurso identificado por URI (Uniform Resource Identifier).

En el contexto de la programación web y de redes, una URI se utiliza para identificar de manera única un recurso en internet. Puede ser una dirección web, una ubicación de un archivo, una dirección de correo electrónico, entre otros.

Por ejemplo, en el código proporcionado, la URI se utiliza para especificar la dirección a la que se enviará la solicitud HTTP para establecer la hora actual en el dispositivo remoto. La URI proporciona la ubicación del recurso al que se accederá mediante la solicitud HTTP. En este caso, la URI incluye la dirección IP del dispositivo remoto y una ruta específica que apunta al recurso que maneja la configuración de la hora del dispositivo.

Se establecen las credenciales para la autenticación HTTP básica. En este caso, se proporciona un nombre de usuario "admin" y una contraseña "gh1001gh".

Se crea una nueva instancia de la clase "**DateTime**" llamada "**dt**" para almacenar la fecha y la hora actual, posteriormente se le asigna a esa variable la fecha y la hora actual del sistema.

Se inicia un bloque **"try"** para manejar cualquier excepción que pueda ocurrir durante la ejecución del código dentro de este bloque.

Se envía una solicitud HTTP POST al dispositivo remoto utilizando el método **"UploadString"** del objeto **"CurrentTime"**. La URL especificada incluye los parámetros necesarios para establecer la hora actual en el dispositivo remoto, utilizando la fecha y hora almacenadas en la variable **"dt"**. Si ocurre alguna excepción durante la ejecución del bloque **"try"**, se captura y se maneja en el bloque **"catch"**. En este caso, se captura cualquier excepción de tipo **"Exception"** y se almacena en la variable **"e"**.

Se evalúa la variable **"debugMode"** para determinar si se debe imprimir información de depuración (**DEBUG**) o mensajes de error (**ERROR**). La variable **"debugMode"** parece ser una variable booleana que controla si se activa o desactiva el modo de depuración.

Esta función se utiliza para establecer la hora actual en un dispositivo remoto mediante una solicitud HTTP. Si ocurre algún error durante el proceso, se maneja adecuadamente y se informa al usuario mediante mensajes de depuración o mensajes de error, dependiendo del modo de operación configurado.

4.2.2 Función SHOW MESSAGE

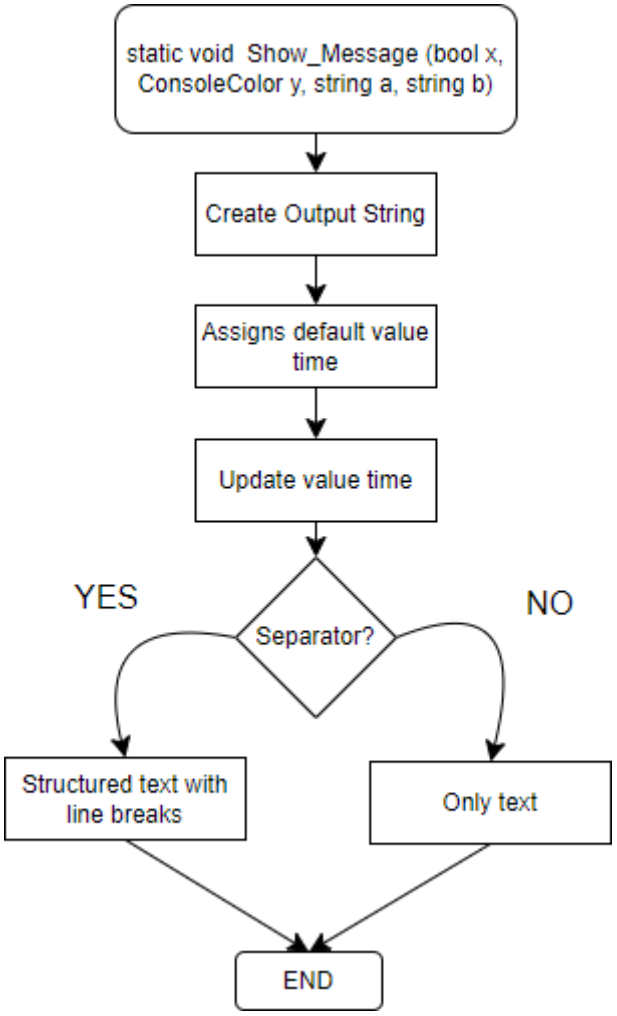


Figura 17: Estructura de la función estática "Show_Message".

Esta es la declaración de la función **"Msg".** Es una función estática que no devuelve ningún valor (**void**). Toma cuatro parámetros: **"separator"** (un booleano que indica si se debe imprimir un separador adicional), **"color"** (el color del mensaje en la consola), **"message1"** y **"message2"** (los mensajes a imprimir).

Se crea una nueva instancia de la clase **"DateTime"** llamada **"dt"** para almacenar la fecha y hora actual. Se asigna a la variable **"dt"** la fecha y hora actual del sistema.

Se evalúa la variable **"separator"** para determinar si se debe imprimir un separador adicional. Si **"separator"** es verdadero, se imprime un mensaje con un separador adicional en la consola; de lo contrario, se imprime solo el mensaje.

En resumen, esta función **"Msg"** se utiliza para imprimir mensajes en la consola con un formato específico que incluye la fecha y hora actuales. Dependiendo del valor de **"separator"**, puede imprimir mensajes con o sin un separador adicional para distinguir entre ellos. El color del mensaje en la consola también se puede especificar mediante el parámetro **"color"**.

4.2.3 Función REQUEST VIDEO

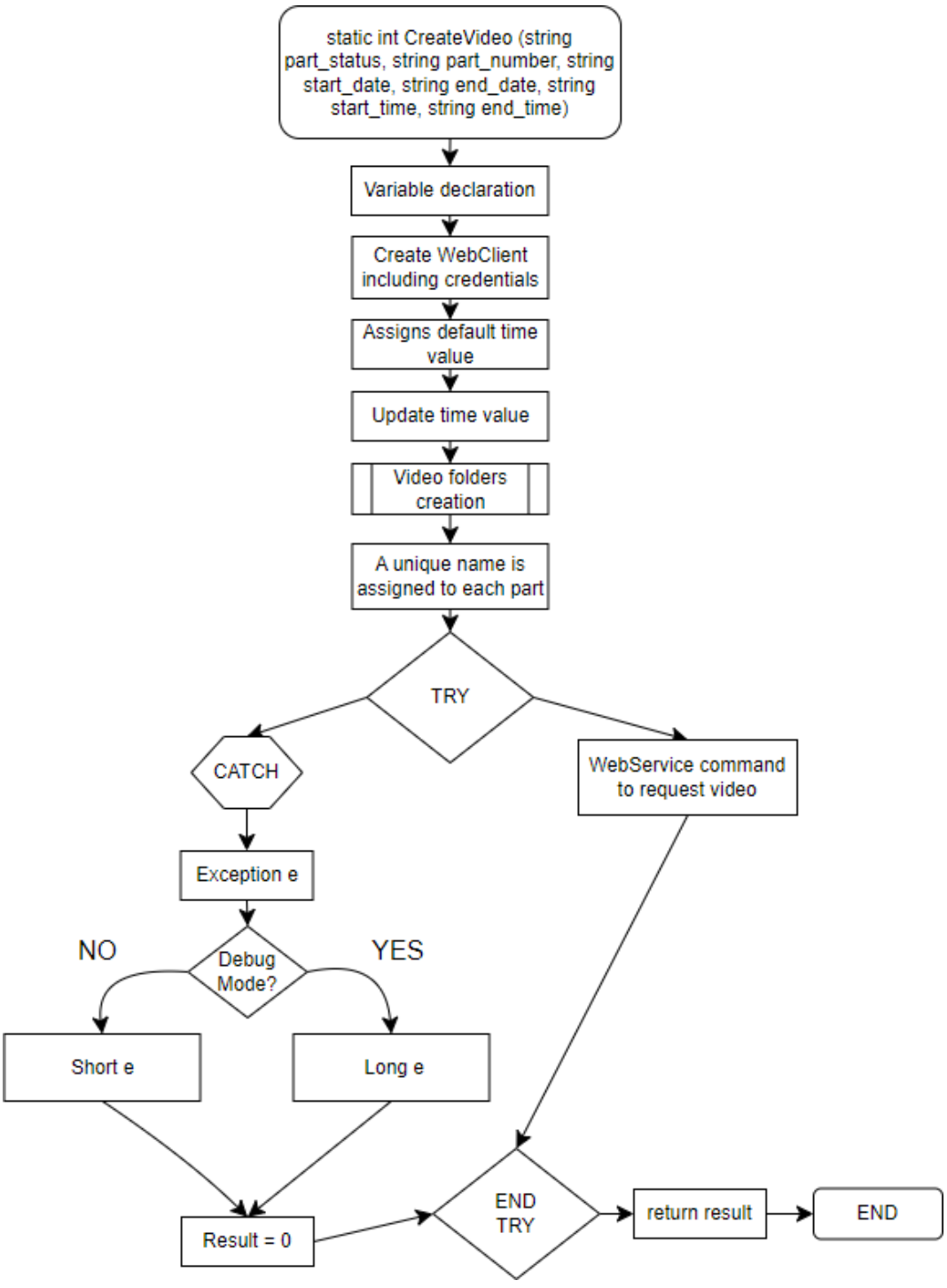


Figura 18: Estructura de la función estática "CreateVideo".

Esta es la declaración de la función **"createVideo"**. Es una función estática que devuelve un valor entero (**int**) como resultado. Toma seis parámetros de entrada que representan el estado de la pieza, el número de pieza, la fecha y hora de inicio y fin de la grabación del video.

Se inicializa la variable **"result"** con el valor 0. Esta variable se utiliza para indicar el resultado de la operación. También se declaran las variables **"filename"**, **"path"** y **"pathNfilename"** para almacenar el nombre del archivo de video, la ruta de almacenamiento y la ruta completa del archivo de video, junto con un arreglo de bytes **"byteResponse"** para almacenar la respuesta de la solicitud HTTP.

Se crea una nueva instancia de la clase **"WebClient"** llamada **"part"** para manejar la comunicación web y se le asignan las credenciales correspondientes y una nueva instancia de la clase **"DateTime"** llamada **"dt"** para almacenar la fecha y hora actual.

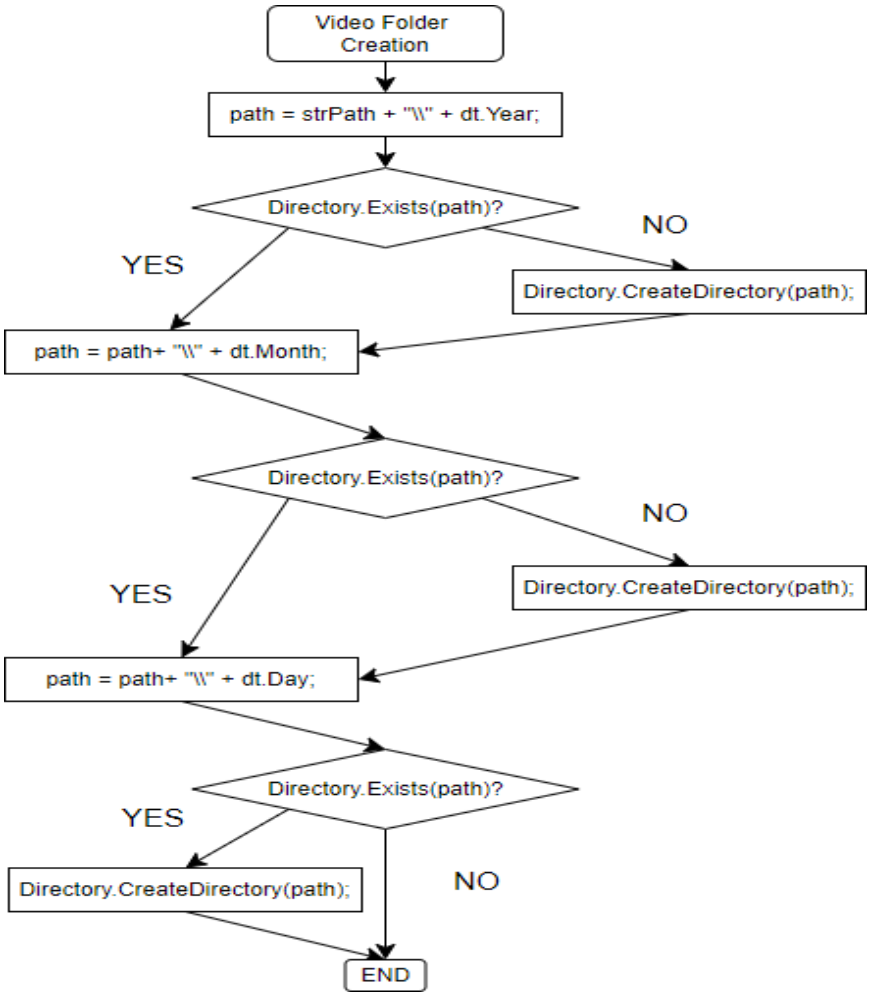


Figura 19: Estructura del subconjunto de código "Video Folder Creation" perteneciente a la función estática "CreateVideo".

Se crean las carpetas donde se almacenarán los videos, si aún no existen, utilizando la fecha actual como parte de la ruta de almacenamiento y se asigna un nombre único al archivo de video basado en el número de parte, el estado de la parte y la fecha y hora actuales.

Por último, se realiza una solicitud HTTP para descargar el archivo de video del servidor remoto utilizando el método **"DownloadFile"** del objeto **"part"**. La URL especificada incluye los parámetros necesarios para iniciar la descarga del video. Si la descarga se realiza correctamente, se establece **"result"** en 1, si ocurre algún error durante el proceso, se captura y se maneja adecuadamente en el bloque **"catch"**, mostrando un mensaje de error en la consola y estableciendo **"result"** en 0. Finalmente, la función devuelve el valor de **"result"** como resultado de la función para indicar si la operación fue exitosa (1) o no (0).

4.2.4 Función LOAD PARAMETERS

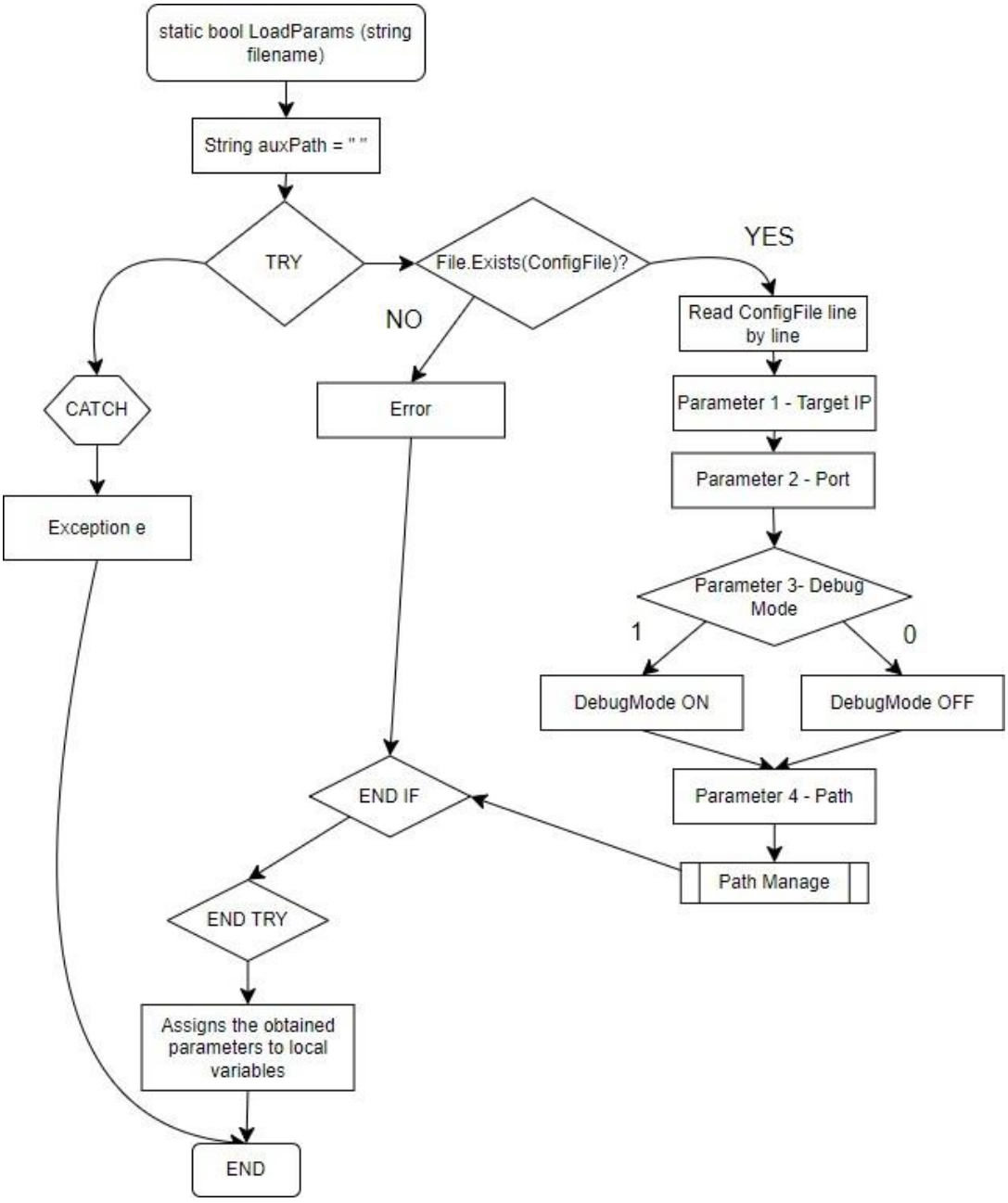


Figura 20: Estructura de la función estática "LoadParams".

Esta es la declaración de la función **"loadParams"**. Es una función estática que devuelve un valor booleano (**bool**) como resultado. Toma un parámetro de entrada que representa el nombre del archivo de configuración.

Se inicializa la variable **"auxPath"** como una cadena vacía. Esta variable se utilizará para construir la ruta de acceso a la carpeta de destino. En el bloque **"try"**, se intenta cargar los parámetros de configuración desde el archivo especificado.

Se verifica si el archivo de configuración especificado existe en la ruta especificada (**“configFile”**). Si el archivo existe, se leen los parámetros de configuración desde el archivo y se almacenan en un arreglo llamado **“parameter”**.

Se asignan los valores de los parámetros a las variables correspondientes (**“strLocalIP”**, **“strLocalPort”**, **“debugMode”** y **“strPath”**). Estos valores se obtienen del arreglo **“parameter”** utilizando índices específicos.

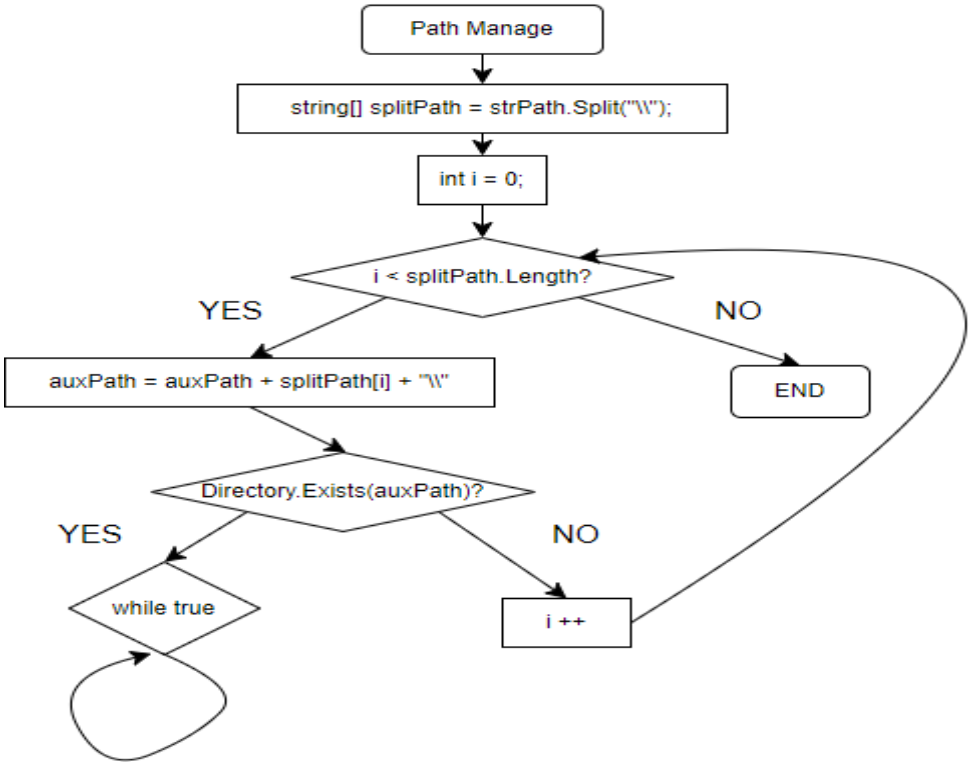


Figura 21: Estructura del subconjunto de código “Path Manage” perteneciente a la función estática “LoadParams”.

Se verifica y se crea la ruta de acceso especificada en el archivo de configuración, si aún no existe.

Si ocurre alguna excepción durante el proceso, se captura y se maneja adecuadamente en el bloque **“catch”**. Se muestra un mensaje de error en la consola y se devuelve **“false”** para indicar que ha ocurrido un error durante la carga de los parámetros.

Después de cargar correctamente los parámetros, se convierten y asignan valores a las variables **“localPort”** y **“localIP”**, y se actualiza la variable **“strPath”** con la ruta de acceso completa. Luego, se devuelve **“true”** para indicar que la carga de parámetros ha sido exitosa.

Esta función **“loadParams”** se utiliza para cargar los parámetros de configuración desde un archivo de texto, manejar cualquier excepción que pueda ocurrir durante el proceso y actualizar las variables correspondientes con los valores cargados.

4.2.5 Función PROCESS DATA

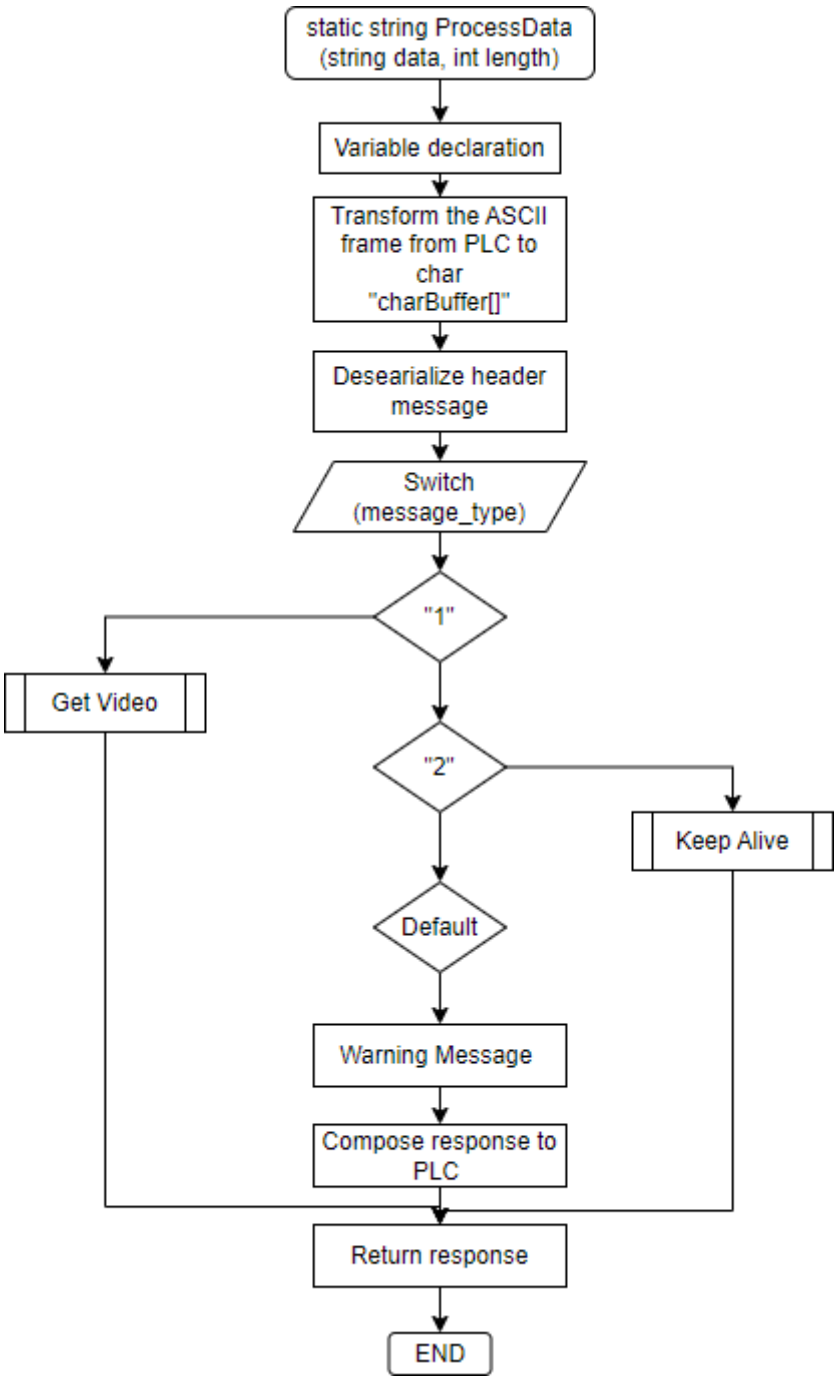


Figura 22: Estructura de la función estática "ProcessData".

Esta es la declaración de la función **"processData"**. Es una función estática que devuelve una cadena (**"string"**) como resultado. Toma dos parámetros de entrada: **"data"**, que representa los datos recibidos desde el PLC, y **"length"**, que indica la longitud de los datos recibidos.

Se declaran e inicializan algunas variables locales necesarias para el procesamiento de los datos.

Se convierte la cadena de datos recibida (**"data"**) en un arreglo de bytes (**"byteBuffer"**) utilizando la codificación ASCII.

Se extraen y deserializan los diferentes campos del mensaje recibido, como el carácter de inicio (**"STX"**), el identificador de mensaje (**"msg_ID"**) y el tipo de mensaje (**"message_type"**).

Dependiendo del tipo de mensaje recibido (**"message_type"**), se ejecuta un bloque de código específico. En este caso, hay dos tipos de mensajes: "1" (get Video) y "2" (Keep alive).

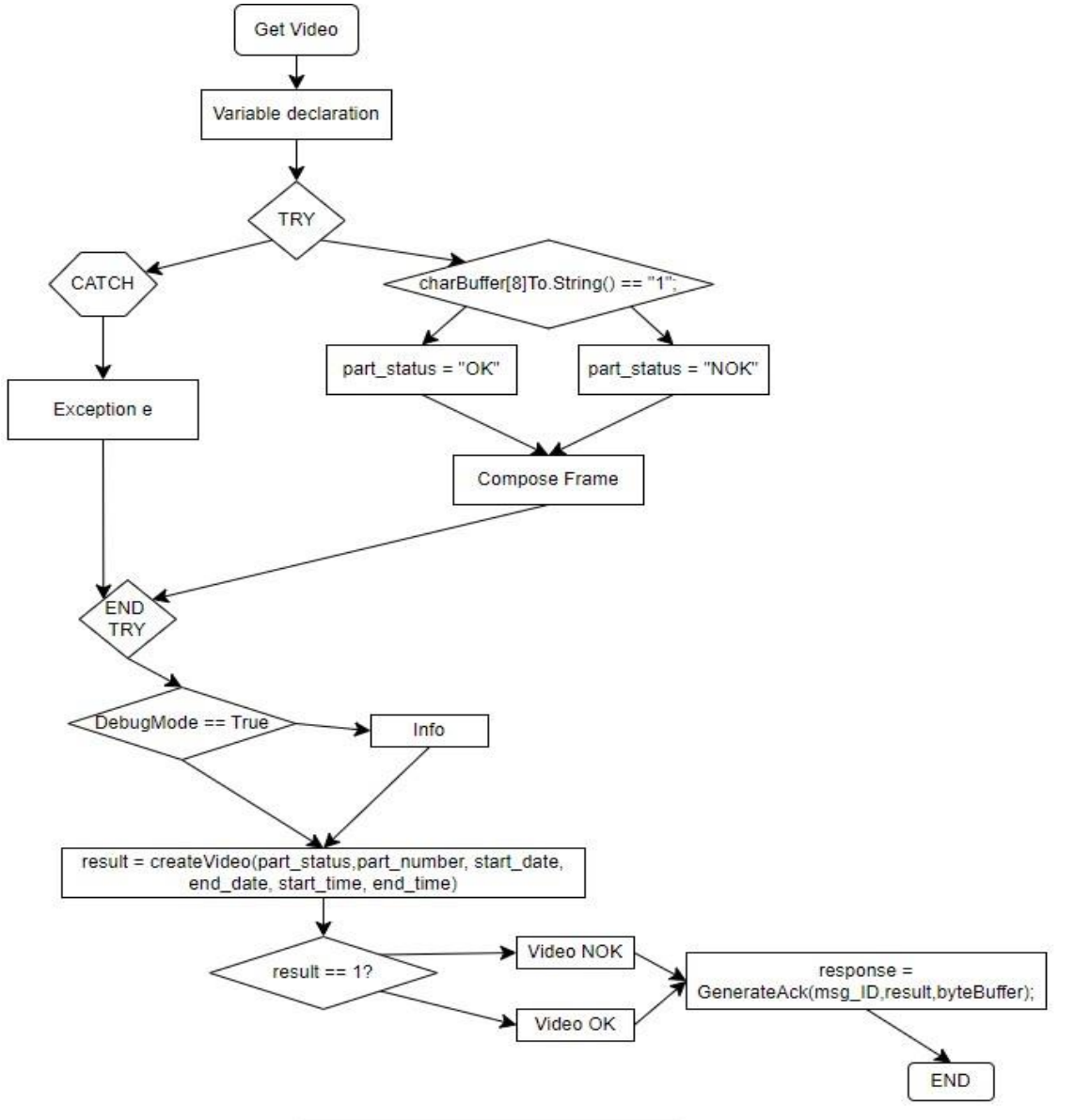


Figura 23: Estructura del subconjunto de código "Get Video" perteneciente a la función estática "ProcessData".

En el caso de un mensaje de tipo "1" (get Video), se extraen y procesan los diferentes parámetros del mensaje, como el estado de la parte (**"part_status"**), el número de parte (**"part_number"**), la fecha de inicio (**"start_date"**), la hora de inicio (**"start_time"**), la fecha de finalización (**"end_date"**) y la hora de finalización (**"end_time"**). Luego se llama a la función **"createVideo"** para crear un video y se genera una respuesta de ACK al PLC.

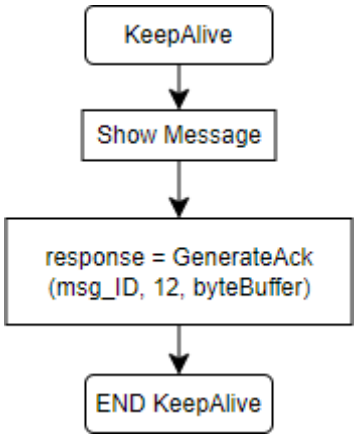


Figura 24: Estructura del subconjunto de código "Keep Alive" perteneciente a la función estática "ProcessData".

En el caso de un mensaje de tipo "2" (Keep alive), se muestra un mensaje informativo en la consola y se genera una respuesta de ACK al PLC.

Si el tipo de mensaje no coincide con ninguno de los casos anteriores, se muestra un mensaje de error en la consola y se genera una respuesta de ACK al PLC.

Esta función se encarga de procesar los datos recibidos desde el PLC, interpretar los diferentes tipos de mensajes, realizar las acciones correspondientes y generar una respuesta adecuada para enviar de vuelta al PLC.

4.2.6 Función GENERATE ACK

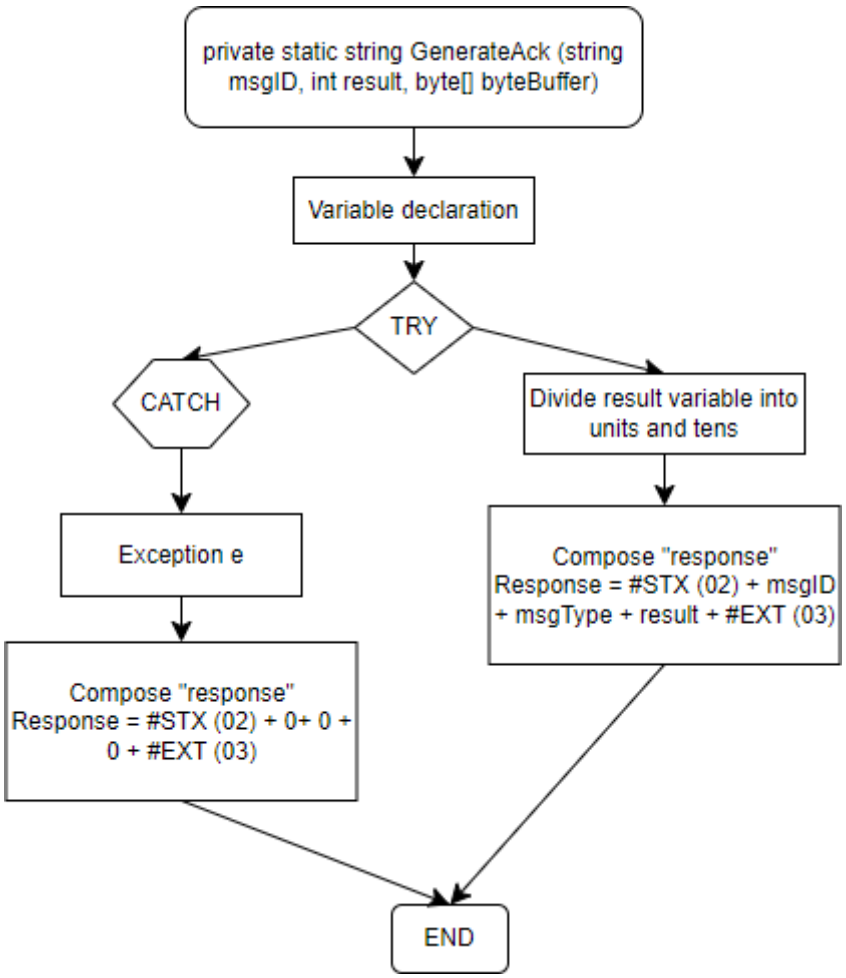


Figura 25: Estructura de la función estática "GenerateAck".

Esta es la declaración de la función **"GenerateAck"**. Es una función privada y estática que devuelve una cadena (**"string"**) como resultado. Toma tres parámetros de entrada: **"msgID"**, que representa el identificador del mensaje recibido, **"result"**, que indica el resultado del procesamiento del mensaje, y **"byteBuffer"**, que contiene los bytes del mensaje original recibido.

Se realiza una serie de operaciones para construir la respuesta de ACK. Esto incluye la conversión del resultado (**"result"**) en un arreglo de bytes (**"test"**) y la creación de un arreglo de bytes (**"responseByteBuffer"**) que contendrá la respuesta de ACK.

Se asignan valores específicos a cada posición del arreglo **"responseByteBuffer"** de acuerdo con el formato esperado para la respuesta de ACK. Esto incluye el carácter de

inicio (**#STX**), el identificador de mensaje (**"msgID"**), el tipo de mensaje, el resultado (**"result"**) y el carácter de finalización (**#ETX**).

La respuesta de ACK se convierte de bytes a una cadena (**"string"**) utilizando la codificación predeterminada y se devuelve como resultado de la función.

Se manejan posibles excepciones que puedan ocurrir durante la generación de la respuesta de ACK. En caso de que se produzca una excepción, se genera una respuesta de ACK predeterminada y se muestra un mensaje de error en la consola.

Esta función se encarga de construir una respuesta de ACK basada en el resultado del procesamiento del mensaje recibido y devuelve esta respuesta como una cadena para ser enviada de vuelta al PLC.

4.2.7 Función CHECK INSTANCES

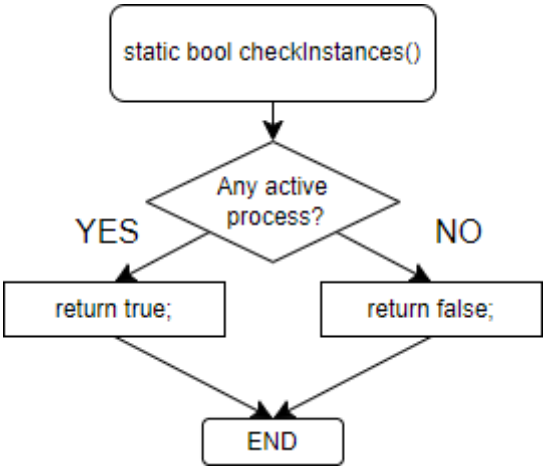


Figura 26: Estructura de la función estática "CheckInstances".

Esta es la declaración de la función **"checkInstances"**. Es una función privada y estática que devuelve un valor booleano (**"true"** o **"false"**). No toma ningún parámetro de entrada.

Se obtiene una referencia al proceso actual y se devuelve el nombre del proceso actual.

Una vez obtenido el nombre, la función busca todos los procesos en ejecución con el mismo nombre que el proceso actual. También devuelve el número de procesos encontrados con ese nombre.

Se compara la cantidad de procesos encontrados con 1 para determinar si ya hay una instancia del programa en ejecución.

La función devuelve **"true"** si se encuentra más de una instancia, de lo contrario **"false"**.

4.2.8 Región COMMS METHODS

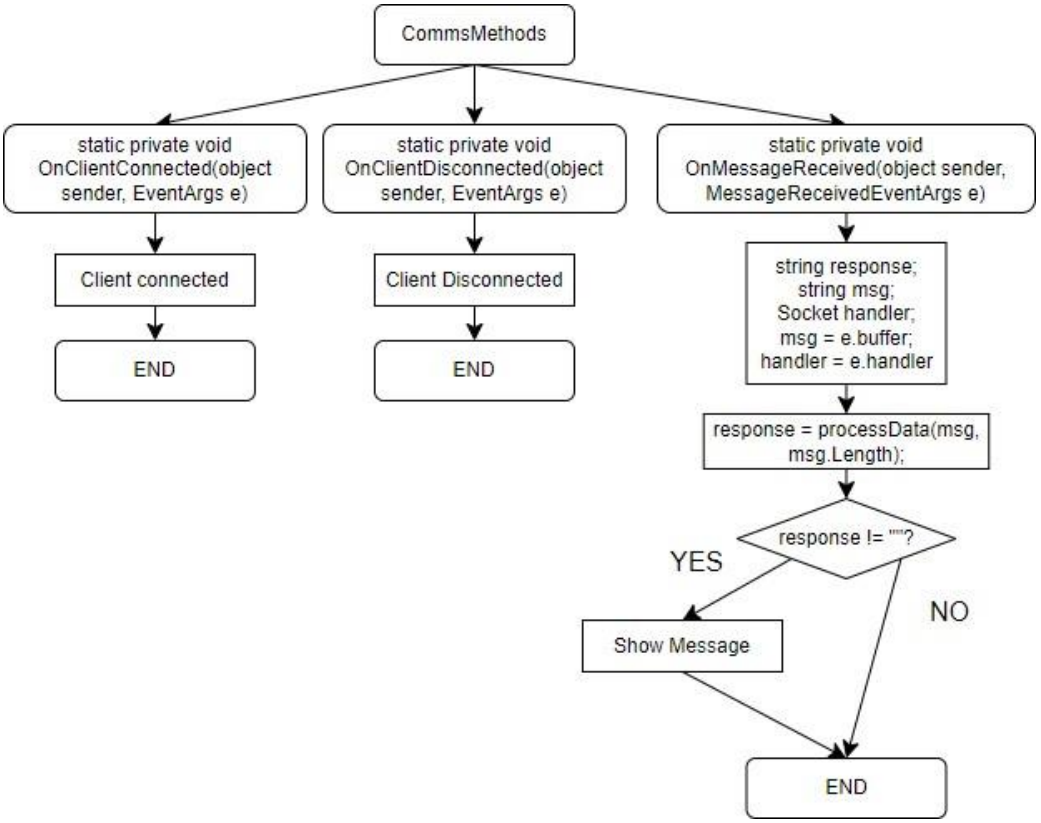


Figura 27: Estructura de la región de los métodos de comunicación "CommsMethods".

Este bloque de código contiene métodos que se utilizan para manejar eventos relacionados con la comunicación del servidor.

"OnClientConnected": Este método se llama cuando un cliente se conecta al servidor. Muestra un mensaje indicando que un cliente se ha conectado.

"OnClientDisconnected": Este método se llama cuando un cliente se desconecta del servidor. Muestra un mensaje indicando que un cliente se ha desconectado.

"OnMessageReceived": Este método se llama cuando se recibe un nuevo mensaje del cliente. Obtiene el mensaje recibido y el controlador del socket asociado. Llama a la función **"processData"** para procesar el mensaje y obtener una respuesta. Luego, envía la respuesta de vuelta al cliente.

4.2.9 Class AsynchronousSocketListener

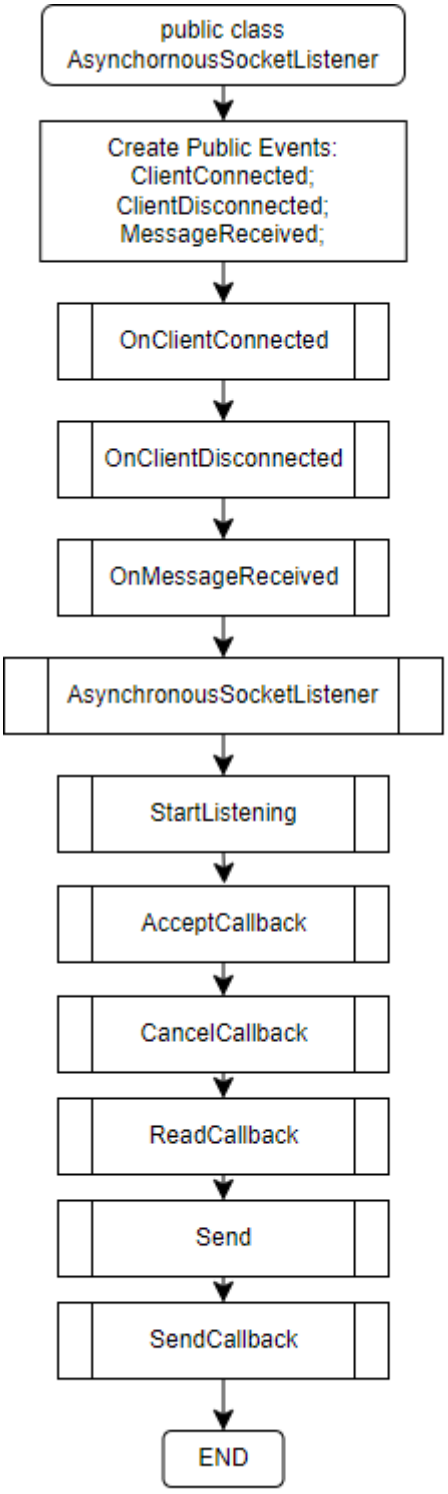


Figura 28: Estructura de la clase "AsynchronousSocketListener".

Este código implementa una clase llamada **"AsynchronousSocketListener"** que actúa como un servidor TCP asíncrono.

La clase define tres eventos públicos: **"ClientConnected"**, **"ClientDisconnected"** y **"MessageReceived"**. Estos eventos se desencadenan cuando un cliente se conecta al servidor, se desconecta del servidor o se recibe un mensaje del cliente, respectivamente.

La clase proporciona el método **"StartListening"**, que inicia la escucha del servidor en una dirección IP y un puerto específico. Este método crea un socket TCP y comienza a escuchar conexiones entrantes en el puerto especificado.

La clase define métodos de **"callback"** para manejar eventos de aceptación de conexiones (**"AcceptCallback"**) y recepción de datos (**"ReadCallback"**). Estos métodos se llaman de forma asincrónica cuando se reciben conexiones de clientes o datos del cliente, respectivamente.

La clase proporciona un método **"Send"** para enviar datos al cliente conectado. Este método convierte los datos de cadena en bytes utilizando la codificación ASCII y los envía al cliente.

Los métodos **"OnClientConnected"**, **"OnClientDisconnected"** y **"OnMessageReceived"** se utilizan para desencadenar los eventos correspondientes cuando se produce una conexión, desconexión o recepción de mensajes, respectivamente.

En resumen, esta clase implementa un servidor TCP asíncrono que puede aceptar conexiones de clientes, recibir datos de los clientes y enviar datos a los clientes conectados. Utiliza **"callbacks"** asincrónicos para manejar eventos de red de manera eficiente.

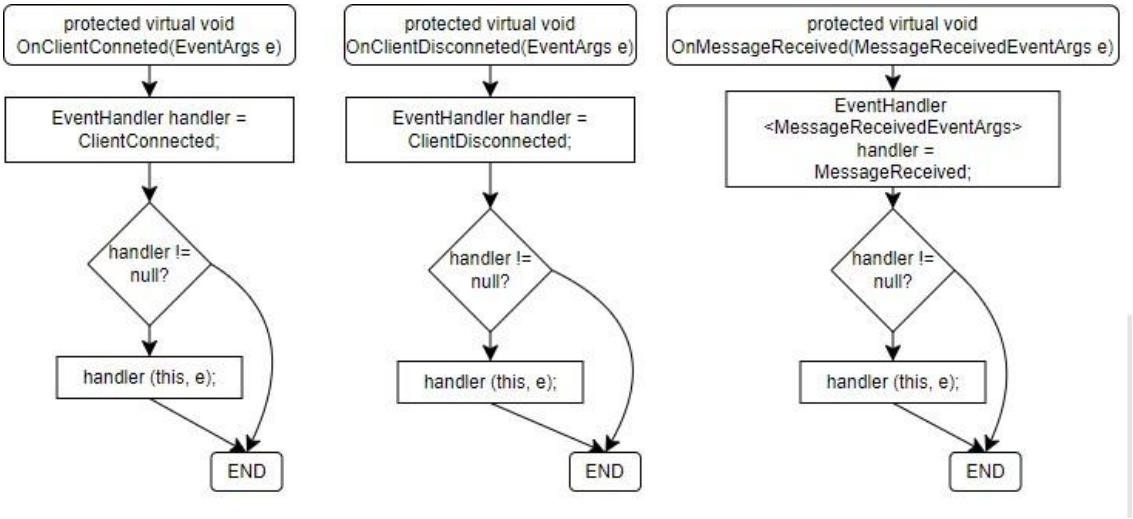


Figura 29: Estructura de los métodos "OnClientConnected" ,"OnClientDisconnected", "OnMessageReceived" de la clase "AsynchronousSocketListener".

Estos son métodos protegidos utilizados para invocar eventos cuando se produce una conexión de cliente (**"OnClientConnected"**), desconexión de cliente (**"OnClientDisconnected"**) y recepción de mensaje (**"OnMessageReceived"**).

"OnClientConnected": Este método invoca el evento **"ClientConnected"** cuando se establece una conexión de cliente. Primero, verifica si hay manejadores registrados para el evento. Si hay

manejadores registrados, invoca cada manejador con el objeto que desencadenó el evento (**"this"**) y los argumentos del evento (**"e**).

"OnClientDisconnected": Similar al método **"OnClientConnected"**, este método invoca el evento **"ClientDisconnected"** cuando se produce una desconexión de cliente. Si hay manejadores registrados para el evento, los invoca con los mismos parámetros que el método **"OnClientConnected"**.

"OnMessageReceived": Este método invoca el evento **"MessageReceived"** cuando se recibe un mensaje del cliente. Si hay manejadores registrados para el evento, los invoca con el objeto que desencadenó el evento (**"this"**) y los argumentos del evento (**"e**), que incluyen el mensaje recibido.

Permiten que otros componentes de la aplicación se suscriban y respondan a estos eventos de manera apropiada.

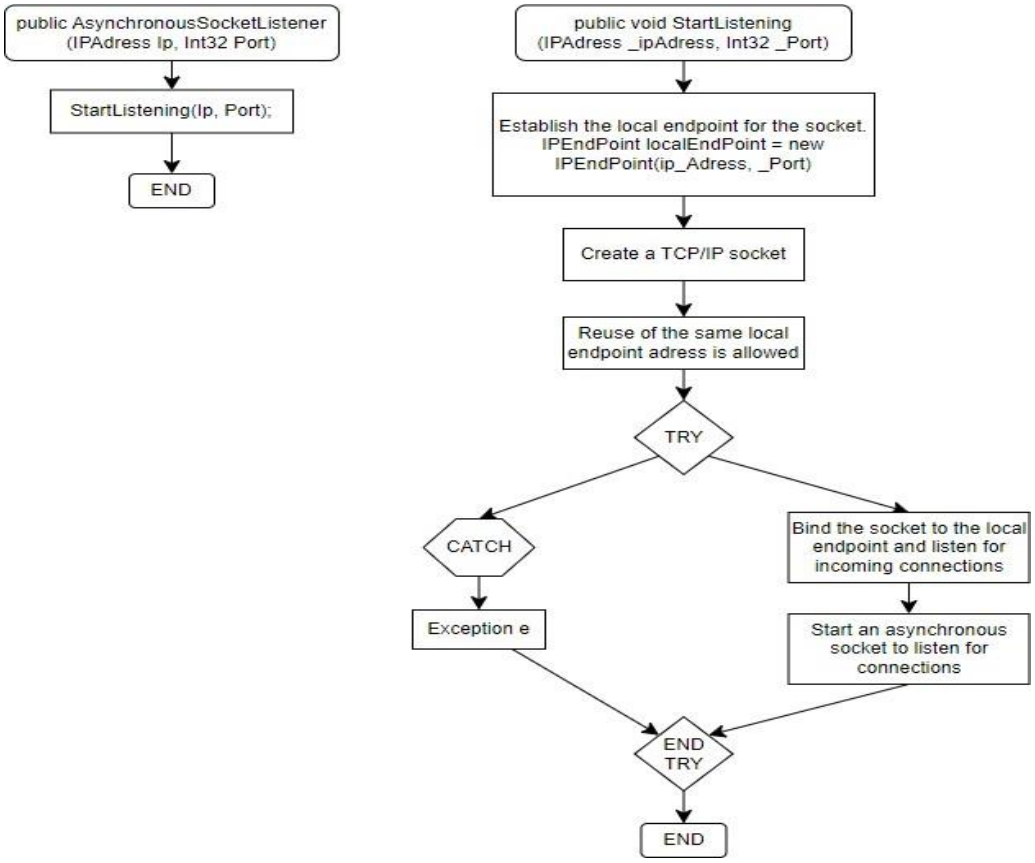


Figura 30: Estructura del constructor "AsynchronousSocketListener" y el método "StartListening" de la clase "AsynchronousSocketListener".

Este código define una clase llamada **AsynchronousSocketListener** que se encarga de escuchar las conexiones de cliente de forma asíncrona.

Constructor **"AsynchronousSocketListener (IpAdress Ip, Int32 Port)":**

- Este constructor toma como argumentos una dirección IP (**"Ip"**) y un número de puerto (**"Port"**).
- Llama al método **"StartListening"** para iniciar la escucha en el socket utilizando la dirección IP y el puerto especificado.

Método **"StartListening (IPAdress _ipAdress, Int32 _Port)":**

- Este método se encarga de configurar y comenzar a escuchar en el socket para las conexiones entrantes.
- Crea un socket TCP/IP utilizando la dirección IP y el puerto proporcionado.
- Habilita la opción **"ReuseAddress"** del socket para permitir reutilizar la misma dirección de **"endpoint"** local.
- Vincula el socket al **"endpoint"** local y establece el límite de conexiones en espera.
- Inicia la escucha para las conexiones entrantes utilizando el método **"BeginAccept"**, que es una operación asincrónica que espera una conexión entrante.
- Si ocurre algún error durante el proceso de enlace o escucha, se maneja utilizando una excepción de socket (**"SocketException"**).

Estos métodos configuran y gestionan la escucha de conexiones entrantes en el socket del servidor, permitiendo que el servidor acepte conexiones de clientes de manera asíncrona.

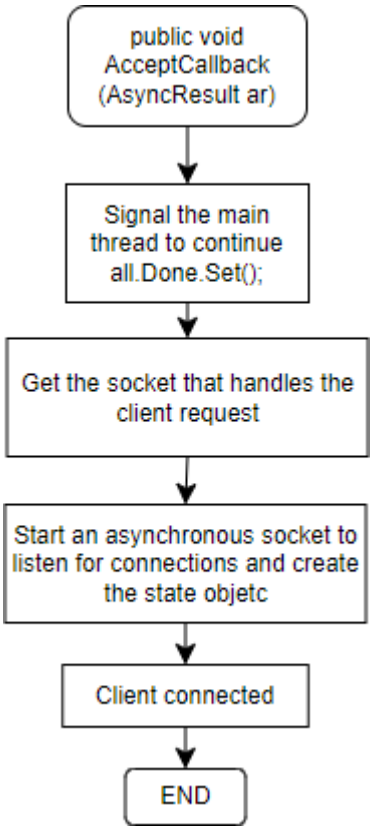


Figura 31: Estructura del método ”AcceptCallback” de la clase “AsynchronousSocketListener”.

El método “**AcceptCallback**” se ejecuta cuando se acepta una conexión entrante.
Se señala el hilo principal que continúe. Se usa junto con el “**ManualResetEvent**” para controlar cuándo se ha completado una operación asincrónica.
Recupera el socket que está escuchando las conexiones entrantes del objeto “**IAsyncResult**”.

Acepta la conexión entrante y devuelve un nuevo socket que representa la conexión establecida entre el servidor y el cliente.

Comienza a esperar la próxima conexión entrante. Esto permite que el servidor acepte múltiples conexiones de forma concurrente.

Crea un nuevo objeto “**StateObject**” para mantener el estado de la conexión.

Asigna el socket recién aceptado al campo “**workSocket**” del objeto “**StateObject**”.

Inicia una operación asincrónica para recibir datos del cliente en el socket aceptado. Cuando se reciban datos, se llamará al método “**ReadCallback**”.

Llama al evento “**ClientConnected**”, indicando que un cliente se ha conectado con éxito al servidor.

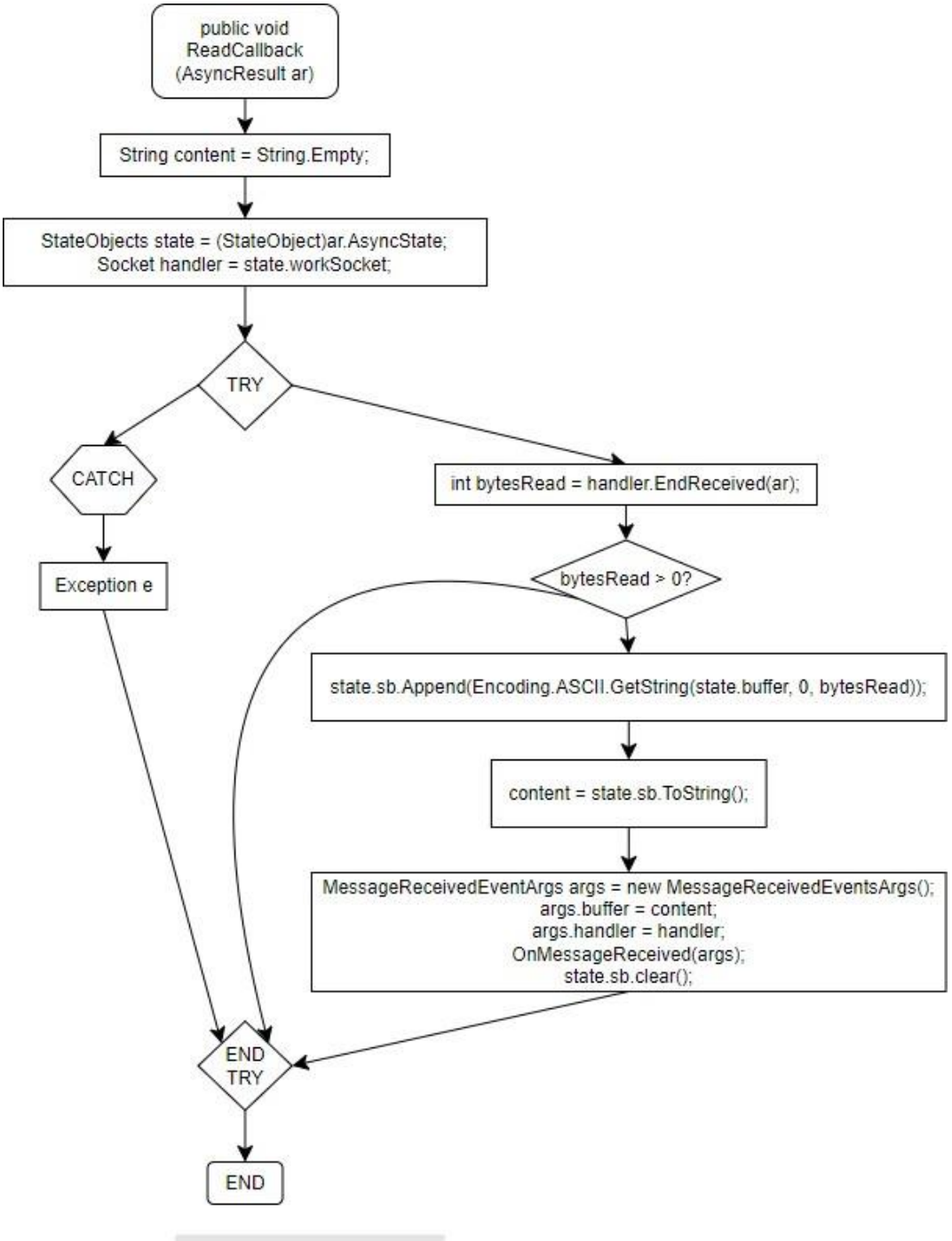


Figura 32: Estructura del método ”ReadCallback” de la clase “AsynchronousSocketListener”.

El método **“ReadCallback”** se invoca cuando los datos se han recibido completamente en el socket del cliente.

Recupera el objeto **“StateObject”** que se pasó como estado adicional cuando se inició la operación de lectura asincrónica y obtiene el socket del cliente desde el objeto **“StateObject”**.

Finaliza la operación de lectura asincrónica y devuelve el número de bytes leídos del cliente.

Verifica si se han leído algunos bytes del cliente y convierte los bytes recibidos en una cadena y los agrega al búfer de cadena del objeto **"StateObject"**.

Crea un nuevo objeto **"MessageReceivedEventArgs"** para pasar los datos recibidos y el socket del cliente al evento **"MessageReceived"**.

Asigna la cadena recibida al campo **"buffer"** del objeto **"MessageReceivedEventArgs"**.

Asigna el socket del cliente al campo **"handler"** del objeto **"MessageReceivedEventArgs".**

Llama al evento **"MessageReceived"**, pasando los datos recibidos como argumento.

Borra el contenido del búfer de cadena del objeto **"StateObject"** para prepararse para la próxima lectura de datos.

Inicia una nueva operación de lectura asincrónica para seguir recibiendo datos del cliente.

El método maneja el flujo de datos recibidos del cliente, asegurando que se lean completamente y que el servidor esté listo para recibir más datos. Si ocurre alguna excepción durante el proceso de lectura, simplemente se captura y se ignora, lo que puede ser útil para manejar situaciones excepcionales sin detener la ejecución del servidor.

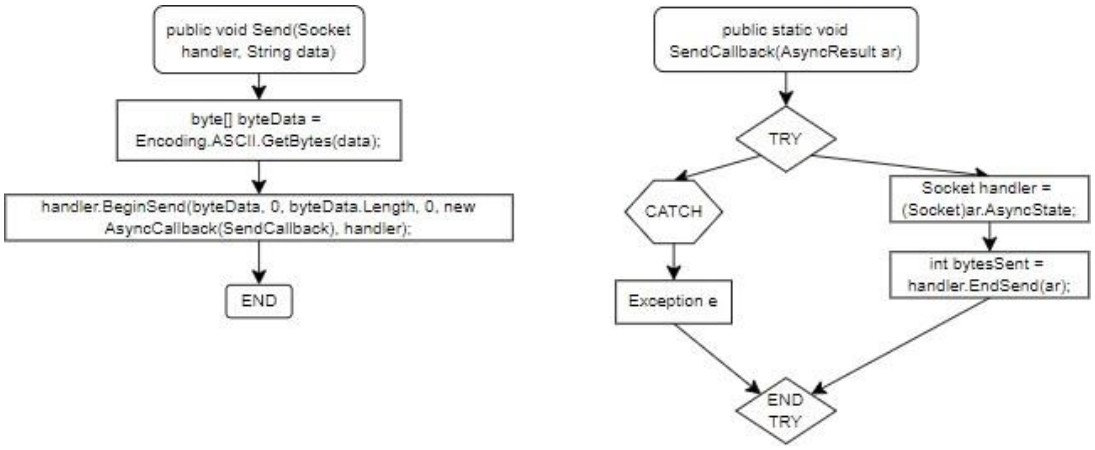


Figura 32: Estructura del método "Send" y "SendCallback" de la clase "AsynchronousSocketListener".

El método **"Send"** se utiliza para enviar datos a un dispositivo remoto a través del socket.

Convierte los datos de cadena en una matriz de bytes utilizando la codificación ASCII. Esto es necesario ya que los sockets solo pueden enviar datos en forma de bytes.

Inicia una operación de envío asincrónico en el socket **"handler"**. Toma como argumentos los datos de bytes a enviar, el índice de inicio en el búfer de datos, la longitud de los datos y algunas banderas opcionales. También se especifica el método de devolución de llamada (**"SendCallback"**) que se llamará cuando se complete la operación de envío. Además, el propio socket se pasa como estado adicional.

El método de devolución de llamada **"SendCallback"** se invoca cuando se completa la operación de envío asincrónico.

Recupera el socket del dispositivo remoto desde el objeto de estado asincrónico. Este objeto de estado es el propio socket que se pasó al iniciar la operación de envío asincrónico.

Finaliza la operación de envío asincrónico y devuelve el número de bytes que se enviaron correctamente al dispositivo remoto.

El método **"Send"** inicia una operación de envío de datos asincrónica a través del socket, mientras que el método de devolución de llamada **"SendCallback"** maneja la finalización de esa operación y cualquier excepción que pueda surgir durante el proceso de envío.

4.2.10 Bucle principal infinito MAIN

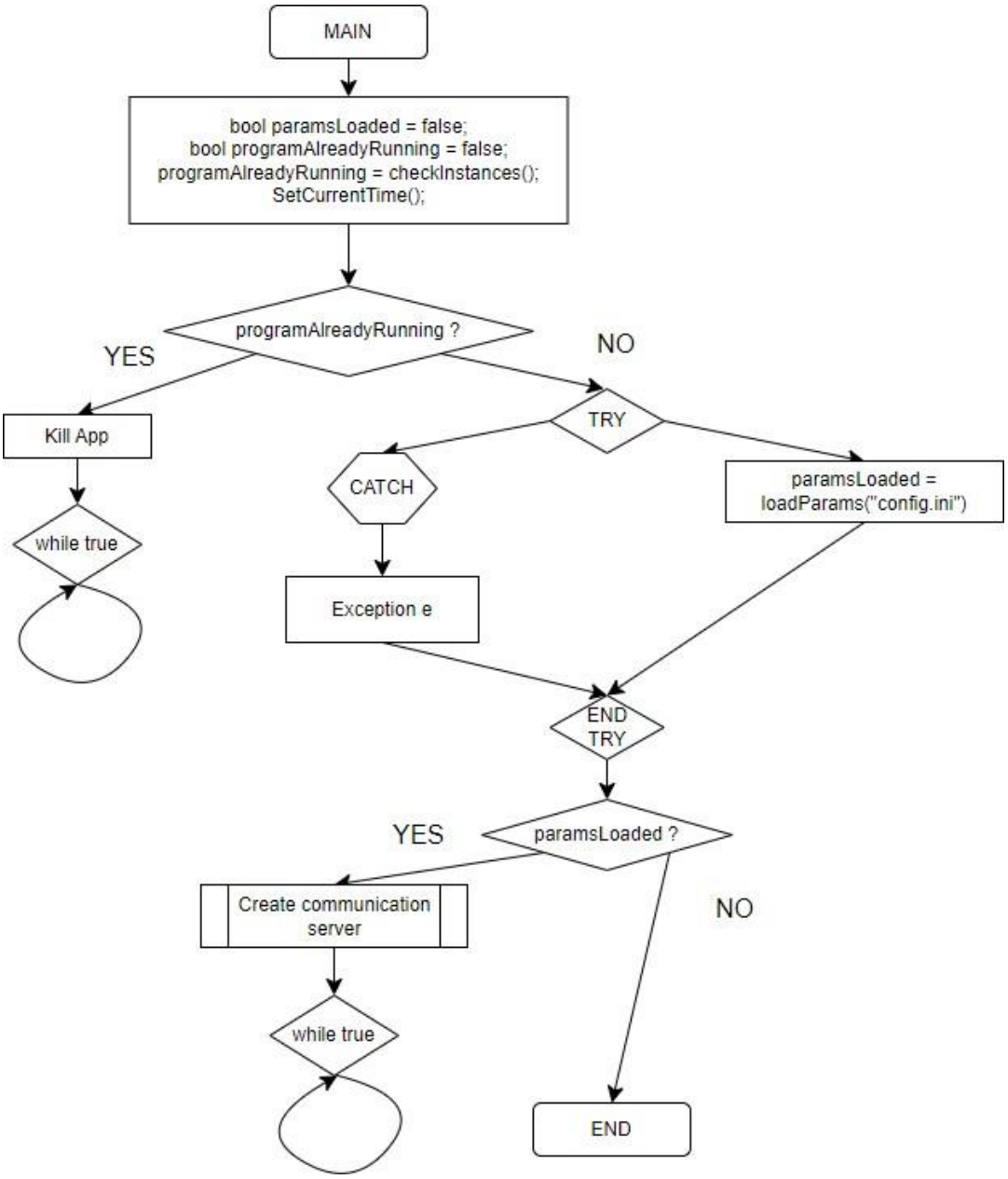


Figura 33: Estructura del método principal ”Main” del programa.

Este es el método principal **“Main”** del programa. Aquí se inicia la ejecución del programa.

Se definen dos variables locales, **“paramsLoaded”** y **“programAlreadyRunning”**, ambas inicializadas como **“false”**. Estas variables se utilizan para rastrear si los parámetros se cargaron correctamente y si el programa ya está en ejecución, respectivamente.

Se llama a la función **"checkInstances ()"** para verificar si ya hay otra instancia del programa en ejecución.

Se llama a la función **"Set_Current_Time ()"** para establecer la hora actual del dispositivo.

Si se detecta que el programa ya está en ejecución, se muestra un mensaje informativo y se establece un temporizador para cerrar la aplicación después de un período de tiempo, luego el programa entra en un bucle infinito, sin hacer nada más.

Se intenta cargar los parámetros desde el archivo "config.ini" llamando a la función **"loadParams ()"**. Si ocurre algún error durante la carga de los parámetros, se muestra un mensaje de error.

Si los parámetros se cargan correctamente, se intenta crear un servidor de comunicaciones utilizando la clase **"AsynchronousSocketListener"**. Se suscriben los eventos del servidor (**"ClientConnected"**, **"ClientDisconnected"** y **"MessageReceived"**). Si hay algún error durante la creación del servidor, se muestra un mensaje de error.

Si tanto los parámetros se cargan correctamente como el servidor se crea con éxito, el programa entra en un bucle infinito. Este bucle no realiza ninguna tarea activa, ya que el servidor de comunicaciones y otros procesos se ejecutan en hilos separados.

Si los parámetros no se cargan correctamente, se muestra un mensaje de error y el programa finaliza.

El método **"Main"** se encarga de iniciar el programa, cargar los parámetros, crear el servidor de comunicaciones y entrar en un bucle infinito para mantener el programa en ejecución, siempre y cuando no haya ocurrido ningún error crítico.

5. Manual de usuario

5.1 Propósito y audiencia

El propósito de este manual es proporcionar una guía detallada para la instalación correcta y el uso eficiente del programa desarrollado. Está diseñado específicamente para los operarios técnicos que trabajan con la máquina, con el objetivo de facilitarles el acceso a la información necesaria para configurar y operar el sistema de manera efectiva.

Este manual está dirigido a los operarios técnicos responsables de la instalación, configuración y operación del sistema desarrollado. Está destinado a aquellos con conocimientos técnicos básicos en el manejo de equipos industriales y familiaridad con el entorno de programación utilizado.

5.2 Instalación

Para la preparación de archivos se debe compilar el proyecto en Visual Studio para generar el archivo ejecutable (.exe) y asegurarse de que el archivo de configuración (.config) esté configurado correctamente con los parámetros necesarios para la ejecución del programa. Ambos deben estar en la misma carpeta, de lo contario el programa no será capaz de leer los parámetros del archivo.

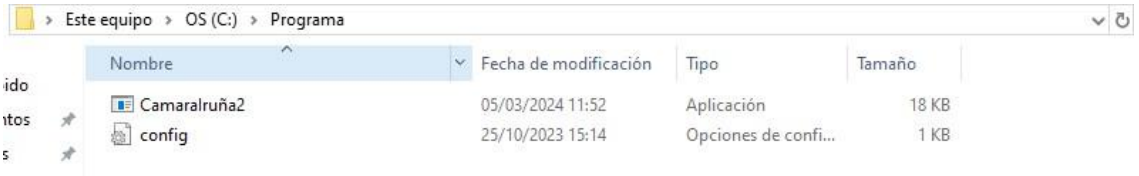


Figura 33: Ubicación de los archivos “Camaralruña2” y “config”.

```
Archivo  Edición  Formato  Ver  Ayuda
### CONFIGURATION FILE ####
[1] Param. 1 - Type the IP address of the computer where the program is running:
192.168.73.240
[2] Param. 2 - Define a port to establish the communication, advisably between these values 2000-8000.
7000
[3] Param. 3 - Debug mode
0
[4] Param. 4 - Path
C:\LOGS\VIDEOS
```

Figura 34: Estructura del archivo de configuración “config”.

Para asegurar que el programa se inicie automáticamente junto con el arranque del sistema operativo del PLC se debe hacer lo siguiente:

Acceder al sistema de archivos del PLC y presionar la tecla del logotipo de Windows + R, escribir **shell:startup** y luego aceptar. Esto abrirá la carpeta de **Inicio**, ahí se deberá dejar un

acceso directo del .exe. Esto es crucial para que el programa se ejecute a la vez que el sistema operativo y no de ningún tipo de problema. [5]

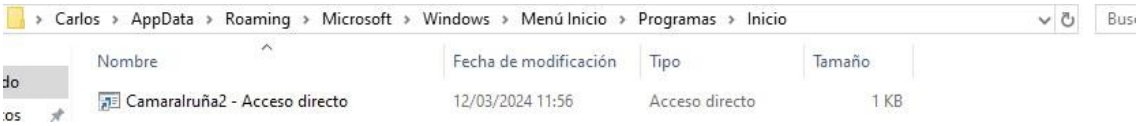


Figura 35: Ubicación del acceso directo del archivo "Camaralruña2".

Reinicie el PLC para que los cambios surtan efecto. Después del reinicio, el programa debería iniciarse automáticamente junto con el arranque del sistema Operativo del PLC.

5.3 Configuración

La configuración de las comunicaciones entre los distintos elementos es crucial para que estos se puedan comunicar.

En primer lugar, la configuración IP del PLC. Para ello, se debe acceder al software TIA Portal y abrir el programa correspondiente. Una vez abierto, se debe pulsar en "**Device configuration**", abrir las propiedades de "**PROFINET onboard [X2]**" y pulsar en "**PROFINET interface [X2]**".

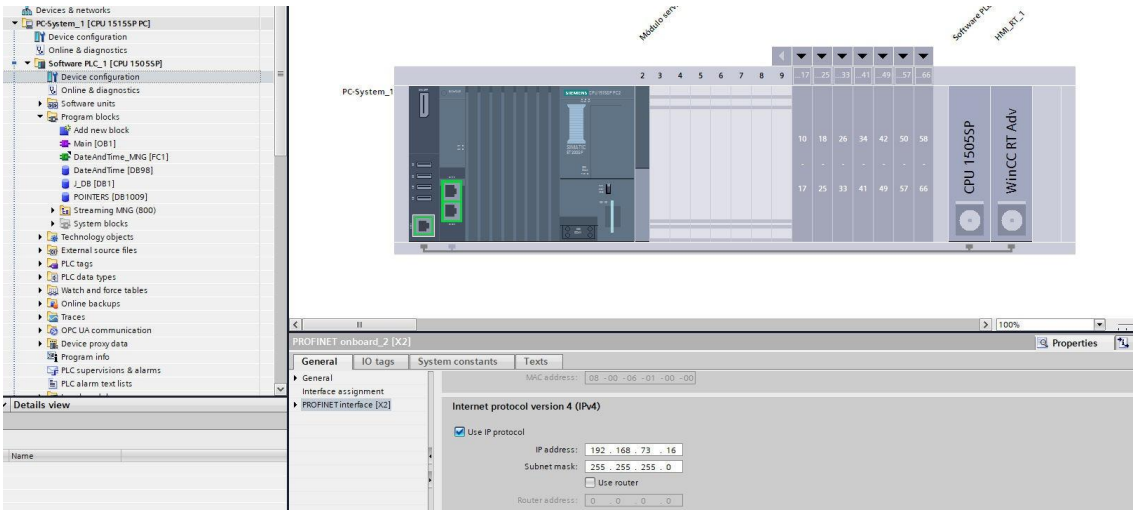


Figura 36: Pantalla del TIA Portal "Device Configuration", pestaña de propiedades.

En este puerto irá conectada la pantalla donde se ejecuta el programa de Visual Studio, por lo tanto, ambas deben estar en rango. En este caso se ha puesto la IP 192.168.73.16 con la máscara subred 255.255.255.0.

Por otro lado, al otro puerto físico del PLC se le ha dado la IP 192.168.0.6 con la máscara subred 255.255.255.0. En este puerto se conectará toda la periferia de servos y motores que no corresponden con este proyecto.

Por último, existe un puerto virtual proporcionado por siemens que permite la comunicación entre el PLC y la CPU. Abriendo de nuevo la pantalla **"Device configuration"**, pulsando en **"CPU 1505SP"**, dentro de **"Runtime communication interface"** y entrando en **"Ethernet addreses"** aparecerá el siguiente panel.

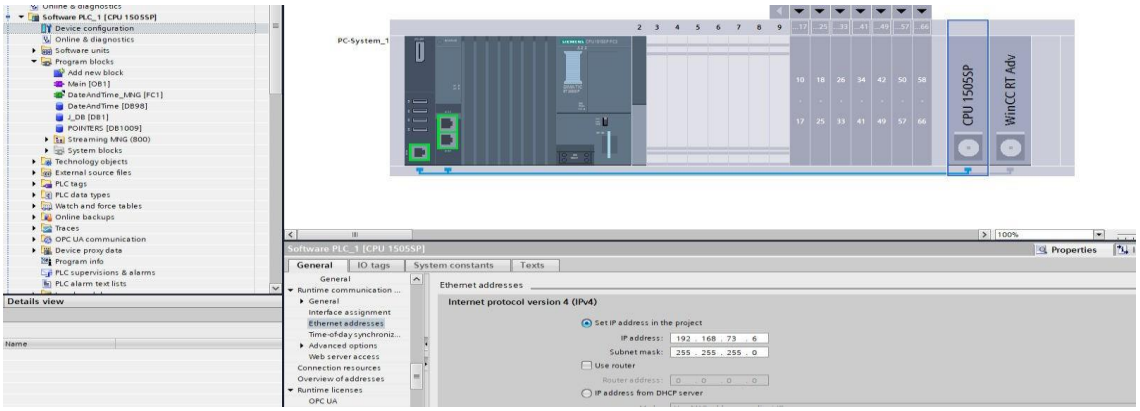


Figura 37: Pantalla del TIA Portal "Device Configuration", propiedades de la CPU.

A este puerto virtual se le ha dado la IP 192.168.73.6, con la máscara subred 255.255.255.0. En rango con el puerto físico donde se conectará la cámara y con el que se tendrá que establecer comunicaciones con el servidor **"Socket"**.

En cuanto a las direcciones IP de la pantalla (realmente es el sistema operativo del PLC), una es la física conectada al puerto y la otra es la virtual que se ha comentado anteriormente, ambas deben estar en rango respectivamente. IP física 192.168.73.240, IP virtual 192.168.73.200, ambas con la máscara subred 255.255.255.0.

Siguiendo los pasos del fabricante, cambiamos la IP predeterminada de la cámara por la siguiente IP 192.168.73.250 con la máscara subred 255.255.255.0. [6]

Por último, solo se debe configurar la cámara para que grabe las 24 horas del día sin descanso, ya que esta por defecto solo retransmite en vivo el video, no almacena nada. La función real de la aplicación es de cortar los trozos de video de interés (los procesos de templado y revenido de los cilindros corrugados) gracias a la fecha inicial y final del proceso de la pieza. Para esto es necesario el uso de un disco duro externo con capacidad suficiente para almacenar toda la información.

A continuación, se muestra en las figuras 38, 39 y 40 los pantallazos del *web server* de la cámara donde se muestra la configuración necesaria.

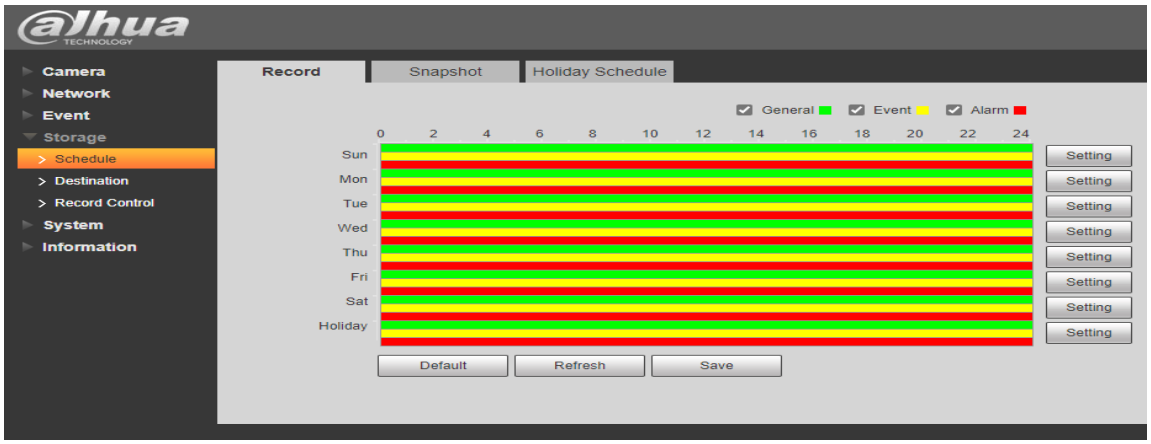


Figura 38. Pantalla *web server* de la cámara. **Storage>Schedule>Record**.

Figura 39. Pantalla *web server* de la cámara. **Storage>Destination>Path**.

Figura 40. Pantalla *web server* de la cámara. **Storage>Record Control**

5.4 Uso del Sistema

Una vez completada la instalación y configuración del sistema, el operario simplemente necesita iniciar la máquina y esperar a que el cliente se conecte al servidor. Si todo funciona correctamente, recibirá un mensaje en la pantalla indicando que todo está operativo.

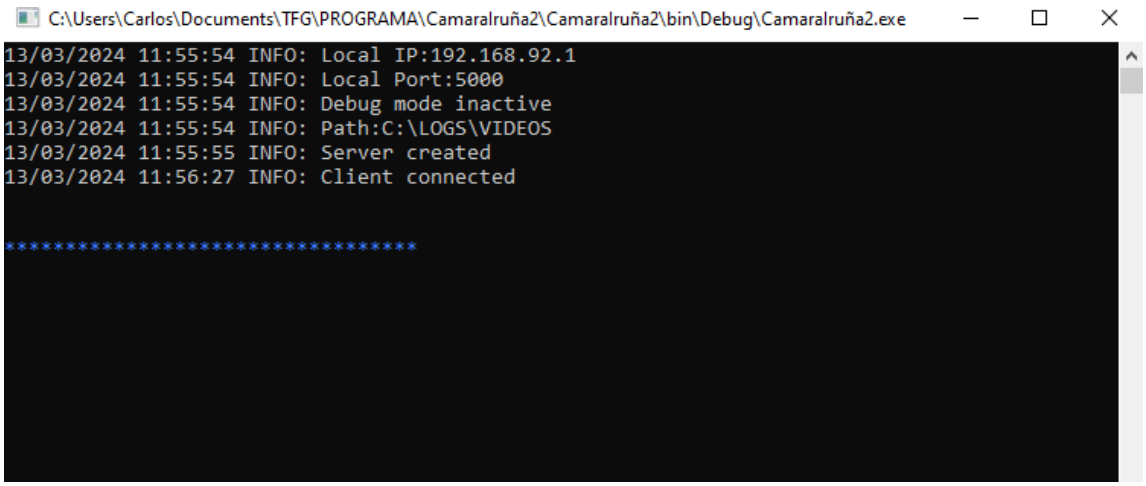


Figura 41: Confirmación de cliente conectado visto desde la consola.

En el contexto del sistema, el cliente envía periódicamente un mensaje de **"keep alive"** al servidor. Este mensaje tiene como objetivo principal confirmar que la conexión entre cliente y el servidor sigue activa y operativa. Al recibir este mensaje, el servidor sabe que el cliente sigue disponible y que la comunicación puede continuar sin problemas. Por lo tanto, por defecto, el operario verá lo siguiente.

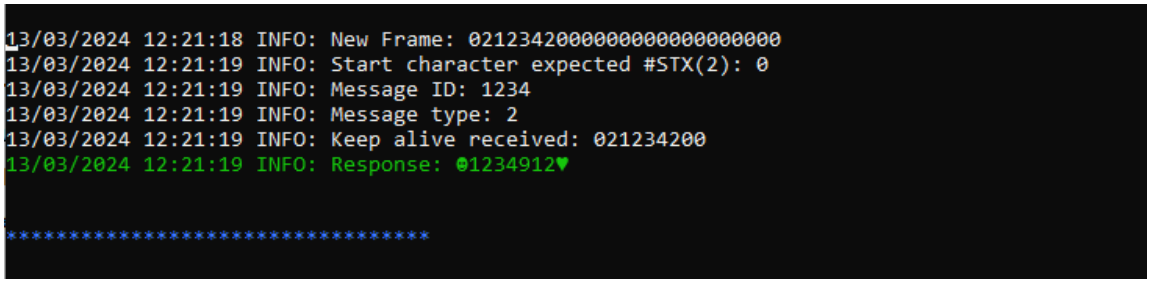


Figura 42: Confirmación de mensaje "keep alive" recibido visto en la consola.

Al final de cada calentamiento de pieza, el PLC almacenará los datos necesarios para la captura de video (fecha inicial y final, número de pieza y pieza OK/NOK) y activará el pulsador de solicitud. (referenciar con el diagrama de bloques de TIA Portal)

Si todo va bien, el servidor proporcionará una confirmación o mensaje de éxito indicando que el video se ha creado correctamente.

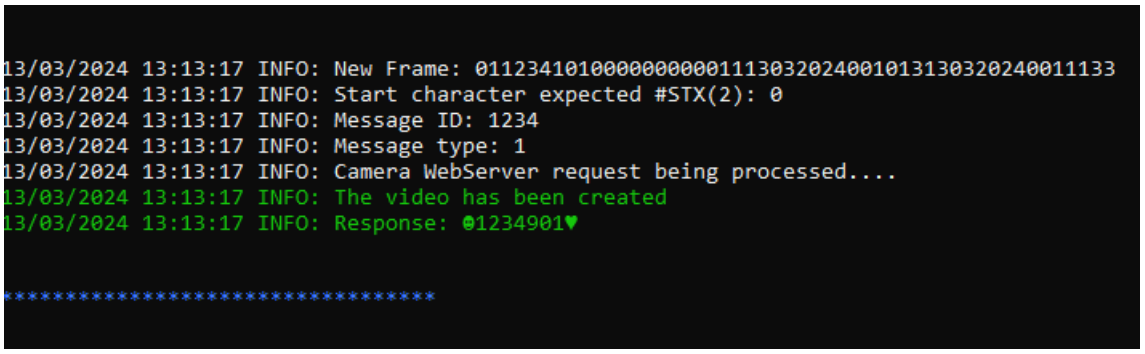


Figura 43: Confirmación de video creado visto en la consola.

Por lo tanto, en C:\LOGS\VIDEOS se debe haber creado una nueva carpeta con el año, mes y día en la que se calentó la pieza. En este caso se creó el 13/03/2024. Por otro lado, el video se debe haber guardado con la estructura explicada en las funciones, número de pieza, estado de la pieza y fecha y hora de la creación de la pieza.

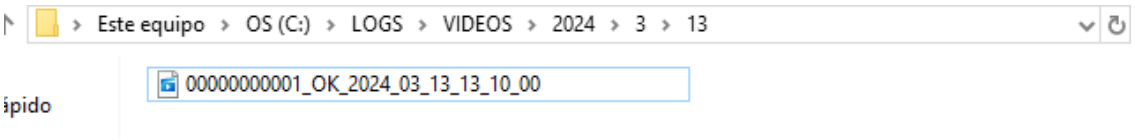


Figura 44: Ubicación del video creado.

Por último, si el operario necesita una información más detallada acerca de los procesos de la aplicación, deberá activar el modo "**debug**" en el archivo config.

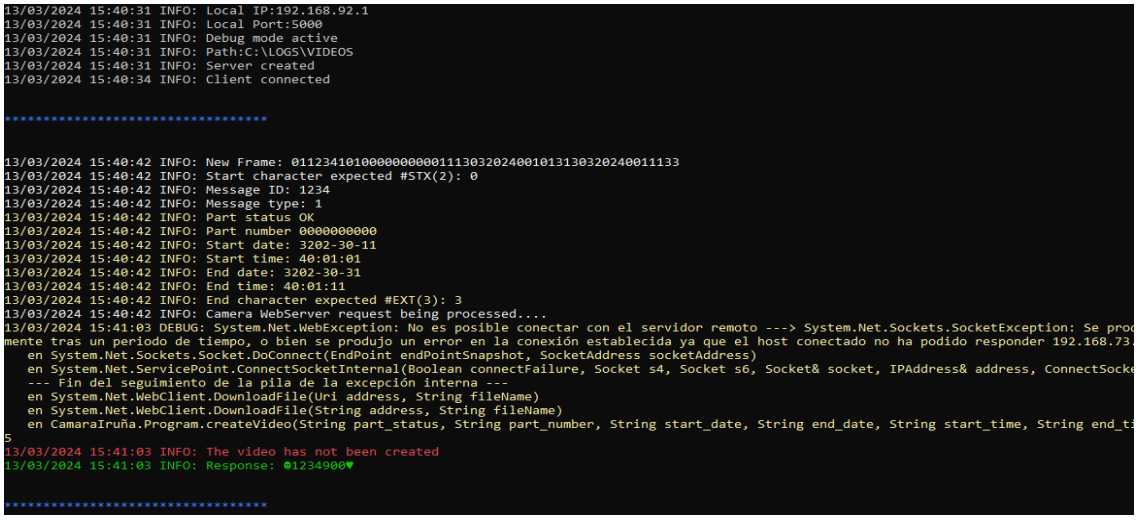


Figura 45: Ejemplo en la consola del modo "Debug".

5.5 Problemas frecuentes

Durante la ejecución del código, pueden surgir varios problemas que la consola irá indicando. A continuación, se muestran los que pueden pasar con mayor facilidad.

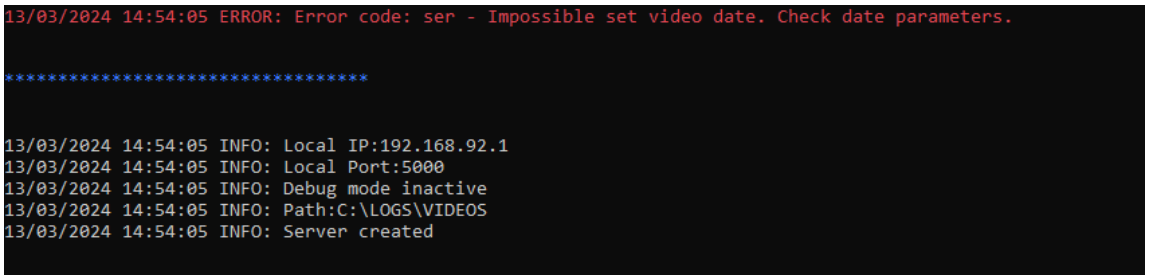


Figura 46: Ejemplo en la consola de un error en la función "Set_Current_Time".

Se ha detectado un inconveniente en la llamada a la función **"set_current_time ()"**. Una excepción ha sido lanzada y se ha mostrado en la pantalla. Este problema puede deberse a que la cámara no esté conectada, haya un problema de comunicación con ella o que la fecha indicada a la cámara sea incorrecta.

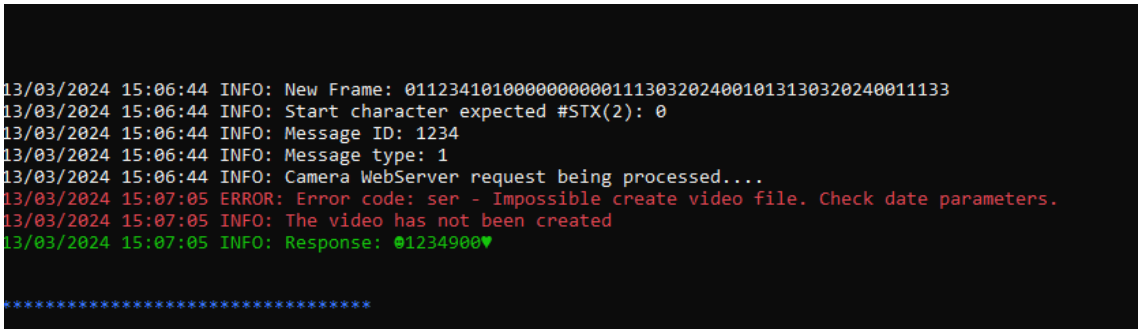


Figura 47: Ejemplo en la consola de un error en la función "Create_Video".

Se ha detectado un inconveniente en la llamada a la función **"CreateVideo ()"**. Una excepción ha sido lanzada y se ha mostrado en la pantalla. Este problema puede deberse a que la cámara no esté conectada, haya un problema de comunicación con ella o que la fecha indicada a la cámara sea incorrecta.

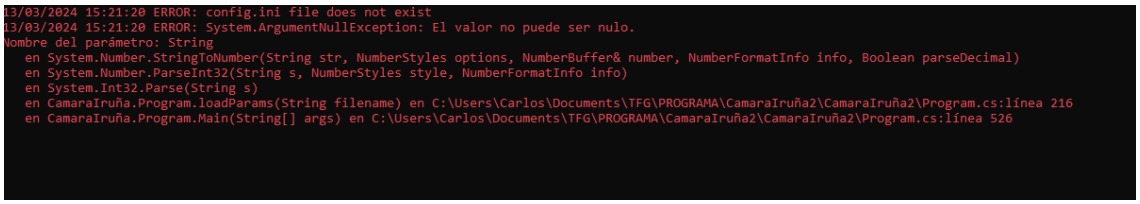


Figura 48: Ejemplo en la consola de un error en la función "LoadParams" por no encontrar el archivo "config".

Se ha detectado un inconveniente en la llamada a la función "**loadParams ()**". Una excepción ha sido lanzada y se ha mostrado en la pantalla. Este problema se debe a que el archivo config no se encuentra en la misma carpeta que el .exe y por lo tanto, la aplicación no es capaz de leer los parámetros.

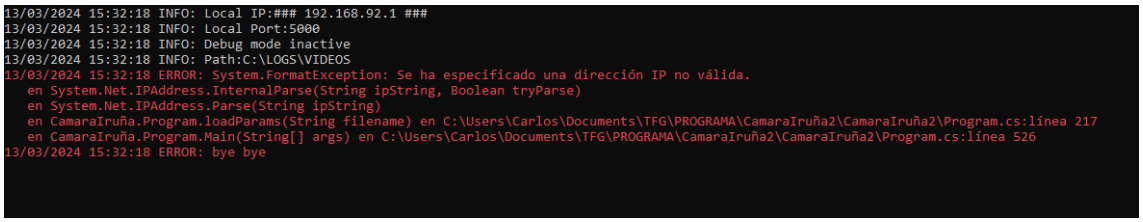


Figura 49: Ejemplo en la consola de un error en la función "LoadParams" por no estar bien configurado el archivo "config".

También, durante la carga de "**loadParams ()**", puede ocurrir que el archivo config no esté bien configurado, por lo que una excepción también será lanzada y mostrada por pantalla.

6. Conclusiones y posibles mejoras

En el transcurso de este proyecto, se ha propuesto diseñar e implementar un sistema de automatización industrial que integrara la comunicación entre un PLC y una cámara mediante sockets TCP/IP. El objetivo era desarrollar una solución efectiva y eficiente que facilitara la supervisión y control de procesos industriales, mejorando así la productividad y la calidad en entornos de fabricación.

Durante el proceso de desarrollo, se han enfrentado varios desafíos técnicos y logísticos, desde la configuración de la comunicación entre dispositivos hasta la gestión de errores y la implementación de medidas de seguridad. Sin embargo, gracias al trabajo y dedicación, se ha logrado superar estos obstáculos y completar con éxito la implementación del sistema.

En cuanto a los objetivos establecidos, se han logrado cumplir de manera satisfactoria. Se ha establecido una comunicación estable y confiable entre el PLC y la cámara, lo que permite la transferencia eficiente de datos y la supervisión en tiempo real de los procesos industriales.

Además, el sistema ha demostrado ser fácil de usar y mantener, lo que facilita su integración en entornos industriales existentes y su adaptación a diferentes aplicaciones y requisitos específicos. Los operarios técnicos pueden interactuar con el sistema de manera intuitiva y realizar tareas de configuración y monitoreo sin dificultad.

Sin embargo, es importante destacar que existen áreas de mejora identificadas durante el desarrollo del proyecto. Estas incluyen la optimización del rendimiento del sistema, la implementación de medidas adicionales de seguridad y la exploración de nuevas tecnologías y protocolos de comunicación, como el servidor OPC UA, para mejorar aún más la funcionalidad y la eficiencia del sistema.

La comunicación efectiva entre los dispositivos y sistemas es esencial para el funcionamiento eficiente de los procesos de producción. En este proyecto, se ha implementado la comunicación utilizando sockets TCP/IP para facilitar la transferencia de datos entre el PLC y la cámara. Sin embargo, a medida que buscamos mejorar la funcionalidad y la eficiencia de nuestro sistema, surge la oportunidad de considerar un enfoque alternativo: la adopción de un servidor OPC UA (**"Open Plataform Communications Unified Architecture"**).

El servidor OPC UA es una tecnología estándar en la industria de la automatización que ofrece una serie de ventajas significativas en comparación con la comunicación tradicional mediante sockets TCP/IP. Se explorarán los beneficios de migrar de la comunicación basada en sockets TCP/IP a un enfoque basado en un servidor OPC UA, y cómo esta transición puede mejorar la interoperabilidad, seguridad, escalabilidad y facilidad de administración de nuestro sistema.

Ventajas de utilizar un servidor OPC UA:

-Interoperabilidad: OPC UA es un estándar ampliamente aceptado en la industria que permite la interoperabilidad entre diferentes dispositivos y sistemas de automatización. Por lo tanto, el proyecto podría integrarse más fácilmente con otros sistemas y dispositivos que también sean compatibles con OPC UA.

-Seguridad avanzada: OPC UA ofrece características avanzadas de seguridad, incluida la autenticación, el cifrado y la integridad de los datos. Esto garantiza que la comunicación entre el PLC, la cámara y el sistema de visualización sea segura y protegida contra posibles amenazas de seguridad.

-Escalabilidad: OPC UA es altamente escalable y puede manejar grandes cantidades de datos y dispositivos conectados. Esto es especialmente beneficioso si el proyecto crece en el futuro y necesita manejar más dispositivos o datos.

-Descubrimiento de servicios: OPC UA admite el descubrimiento de servicios, lo que facilita la configuración y la administración de la red. Los dispositivos OPC UA pueden descubrirse automáticamente y conectarse entre sí sin necesidad de configuración manual, lo que simplifica la integración y la administración del sistema.

-Estándar industrial: OPC UA es un estándar reconocido en la industria, respaldado por organizaciones y fabricantes de renombre. Es el futuro al que todas las grandes empresas apuntan.

Todas estas propiedades mejorarían el proyecto al hacerlo más compatible con otros sistemas y dispositivos industriales, asegurando una comunicación segura y eficiente, y facilitando su mantenimiento y escalabilidad a largo plazo. [7]

Otras posibles mejoras del sistema:

-**Optimización de bloques en TIA Portal:** En lugar de funciones de ciclo (FC), se podría considerar el uso de bloques de función (FB), ya que estos últimos ofrecen ventajas significativas en términos de modularidad y reutilización de código. Además, al disponer de memoria, las FB pueden ser más convenientes para la aplicación, permitiendo una estructura más organizada del programa y más protegida.

-**Ampliación del intercambio de datos en las tramas:** Incrementar la cantidad de datos intercambiados entre los dispositivos permitiría obtener una visión más completa del sistema y detectar una gama más amplia de problemas potenciales. Esto podría lograrse añadiendo más funciones de diagnóstico, monitorización y control al protocolo de comunicación, lo que proporcionaría un mejor conocimiento del estado y rendimiento del sistema en tiempo real.

7. Referencias

[1] PHOENIXCONTACT. La PLCnext Technology conecta los mundos de las IT y OT.

https://www.phoenixcontact.com/es-es/industrias/plcnext-technology

[Accedido el 16/04/2024]

[2] TODOELECTRONICA. Cámaras Dahua.

https://www.todoelectronica.com/53626-camaras-dahua?gad_source=1&gclid=Cj0KCQjwq86wBhDiARIsAJhuphnStEH7oeyZltOtfFTueVECu4dxyD8PdEH9Qdt4slloo6JoNAjclwgaAlcUEALw_wcB)

[Accedido el 16/04/2024]

[3] MICROSOFT. Uso de sockets para enviar y recibir datos a través de TCP.

https://learn.microsoft.com/es-es/dotnet/fundamentals/networking/sockets/socket-services#create-a-socket-client

[Accedido en 15/04/2024].

[4] SIEMENS. Módulo TIA Portal 031-100 Principios básicos de la programación de FC con SIMATIC S7-1200.

[5] MICROSOFT. Agregar una aplicación para que se ejecute automáticamente al inicio en Windows 10.

https://support.microsoft.com/es-es/windows/agregar-una-aplicaci%C3%B3n-para-que-se-ejecute-autom%C3%A1ticamente-al-inicio-en-windows-10-150da165-dcd9-7230-517b-cf3c295d89dd

[Accedido 16/04/2024]

[6] CONTAVAL. Integración de cámara IP (Dahua) en comfort panel de Siemens.

[7] SIEMENS. Módulo TIA Portal 092-300 OPC UA con SIMATIC S7-1500 como servidor OPC, así como OPC SCOUT y SIMIT como clientes OPC.

8. Anexos

Código Visual Studio

Program.cs

```
1   using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Net.Sockets;
6  using System.Net;
7  using System.IO;
8  using System.Timers;
9  using System.Diagnostics;
10 using GH_Comms.Comunications;
11
12
13 namespace CamaraIruña
14 {
15     class Program
16     {
17
18         #region attributes
19
20         //Default file. MAKE SURE TO CHANGE THIS LOCATION AND FILE PATH TO
21 YOUR FILE
22         static readonly string configFile = @"config.ini";
23         static bool debugMode;
24         static string strLocalIP;
25         static string strLocalPort;
26         static string strPath;
27         static private int localPort;
28         static private IPAddress localIP;
29         static private byte[] ipAddress = new byte[4];
30         private static System.Timers.Timer aTimer;
31         private static System.Timers.Timer bTimer;
32         private static System.Timers.Timer cTimer;
33         private static System.Timers.Timer dTimer;
34         static private AsynchronousSocketListener Server;
35
36
37         #endregion
38
39 //******************* SET CURRENT TIME ON THE CAMERA
40 **************************//
41
42         static void Set_Current_Time()
43         {
44             WebClient CurrentTime = new WebClient();
45             CurrentTime.Credentials = new NetworkCredential("admin",
46 "gh1001gh");
47
48             DateTime dt = new DateTime(); // assigns default value
49 01/01/0001 00:00:00
50             dt = DateTime.Now;
```

```
51              string data = " ";
52              try
53              {
54                  CurrentTime.UploadString("http://192.168.73.250/cgi-
55 bin/global.cgi?action=setCurrentTime&time=" + dt.Year.ToString() + "-" +
56 dt.Month.ToString() + "-" + dt.Day.ToString() + "%20" + dt.Hour.ToString()
57 + ":" + dt.Minute.ToString() + ":" + dt.Second.ToString(), data);
58
59              }
60
61              catch(Exception e)
62              {
63
64                  string errorCode = e.ToString().Substring(55, 3);
65
66                  if(debugMode)
67                      Msg(false, ConsoleColor.Yellow, "DEBUG: ",
68 e.ToString());
69                  else
70                      Msg(true, ConsoleColor.Red, "ERROR: ", "Error code: " +
71 errorCode + " - Impossible set video date. Check date parameters.");
72              }
73
74          }
75
76
77 //***************** SHOW MESAGE FUNCTION ***************//
78
79          static void Msg(bool separator, ConsoleColor color, string
80 message1, string message2)
81          {
82              string aux = message1 + message2;
83
84              DateTime dt = new DateTime(); // assigns default value
85 01/01/0001 00:00:00
86              dt = DateTime.Now;
87              Console.ForegroundColor = color;
88
89              if (separator)
90              {
91                  Console.WriteLine(dt + " " + aux);
92                  Console.ForegroundColor = ConsoleColor.Blue;
93                  Console.WriteLine(Environment.NewLine);
94                  Console.WriteLine("*********************************");
95                  Console.WriteLine(Environment.NewLine);
96              }
97              else
98              {
99                  Console.WriteLine(dt + " " + aux);
100             }
101
102         }
103
104
105 //****************** REQUEST VIDEO ****************//
106
107         static int createVideo(string part_status, string part_number,
```

```
string start_date, string end_date, string start_time, string end_time)
        {
            int result = 0;
            string filename;
            string path;
string pathNfilename;
            byte[] byteResponse = new byte[1024];

            WebClient part = new WebClient(); //Create a webclient
including credentials
            part.Credentials = new NetworkCredential("admin", "gh1001gh");

            DateTime dt = new DateTime(); // assigns default value
01/01/0001 00:00:00
            dt = DateTime.Now;

            //The folders where the videos will be included are created
            path = strPath + "\\" + dt.Year;
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
            path = path + "\\" + dt.Month;
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
            path = path + "\\" + dt.Day;
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }

            //A unique name is assigned to each part
            filename = part_number + "_" + part_status + "_" + dt.Year +
"_" + dt.Month + "_" + dt.Day + "_" + dt.Hour + "_" + dt.Minute + "_" +
dt.Second + ".dav";
            pathNfilename = path + "\\" + filename;

            try
            {
                Msg(false, ConsoleColor.White, "INFO: ", "Camera WebServer
request being processed.……");

                part.DownloadFile("http://192.168.73.250/cgi-
bin/loadfile.cgi?action=startLoad&channel=1&startTime=" + start_date +
"%20" + start_time + "&endTime=" + end_date + "%20" + end_time +
"&subtype=0Types=dav", pathNfilename);
                result = 1;
            }
            catch (Exception e)
            {
                string errorCode = e.ToString().Substring(55, 3);

                if (debugMode)
                    Msg(false, ConsoleColor.Yellow, "DEBUG: ",
e.ToString());
```

```
165                 else
166                     Msg(false, ConsoleColor.Red, "ERROR: ", "Error code: "
167 + errorCode + " - Impossible create video file. Check date parameters.");
168 
169                     result = 0;
170                 }
171                 return result;
172             }
173 
174 //****************** LOAD PARAMETERS ******************//
175 
176             static bool loadParams(string filename)
177             {
178                 string auxPath = "";
179 
180                 try
181                 {
182                     if (File.Exists(configFile))
183                     {
184                         // Read a text file line by line.
185                         string[] parameter = File.ReadAllLines(configFile);
186 
187                         // Parameter 1 - Target IP
188                         strLocalIP = parameter[2];
189                         Msg(false, ConsoleColor.Gray, "INFO: Local IP:",
190 strLocalIP);
191                         // Parameter 2 - Port
192                         strLocalPort = parameter[4];
193                         Msg(false, ConsoleColor.Gray, "INFO: Local Port:",
194 strLocalPort);
195                         // Parameter 3 - Debug mode
196                         if (parameter[6] == "1")
197                         {
198                             debugMode = true;
199                              Msg(false, ConsoleColor.Gray, "INFO: ", "Debug mode
200 active");
201                         }
202                         else
203                         {
204                             debugMode = false;
205                              Msg(false, ConsoleColor.Gray, "INFO: ", "Debug mode
206 inactive");
207                         }
208                         // Parameter 4 - Path
209                         strPath = parameter[8];
210                         Msg(false, ConsoleColor.Gray, "INFO: Path:", strPath);
211 
212                         // Path manage
213                         string[] splitPath = strPath.Split('\\');
214 
215                         for (int i = 0; i < splitPath.Length; i++)
216                         {
217                             auxPath = auxPath + splitPath[i] + "\\";
218 
219                             if (Directory.Exists(auxPath))
220                             {
221                                 // do nothing!
```

```
                        }
                        else
                        {
                            Msg(false, ConsoleColor.Gray, "INFO: ",
"Creating directory" + auxPath);
                            Directory.CreateDirectory(auxPath);
                        }
                    }
                }
                else
                {
                    Msg(false, ConsoleColor.Red, "ERROR: ", "config.ini
file does not exist");
                }
            }
            catch(Exception e)
            {
                Msg(false, ConsoleColor.Red, "ERROR: excepción occurrida
durante la carga de para metros", e.Message);
                return false;
            }

            // Or specify a specific name in the current dir
            localPort = int.Parse(strLocalPort);
            localIP = IPAddress.Parse(strLocalIP);
            strPath = auxPath;
            return true;
        }

//********************  PROCESS RECEIVED MESSAGE
**********************  *****//

        static string processData(string data, int length)
        {
            int result = 0;
            string response;
            string strBuffer = data;

            Msg(false, ConsoleColor.White, "INFO: New Frame: ", data);

            //The frame coming from the PLC is encoded in ASCII  code.
            byte[] byte uffer = Encoding.ASCII.GetBytes(data);
            char[] char uffer = Encoding.ASCII.GetChars(byteBuff er, 0,
length);

            //Desearial ze header message
            string STX = charBuffer[0].ToString();
            Msg(false, ConsoleColor.White, "INFO: Start characte r expected
#STX(2): ", STX.ToStrin ());
            string msg_ D = charBuffer[2].ToString() +
charBuffer[3].ToString()  + charBuffer[4].ToString() +
charBuffer[5].ToString() ;
            Msg(false, ConsoleColor.White, "INFO: Message ID: " , msg_ID);
            string mess ge_type = charBuffer[6].ToString();
            Msg(false, ConsoleColor.White, "INFO: Message type: ",
message_type.ToString() ;

```

```
279             switch (message_type)
280             {
281 
282                 case "1": // get Video
283 
284                     string part_status = string.Empty;
285                     string part_number = string.Empty;
286                     string FinishedPart = string.Empty;
287                     string start_date = string.Empty;
288                     string start_time = string.Empty;
289                     string end_date = string.Empty;
290                     string end_time = string.Empty;
291                     string EXT = string.Empty;
292 
293                     try
294                     {
295 
296                         if (charBuffer[8].ToString() == "1")
297                             part_status = "OK";
298                         else
299                             part_status = "NOK";
300                         part_number = charBuffer[9].ToString() +
301 charBuffer[10].ToString() + charBuffer[11].ToString() +
302 charBuffer[12].ToString() + charBuffer[13].ToString() +
303 charBuffer[14].ToString() + charBuffer[15].ToString() +
304 charBuffer[16].ToString() + charBuffer[17].ToString() +
305 charBuffer[18].ToString();
306                         FinishedPart = charBuffer[19].ToString();
307                         start_date = charBuffer[24].ToString() +
308 charBuffer[25].ToString() + charBuffer[26].ToString() +
309 charBuffer[27].ToString() + '-' + charBuffer[22].ToString() +
310 charBuffer[23].ToString() + '-' + charBuffer[20].ToString() +
311 charBuffer[21].ToString();
312                         start_time = charBuffer[28].ToString() +
313 charBuffer[29].ToString() + ':' + charBuffer[30].ToString() +
314 charBuffer[31].ToString() + ':' + charBuffer[32].ToString() +
315 charBuffer[33].ToString();
316                         end_date = charBuffer[38].ToString() +
317 charBuffer[39].ToString() + charBuffer[40].ToString() +
318 charBuffer[41].ToString() + '-' + charBuffer[36].ToString() +
319 charBuffer[37].ToString() + '-' + charBuffer[34].ToString() +
320 charBuffer[35].ToString();
321                         end_time = charBuffer[42].ToString() +
322 charBuffer[43].ToString() + ':' + charBuffer[44].ToString() +
323 charBuffer[45].ToString() + ':' + charBuffer[46].ToString() +
324 charBuffer[47].ToString();
325                         EXT = charBuffer[49].ToString();
326 
327                     }
328                     catch (Exception e)
329                     {
330                         string ErrorCode = e.ToString();
331                         Msg(false, ConsoleColor.Red, "ERROR: ", "Error
332 code: " + ErrorCode + " ");
333                     }
334 
335                     if (debugMode == true)
```

```
336                     {
337                         Msg(false, ConsoleColor.Yellow, "INFO: Part status
338 ", part_status);
339                         Msg(false, ConsoleColor.Yellow, "INFO: Part number
340 ", part_number);
341                         Msg(false, ConsoleColor.Yellow, "INFO: Start date:
342 ", start_date);
343                         Msg(false, ConsoleColor.Yellow, "INFO: Start time:
344 ", start_time);
345                         Msg(false, ConsoleColor.Yellow, "INFO: End date: "
346 end_date);
347                         Msg(false, ConsoleColor.Yellow, "INFO: End time: ",
348 end_time);
349                         Msg(false, ConsoleColor.Yellow, "INFO: End
350 character expected #EXT(3): ", EXT.ToString());
351                     }
352
353                         result = createVideo(part_status, part_number,
354 start_date, end_date, start_time, end_time);
355
356                         if(result==1)
357                             Msg(false,ConsoleColor.Green, "INFO: ", "The
358 video has been created");
359                         else
360                             Msg(false,ConsoleColor.Red, "INFO: ", "The
361 video has not been created");
362
363                     // Compose response to PLC (ACK result)
364                     response = GenerateAck(msg_ID, result, byteBuffer);
365                     break;
366
367                 case "2": // Keep alive
368
369                     Msg(false,ConsoleColor.White, "INFO: Keep alive
370 received: ", strBuffer = strBuffer.Substring(0, 9));
371
372                     // Compose response to PLC (ACK true)
373                     response = GenerateAck(msg_ID, 12 ,byteBuffer);
374
375                     break;
376
377
378                 default:
379
380                     Msg(false, ConsoleColor.Red, "INFO: None of the
381 possible cases have been selected", "");
382
383
384                     // Compose response to PLC (ACK false)
385                     response = GenerateAck(msg_ID, 13, byteBuffer);
386
387                     break;
388             }
389
390             return response;
391         }
392 private static string GenerateAck(string msgID, int result, byte[]
```

```
byteBuffer)
        {
            string response;
            byte[] test = new byte[2];
            // Response .. #STX (02) + msgID + msgType  + result + #ETX
            byte[] responseByteBuffer = new byte[9];

            try
            {
                test[0] = (byte)(result / 10 + 48);
                test[1] = (byte)(result % 10 + 48);

                byte[] aux =
ASCIIEncoding.ASCII.GetBytes(result.ToString());

                if (aux.Length < 2)
                {
                    test[0] = 48;
                    test[1] = aux[0];
                }
                else
                {
                    test[0] = aux[0];
                    test[1] = aux[1];
                }


                responseByteBuffer[0] = 2;
                responseByteBuffer[1] = byteBuffer[2];
                responseByteBuffer[2] = byteBuffer[3];
                responseByteBuffer[3] = byteBuffer[4];
                responseByteBuffer[4] = byteBuffer[5];
                responseByteBuffer[5] = 57;
                responseByteBuffer[6] = (byte)(test[0]);
                responseByteBuffer[7] = (byte)(test[1]);
                responseByteBuffer[8] = 3;

                response = Encoding.Default.GetString(responseByteBuffer);

                return response;
            }
            catch(Exception e)
            {
                responseByteBuffer[0] = 2;
                responseByteBuffer[1] = 0;
                responseByteBuffer[2] = 0;
                responseByteBuffer[3] = 0;
                responseByteBuffer[4] = 0;
                responseByteBuffer[5] = 0;
                responseByteBuffer[6] = 0;
                responseByteBuffer[7] = 0;
                responseByteBuffer[8] = 3;
                response = Encoding.Default.GetString(responseByteBuffer);

                Msg(false, ConsoleColor.Red, "EROR: Exception generating
response ", e.Message);

```

```
450                 return response;
451             }
452         }
453 
454         //*************** TIMERS ***************//
455 
456         #region timers
457 
458         // On time event :: Waiting connection
459         private static void OnTimedEventWaitingConnection(Object source,
460 ElapsedEventArgs e)
461         {
462             Msg(false,ConsoleColor.White, "INFO: ", "still waiting for
463 conexion... ");
464         }
465         // Set timer
466         private static void SetTimerWaiting(double time)
467         {
468             // Create a timer with a two second interval.
469             aTimer = new System.Timers.Timer(time);
470             // Hook up the Elapsed event for the timer.
471             aTimer.Elapsed += OnTimedEventWaitingConnection;
472             aTimer.AutoReset = true;
473             aTimer.Enabled = true;
474         }
475         // Reset timer
476         private static void ResetTimerWaiting()
477         {
478             aTimer.Stop();
479             aTimer.Dispose();
480         }
481 
482         // On time event :: Timeout waiting ack
483         private static void OnTimedEventWaitingKeepAlive(Object source,
484 ElapsedEventArgs e)
485         {
486             Msg(false,ConsoleColor.White, "INFO: ", "timeout waiting for
487 ack...disconnecting socket ");
488         }
489         // Set timer
490         private static void SetTimerWaitingKeepAlive(double time)
491         {
492             // Create a timer with a two second interval.
493             bTimer = new System.Timers.Timer(time);
494             // Hook up the Elapsed event for the timer.
495             bTimer.Elapsed += OnTimedEventWaitingKeepAlive;
496             bTimer.AutoReset = true;
497             bTimer.Enabled = true;
498         }
499         // Reset timer
500         private static void ResetTimerWaitingKeepAlive()
501         {
502             bTimer.Stop();
503             bTimer.Dispose();
504         }
505 
506         // On time event :: Timeout waiting ack
```

```
507         private static void OnTimedEventReconnect(Object source,
508 ElapsedEventArgs e)
509         {
510             try
511             {
512                 Msg(false,ConsoleColor.White, "INFO: ", "Reconnecting
513 socket ");
514             }
515             catch(SocketException f)
516             {
517                 Msg(false, ConsoleColor.Red, "ERROR: Exception, impossible
518 to reopen the socket ", f.Message);
519             }
520         }
521         // Set timer
522         private static void SetTimerReconnect(double time)
523         {
524             // Create a timer with a two second interval.
525             cTimer = new System.Timers.Timer(time);
526             // Hook up the Elapsed event for the timer.
527             cTimer.Elapsed += OnTimedEventReconnect;
528             cTimer.AutoReset = true;
529             cTimer.Enabled = true;
530         }
531         // Reset timer
532         private static void ResetTimerReconnect()
533         {
534             cTimer.Stop();
535             cTimer.Dispose();
536         }
537
538
539         // On time event :: Timeout kill app
540         private static void OnTimedKillApp(Object source,  ElapsedEventArgs
541 e)
542         {
543             ResetTimerKillApp();
544             Environment.Exit(0);
545         }
546         // Set timer
547         private static void SetTimerKillApp(double time)
548         {
549             // Create a timer with a two second interval.
550             dTimer = new System.Timers.Timer(time);
551             // Hook up the Elapsed event for the timer.
552             dTimer.Elapsed += OnTimedKillApp;
553             dTimer.AutoReset = true;
554             dTimer.Enabled = true;
555         }
556         // Reset timer
557         private static void ResetTimerKillApp()
558         {
559             dTimer.Stop();
560             dTimer.Dispose();
561         }
562
563         #endregion
```

```
564
565 //************** CHECK INSTANCES ***************//
566         static bool checkInstances()
567         {
568             if
569 (Process.GetProcessesByName(Process.GetCurrentProcess().ProcessName).Length
570 > 1)
571                 return true;
572             else
573                 return false;
574         }
575
576 /*************************       MAIN
577 ************************************************************/
578
579
580
581         static void Main(string[] args)
582         {
583             bool paramsLoaded = false;
584             bool programAlreadyRunning = false;
585             programAlreadyRunning = checkInstances();
586             Set_Current_Time();
587
588             if (programAlreadyRunning)
589             {
590                 Msg(false,ConsoleColor.Red, "INFO: ", "Application already
591 running");
592                 SetTimerKillApp(4000);
593                 while (true)
594                 {
595                     // nothing to do
596                 }
597             }
598
599
600             // Load params
601             #region LoadParams
602             try
603             {
604                 paramsLoaded = loadParams("config.ini");
605             }
606             catch (Exception e1)
607             {
608                 Msg(false,ConsoleColor.Red, "ERROR: ", e1.ToString());
609             }
610             #endregion
611
612             if (paramsLoaded)
613             {
614                 // Create communication server
615                 #region Comm server
616                 try
617                 {
618                     Server = new AsynchronousSocketListener(localIP,
619 localPort);
620                     // Server events
```

```
621                     Server.ClientConnected += OnClientConnected;
622                     Server.ClientDisconnected += OnClientDisconnected;
623                     Server.MessageReceived += OnMessageReceived;
624 
625                     Msg(false,ConsoleColor.Gray, "INFO: ", "Server
626 created");
627                 }
628                 catch (Exception e2)
629                 {
630                     Msg(false,ConsoleColor.Red, "ERROR: ", e2.ToString());
631                 }
632                 #endregion
633 
634                 while (true)
635                 {
636                     // nothing to do
637                 }
638             }
639             else
640             {
641                 // The program ends
642 
643                 Msg(false,ConsoleColor.Red, "ERROR: ", "bye bye ");
644 
645             }
646         }
647 
648 
649 
650 /*************       COMMS METHODS      **********************/
651 
652         #region Comms methods
653         // Client connected succesfully
654         static private void OnClientConnected(object sender, EventArgs e)
655         {
656             // Client connected
657             Msg(true,ConsoleColor.Gray, "INFO: ", "Client connected");
658         }
659 
660         // Client disconnected
661         static private void OnClientDisconnected(object sender, EventArgs
662 e)
663         {
664             // Client disconnected
665             Msg(true,ConsoleColor.Gray, "INFO: ", "Client disconnected");
666         }
667 
668         // New message received
669         static private void OnMessageReceived(object sender,
670 MessageReceivedEventArgs e)
671         {
672             string response;
673             string msg;
674             Socket handler;
675             msg = e.buffer;
676             handler = e.handler;
677             // New message received
```

```
678             response = processData(msg, msg.Length);
679 
680             // Response to PLC
681 
682             if (response != "")
683             {
684                     Msg(true, ConsoleColor.Green, "INFO: ", "Response: " +
685 response);
686                     Console.WriteLine(Environment.NewLine);
687                     Server.Send(handler,  response);
688             }
689 
690         }
691 
692         #endregion
693 
694     }
695 }
```

AsynchronousSocketListener.cs

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Linq;
 4 using System.Text;
 5 using System.Net;
 6 using System.Net.Sockets;
 7 using System.Threading;
 8 
 9 namespace GH_Comms.Comunications
10 {
11     public class AsynchronousSocketListener
12     {
13         // Public events
14         public event EventHandler ClientConnected;
15         public event EventHandler ClientDisconnected;
16         public event EventHandler<MessageReceivedEventArgs>
17 MessageReceived;
18 
19 
20 
21         protected virtual void OnClientConnected(EventArgs e)
22         {
23             EventHandler handler = ClientConnected;
24             if (handler != null)
25             {
26                 handler(this, e);
27             }
28         }
29 
30         protected virtual void OnClientDisconnected(EventArgs e)
31         {
32             EventHandler handler = ClientDisconnected;
33             if (handler != null)
34             {
```

```
35                 handler(this, e);
36             }
37         }
38 
39         protected virtual void
40 OnMessageReceived(MessageReceivedEventArgs e)
41         {
42             EventHandler<MessageReceivedEventArgs> handler =
43 MessageReceived;
44             if (handler != null)
45             {
46                 handler(this, e);
47             }
48         }
49 
50         // Thread signal.
51         public ManualResetEvent allDone = new
52 ManualResetEvent(false);
53 
54         // Constructor sin argumentos
55         public AsynchronousSocketListener()
56         {
57         }
58 
59         public AsynchronousSocketListener(IPAddress Ip, Int32 Port)
60         {
61             StartListening(Ip, Port);
62         }
63 
64         public void StartListening(IPAddress _ipAddress, Int32
65 _Port)
66         {
67             // Data buffer for incoming data.
68             byte[] bytes = new Byte[1024];
69 
70             // Data buffer for incoming data.
71             byte[] RcvBuffer = new Byte[1024];
72 
73             // Establish the local endpoint for the socket.
74             IPEndPoint localEndPoint = new IPEndPoint(_ipAddress,
75 _Port);
76 
77             // Create a TCP/IP socket.
78             Socket listener = new
79 Socket(AddressFamily.InterNetwork,
80                 SocketType.Stream, ProtocolType.Tcp);
81 
82             // j.m. Added to allow reusing same local endpoint
83 address.
84             listener.SetSocketOption(SocketOptionLevel.Socket,
85 SocketOptionName.ReuseAddress, true);
86 
87             // Bind the socket to the local endpoint and listen for
88 incoming connections.
89             try
90             {
91                 listener.Bind(localEndPoint);
```

```
92                  listener.Listen(100);
93                  allDone.Reset();
94
95                  // Start an asynchronous socket to listen for
96 connections.
97                  listener.BeginAccept(
98                          new AsyncCallback(AcceptCallback),
99                          listener);
100             }
101             catch (SocketException e)
102             {
103                 //Console.WriteLine(e.ToString());
104             }
105         }
106
107         public void AcceptCallback(IAsyncResult ar)
108         {
109             // Signal the main thread to continue.
110             allDone.Set();
111
112             // Get the socket that handles the client request.
113             Socket listener = (Socket)ar.AsyncState;
114             //Socket handler = listener.EndAccept(ar);
115             Socket handler = listener.EndAccept(ar);
116
117             // Start an asynchronous socket to listen for
118 connections.
119             listener.BeginAccept(
120                     new AsyncCallback(AcceptCallback),
121                     listener);
122
123             // Start an asynchronous socket to listen for
124 connections.
125             //listener.BeginDisconnect(true,
126             //        new AsyncCallback(CancelCallback),
127             //        listener);
128
129             // Create the state object.
130             StateObject state = new StateObject();
131             state.workSocket = handler;
132             handler.BeginReceive(state.buffer, 0,
133 StateObject.BufferSize, 0,
134                 new AsyncCallback(ReadCallback), state);
135
136             // Cliente conectado
137             OnClientConnected(EventArgs.Empty);
138         }
139
140         public void CancelCallback(IAsyncResult ar)
141         {
142             // se desconecta el cliente
143             // Get the socket that handles the client request.
144             Socket listener = (Socket)ar.AsyncState;
145         }
146
147         public void Dispose()
148         {
```

```
149             // cacafuti
150         }
151 
152         public void ReadCallback(IAsyncResult ar)
153         {
154             String content = String.Empty;
155 
156             // Retrieve the state object and the handler socket
157             // from the asynchronous state object.
158             StateObject state = (StateObject)ar.AsyncState;
159             Socket handler = state.workSocket;
160 
161             try
162             {
163                 // Read data from the client socket.
164                 int bytesRead = handler.EndReceive(ar);
165 
166                 if (bytesRead > 0)
167                 {
168                     // There  might be more data, so store the data
169 received so far.
170                     state.sb.Append(Encoding.ASCII.GetString(
171                        state.buffer, 0, bytesRead));
172 
173                     // Check for end-of-file tag. If it is not
174 there, read
175                     // more data.
176                     content = state.sb.ToString();
177                     //if (content.IndexOf("\r") > -1)
178                     //{
179                     // All the data has been read from the
180                     // client. Display it on the console.
181                     //Console.WriteLine("Read {0} bytes from
182 socket. \n Data : {1}",
183                     //    content.Length, content);
184                     // Echo the data back to the client.
185                     //Send(handler, content);
186 
187                     // Mensaje recibido
188                     MessageReceivedEventArgs args = new
189 MessageReceivedEventArgs();
190                     args.buffer = content;
191                     args.handler = handler;
192                     OnMessageReceived(args);
193                     state.sb.Clear();
194                     //}
195                     // else
196                     // {
197                     // Not all data received. Get more.
198                     handler.BeginReceive(state.buffer, 0,
199 StateObject.BufferSize, 0,
200                     new AsyncCallback(ReadCallback), state);
201                     //}
202 
203                     // Start an asynchronous socket to listen  for
204 connections.
205                     //handler.BeginDisconnect(true,
```

```
206                     //        new AsyncCallback(CancelCallback),
207                     //         handler);
208
209                 }
210             }
211             catch (Exception e)
212             {
213
214             }
215         }
216
217         public void Send(Socket handler, String data)
218         {
219             // Convert the string data to byte data using ASCII
220 encoding.
221             byte[] byteData = Encoding.ASCII.GetBytes(data);
222
223             // Begin sending the data to the remote device.
224             handler.BeginSend(byteData, 0, byteData.Length, 0,
225                 new AsyncCallback(SendCallback), handler);
226         }
227
228         private static void SendCallback(IAsyncResult ar)
229         {
230             try
231             {
232                 // Retrieve the socket from the state object.
233                 Socket handler = (Socket)ar.AsyncState;
234
235                 // Complete sending the data to the remote device.
236                 int bytesSent = handler.EndSend(ar);
237                 //Console.WriteLine("Sent {0} bytes to client.",
238 bytesSent);
239
240                 //handler.Shutdown(SocketShutdown.Both);
241                 //handler.Close();
242
243             }
244             catch (Exception e)
245             {
246                 //Console.WriteLine(e.ToString());
247             }
248         }
249     }
250 }
```

MessageReceivedEventArgs.cs

```
1  using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Net;
6 using System.Net.Sockets;
7 using System.Threading;
8
9 namespace GH_Comms.Comunications
10 {
11     public class MessageReceivedEventArgs : EventArgs
12     {
13         public string buffer { get; set; }
14         public Socket handler {get; set;}
15     }
16 }
```

StateObject.cs

```
1  using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Net;
6 using System.Net.Sockets;
7 using System.Threading;
8
9 namespace GH_Comms.Comunications
10 {
11     class StateObject
12     {
13         // Client  socket.
14         public Socket workSocket = null;
15         // Size of receive buffer.
16         public const int BufferSize = 1024;
17         // Receive buffer.
18         public byte[] buffer = new byte[BufferSize];
19         // Received data string.
20         public StringBuilder sb = new StringBuilder();
21     }
22 }
```

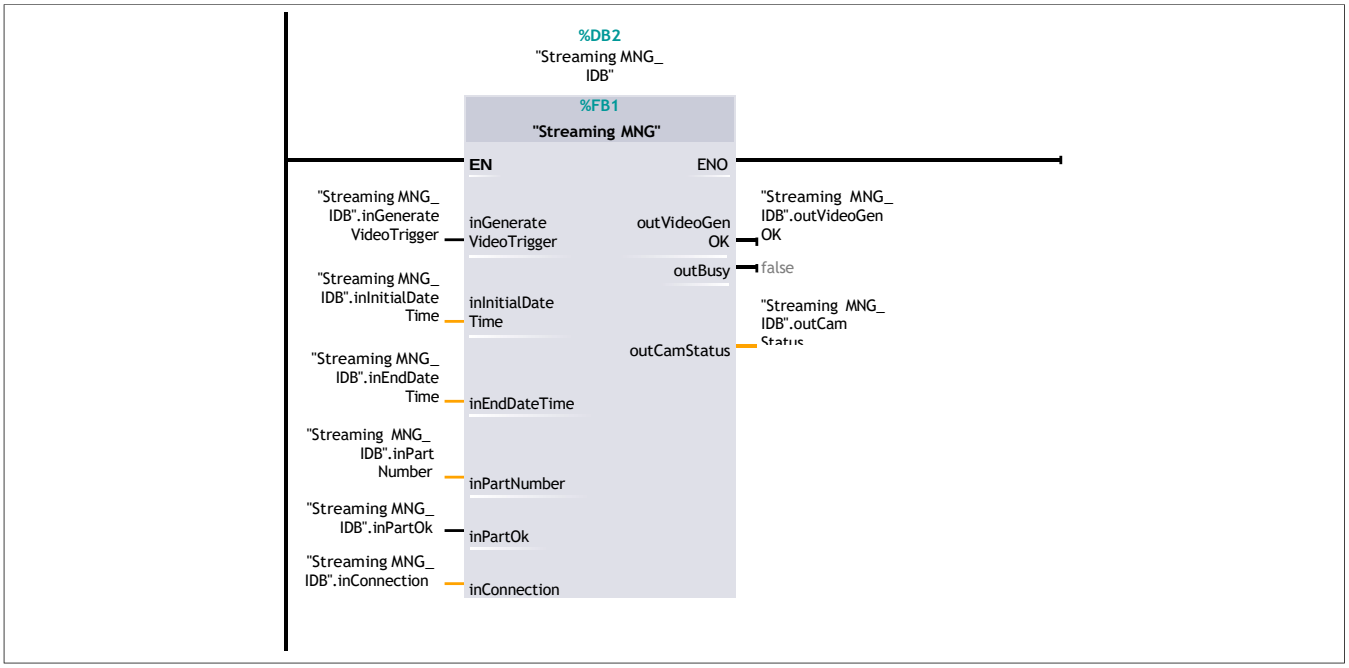
Program blocks

Main [OB1]

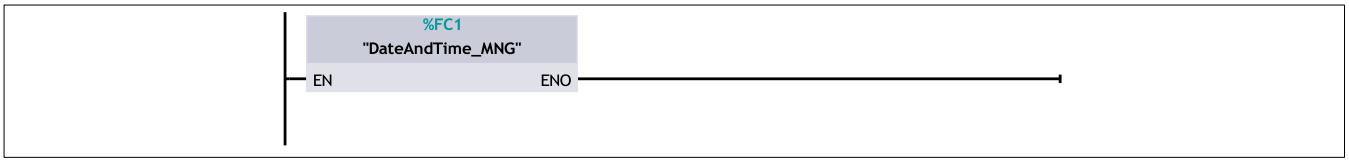
Main Properties					
General					
Name	Main	**Number**	1	**Type**	OB
Language	LAD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value
▼ Input		
Initial_Call	Bool	
Remanence	Bool	
▼ Temp		
tmpRetVal	Int	
Constant		

Network 1:



Network 2:



Program blocks

J_DB [DB1]

J_DB Properties					
General					
Name	J_DB	**Number**	1	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain
▼ Static			
trigger	Bool	false	False
startDateTime	Date_And_Time	DT#1990-01-01-00:00:00	False
endDateTime	Date_And_Time	DT#1990-01-01-00:00:00	False
partNumber	DInt	0	False
partStatus	Bool	false	False

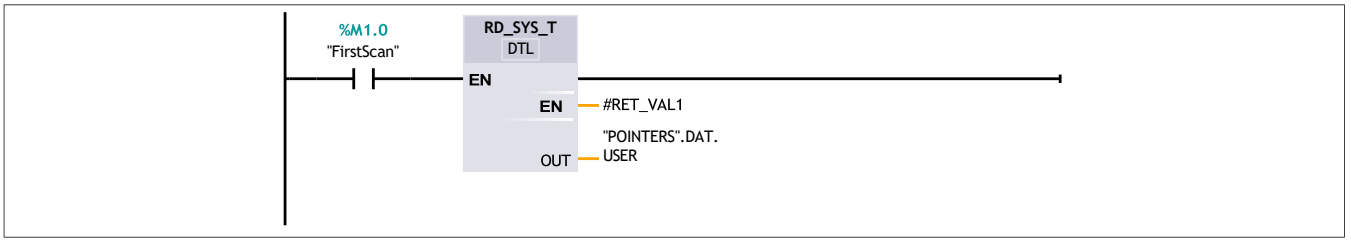
Program blocks

DateAndTime_MNG [FC1] [DateAndTime_MNG V 0.1.0]

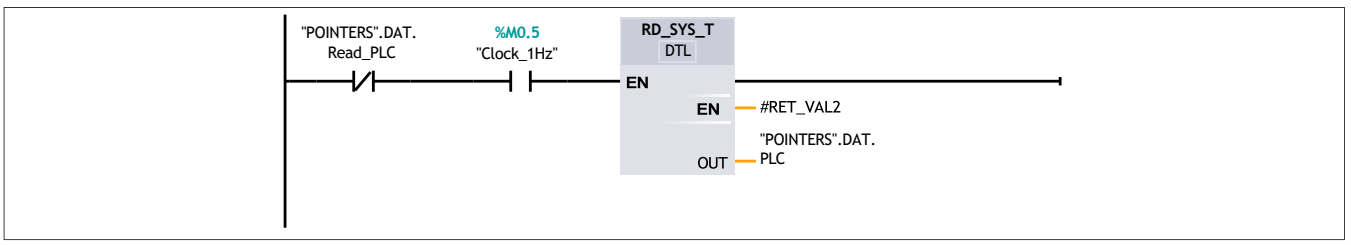
DateAndTime_MNG Properties					
General					
Name	DateAndTime_MNG	**Number**	1	**Type**	FC
Language	LAD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value
Input		
Output		
InOut		
▼ Temp		
RET_VAL1	Word	
RET_VAL2	Word	
RET_VAL3	Word	
RET_VAL4	Word	
Constant		
▼ Return		
DateAndTime_MNG	Void	

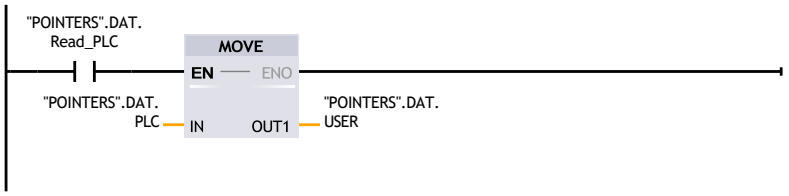
Network 1: Set initial value at USER



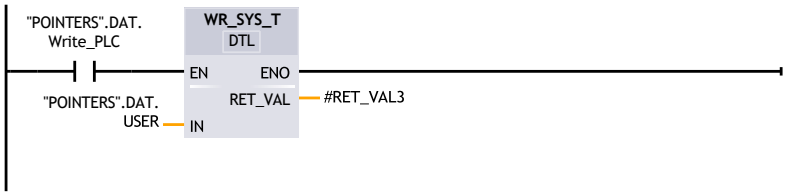
Network 2: Read date from PLC



Network 3: Move PLC date to USER



Network 4: Write USER data to PLC



Program blocks

DateAndTime [DB98]

DateAndTime Properties					
General					
Name	DateAndTime	**Number**	98	**Type**	DB
Language	DB	**Numbering**	Manual		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain
▼ Static			
DAT	"UDT_DateAndTime"		False

Program blocks

POINTERS [DB1009]

POINTERS Properties					
General					
Name	POINTERS	**Number**	1009	**Type**	DB
Language	DB	**Numbering**	Manual		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain
▼ Static			
ControlTask	Array[0..3] of Int		True
Coordination	Array[0..15] of Bool		True
Register	Array[0..4] of Int		True
Image	Array[0..4] of Int		True
DAT	"UDT_DateAndTime"		True
REC_1_REG_Number	Int	1	True
REC_1_REG_Name	String	''	True
REC_1_REC_Number	Int	1	True
REC_1_REC_Name	String	''	True
Id_Recipe_1_Log	DInt	0	True
Mailbox_Recipe_1_Log	Int	0	True
Recipe_1_modified	Int	0	True
REC_2_REG_Number	Int	1	True
REC_2_REG_Name	String	''	True
REC_2_REC_Number	Int	0	True
REC_2_REC_Name	String	''	True
Id_Recipe_2_Log	DInt	0	True
Mailbox_Recipe_2_Log	Int	0	True
Recipe_2_modified	Int	0	True

Program blocks / Streaming MNG (800)

Replace [FC802]

Replace Properties					
General					
Name	Replace	**Number**	802	**Type**	FC
Language	SCL	**Numbering**	Manual		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value
▼ Input		
inCharToErase	Char	
inCharToFill	Char	
Output		
▼ InOut		
inoutBuffer	Array[0..50] of Byte	
▼ Temp		
tmpIndex	Int	
Constant		
▼ Return		
Replace	Void	

```
0001
0002 FOR #tmpIndex := 0 TO 50 DO
0003    // Statement section FOR
0004    // start of buffer data
0005     IF #inoutBuffer[#tmpIndex] = 2 THEN
0006       ;
0007    END_IF;
0008
0009     IF #inoutBuffer[#tmpIndex] = ' ' THEN
0010       #inoutBuffer[#tmpIndex] := '0';
0011    END_IF;
0012
0013    // end of buffer data
0014     IF #inoutBuffer[#tmpIndex] = 3 THEN
0015       RETURN;
0016    END_IF;
0017 END_FOR;
0018
0019
```

Program blocks / Streaming MNG (800)

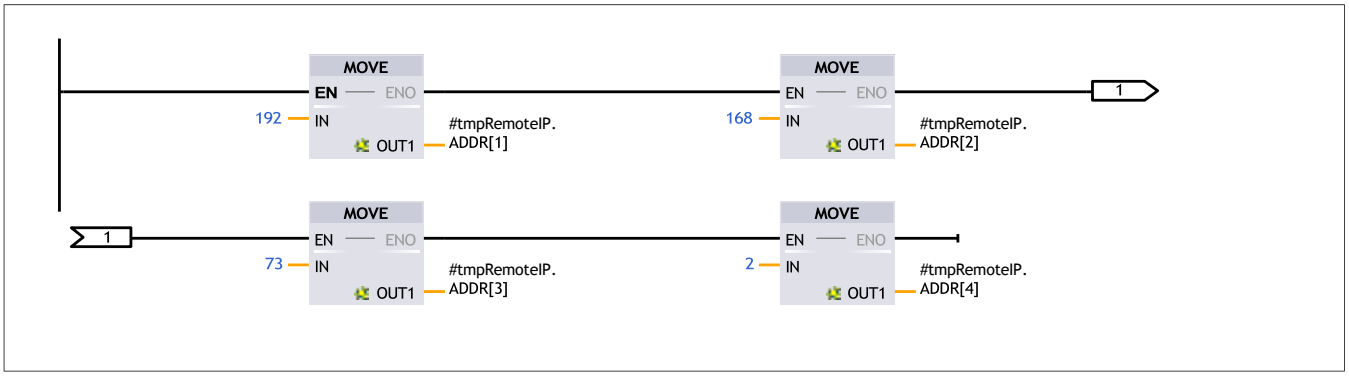
Streaming MNG [FB1]

Streaming MNG Properties					
General					
Name	Streaming MNG	**Number**	1	**Type**	FB
Language	LAD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

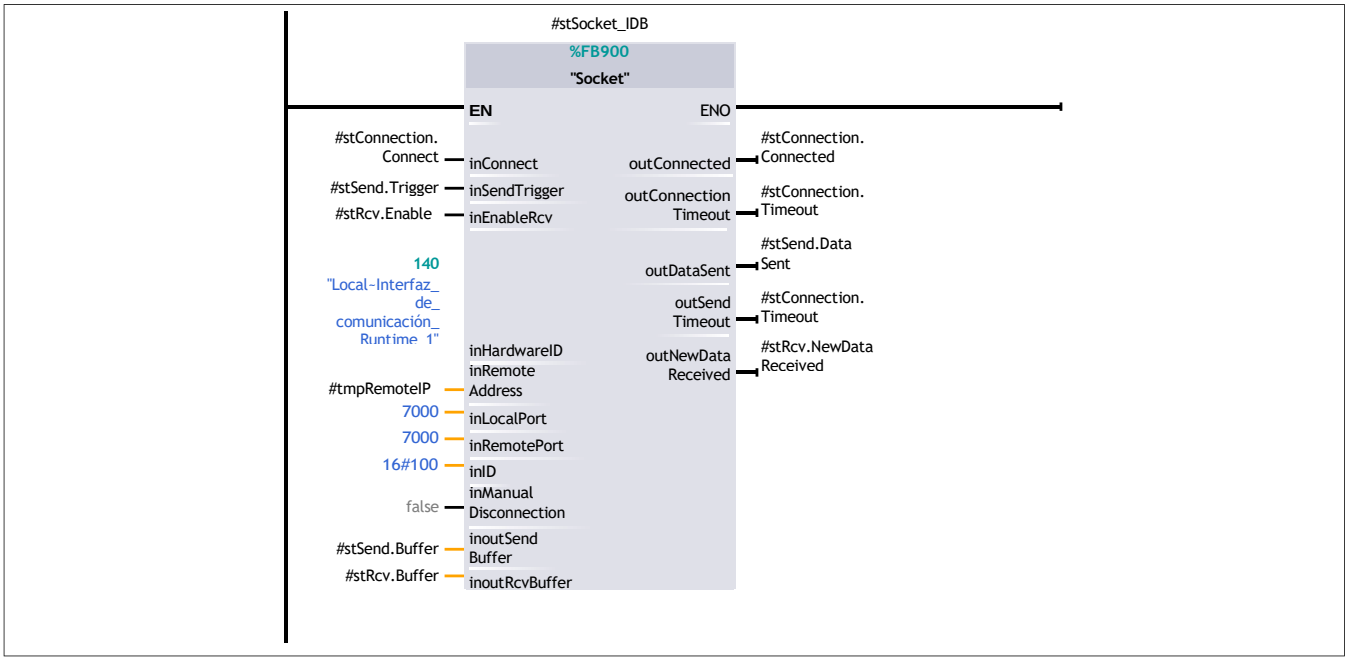
Name	Data type	Default value	Retain
▼ Input			
inGenerateVideoTrigger	Bool	false	Non-retain
inInitialDateTime	DTL	DTL#1970-01-01-00:00:00	Non-retain
inEndDateTime	DTL	DTL#1970-01-01-00:00:00	Non-retain
inPartNumber	DInt	0	Non-retain
inPartOk	Bool	false	Non-retain
inConnection	Struct		Non-retain
▼ Output			
outVideoGenOK	Bool	false	Non-retain
outBusy	Bool	false	Non-retain
outCamStatus	Int	0	Non-retain
InOut			
▼ Static			
stSocket_IDB	"Socket"		
stConnection	Struct		Non-retain
stSend	Struct		Non-retain
stRcv	Struct		Non-retain
Aux	Array[0..20] of Bool		Non-retain
stInitialDateTime	DTL	DTL#1970-01-01-00:00:00	Non-retain
stEndDateTime	DTL	DTL#1970-01-01-00:00:00	Non-retain
stPartNumber	DInt	0	Non-retain
stPartOk	Bool	false	Non-retain
stProcessingVideoRequest	Bool	false	Non-retain
stP	Array[0..10] of Bool		Non-retain
stTON	Array[0..10] of TON_TIME		Non-retain
stAuxDint	Array[0..10] of DInt		Non-retain
stAuxCharArray	Array[0..10] of Char		Non-retain
stEmptyBuffer	Array[0..50] of Byte		Non-retain
▼ Temp			
tmpRemoteIP	IP_V4		
tmpString	String		
tmpInt	Int		
tmpKeepAlive	Bool		

Name	Data type	Default value	Retain
tmpBuffer	Array[0..50] of Byte		
Constant			
cKeepAlive	Int	2	
cGetVideo	Int	1	
cAck	Int	9	

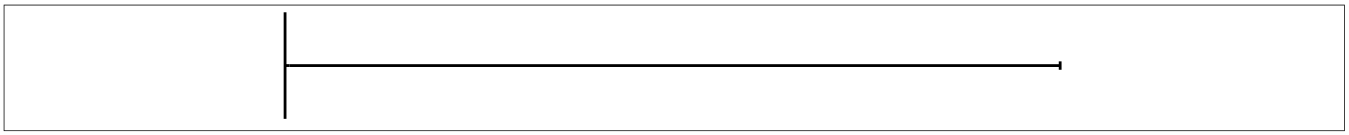
Network 1: Remote adress



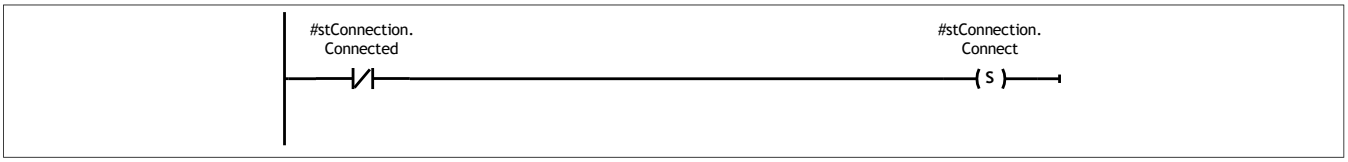
Network 2: Socket



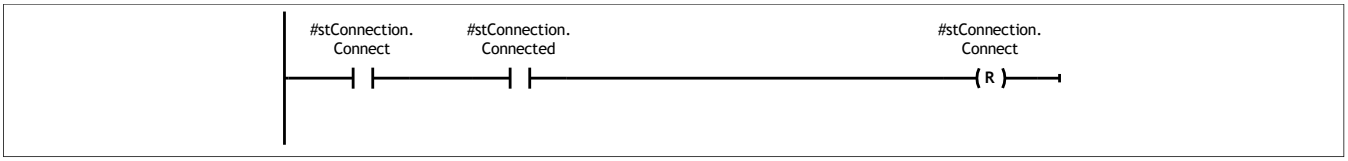
Network 3: ********** SOCKET MNG **********



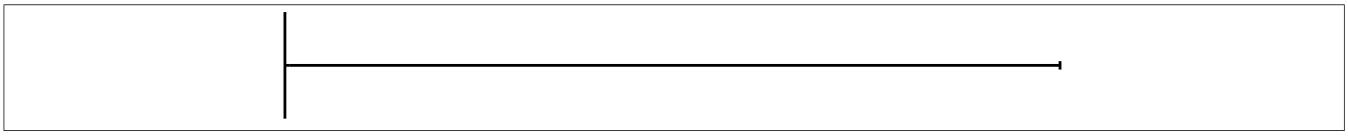
Network 4: Try connection to server



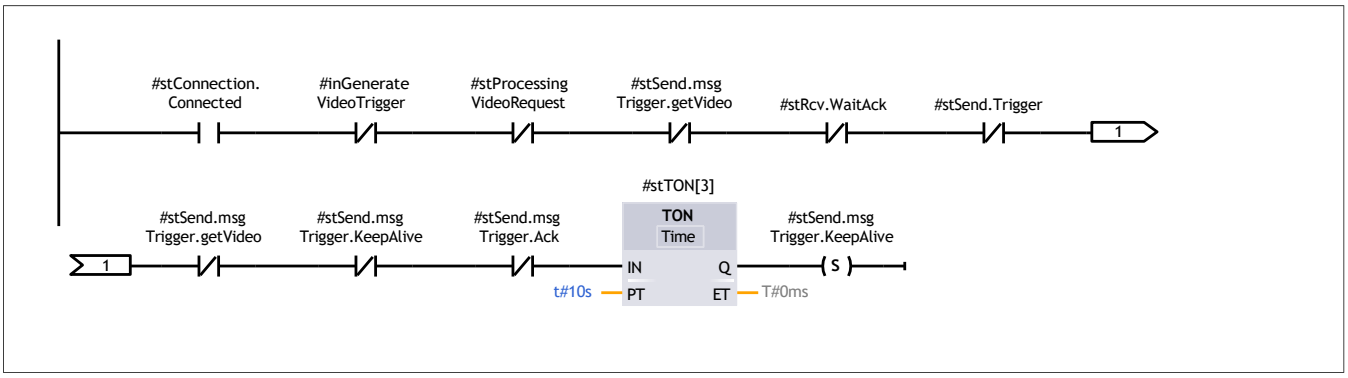
Network 5: Connected



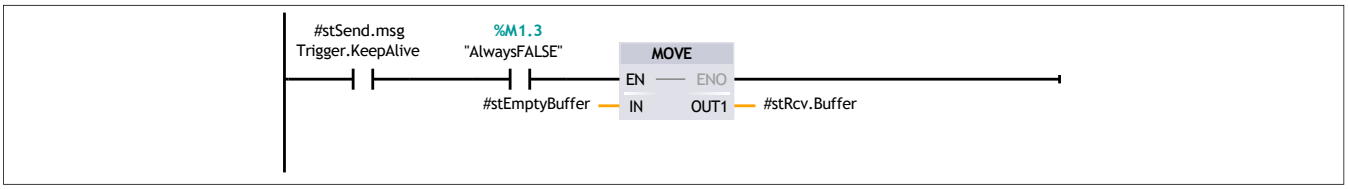
Network 6: ********** KEEP ALIVE ("2") **********



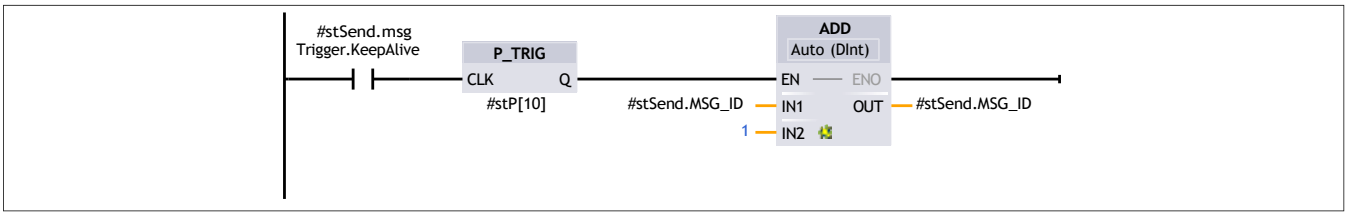
Network 7: SET Trigger



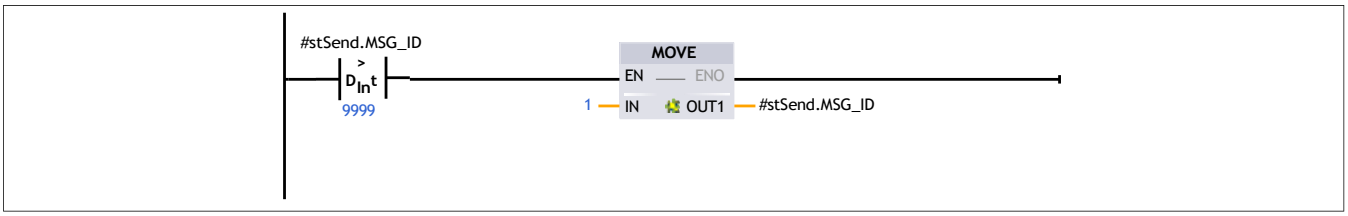
Network 8: Clear buffer



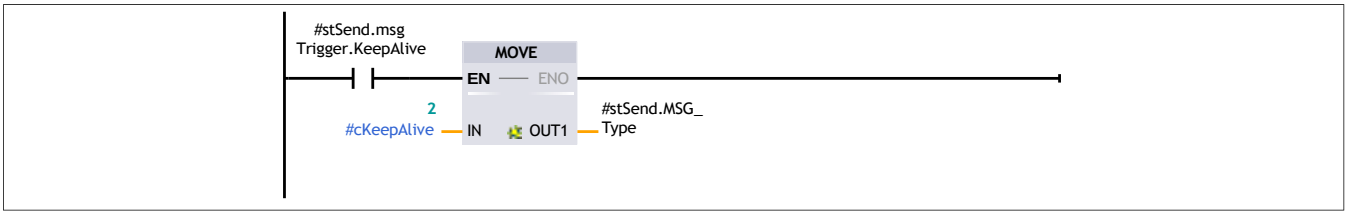
Network 9: Send message ID



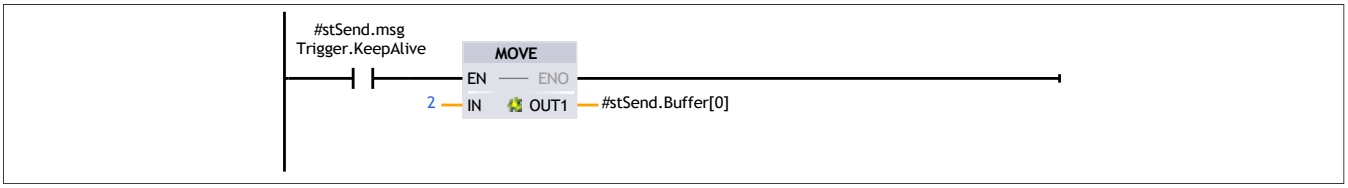
Network 10: Reset message ID



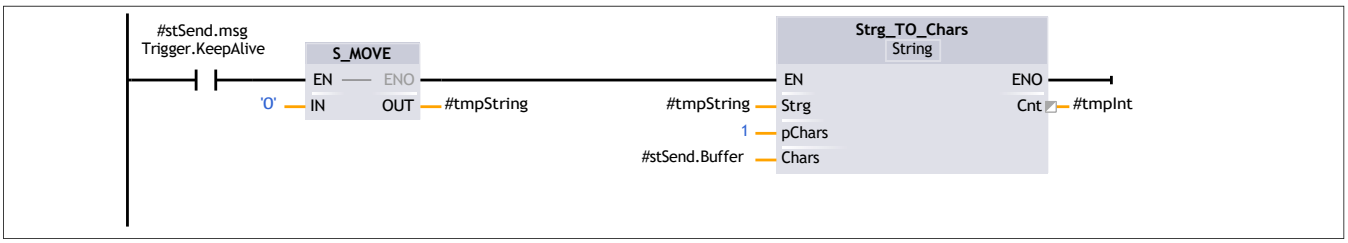
Network 11: Send message type



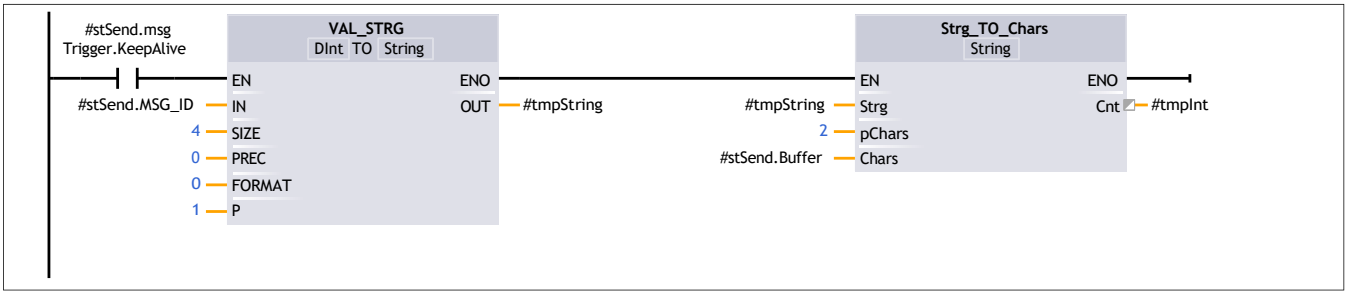
Network 12: Compose frame :: BYTE 0 :: STX (02h)



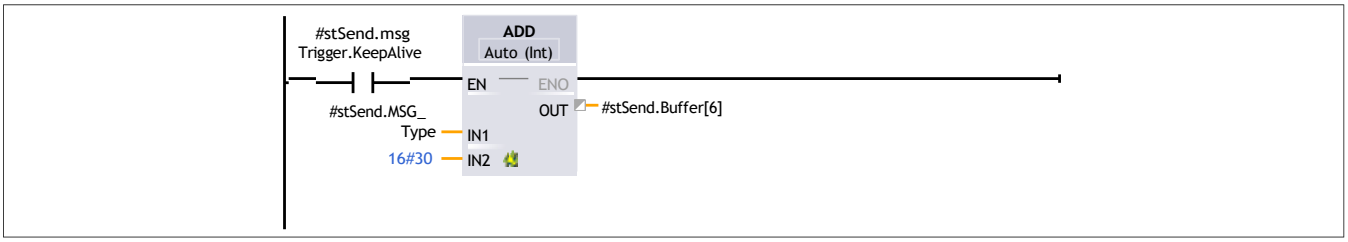
Network 13: Compose frame :: BYTE 1 :: 0



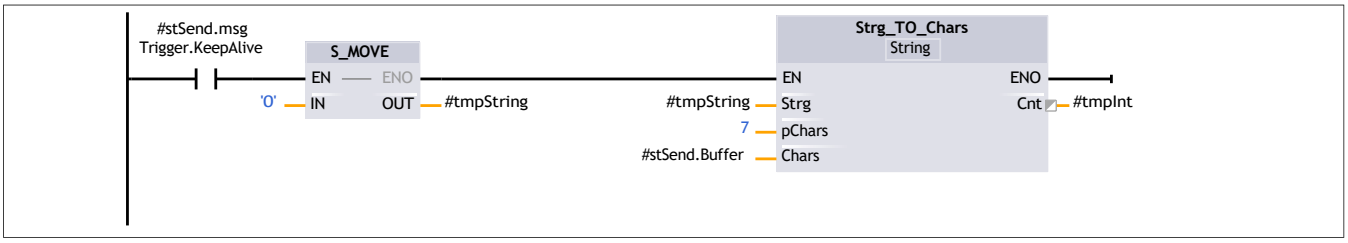
Network 14: Compose frame :: BYTE 2...5 :: Message ID



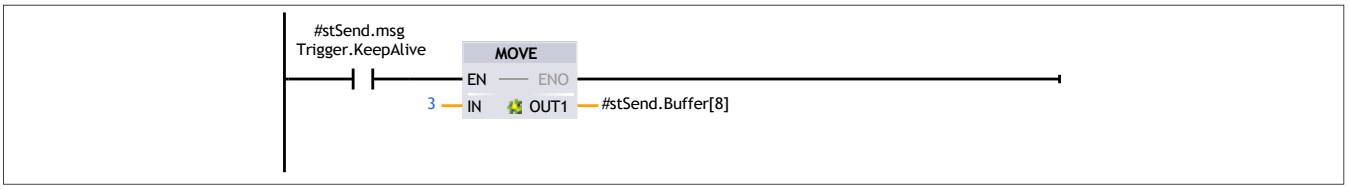
Network 15: Compose frame :: BYTE 6 :: message type



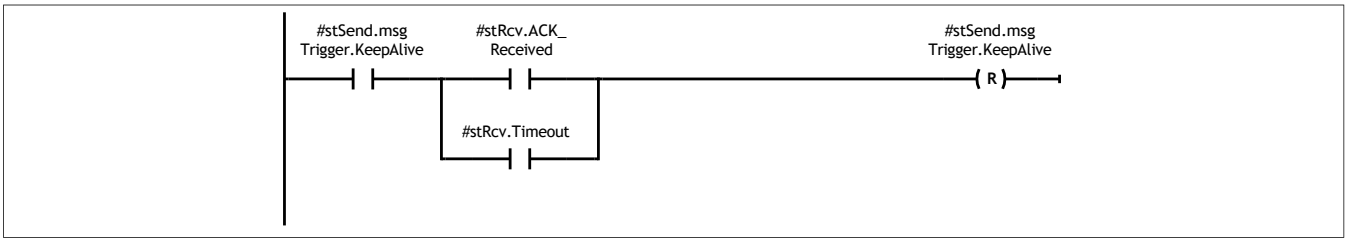
Network 16: Compose frame :: BYTE 7 :: 0



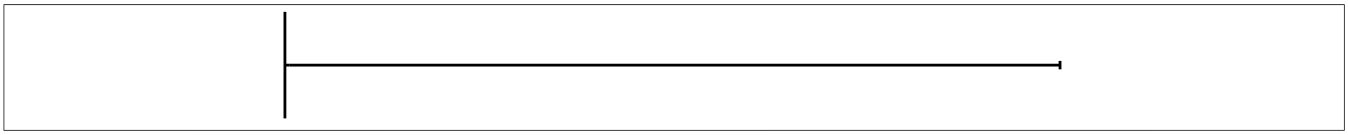
Network 17: Compose frame :: BYTE 8 :: ETX (3)



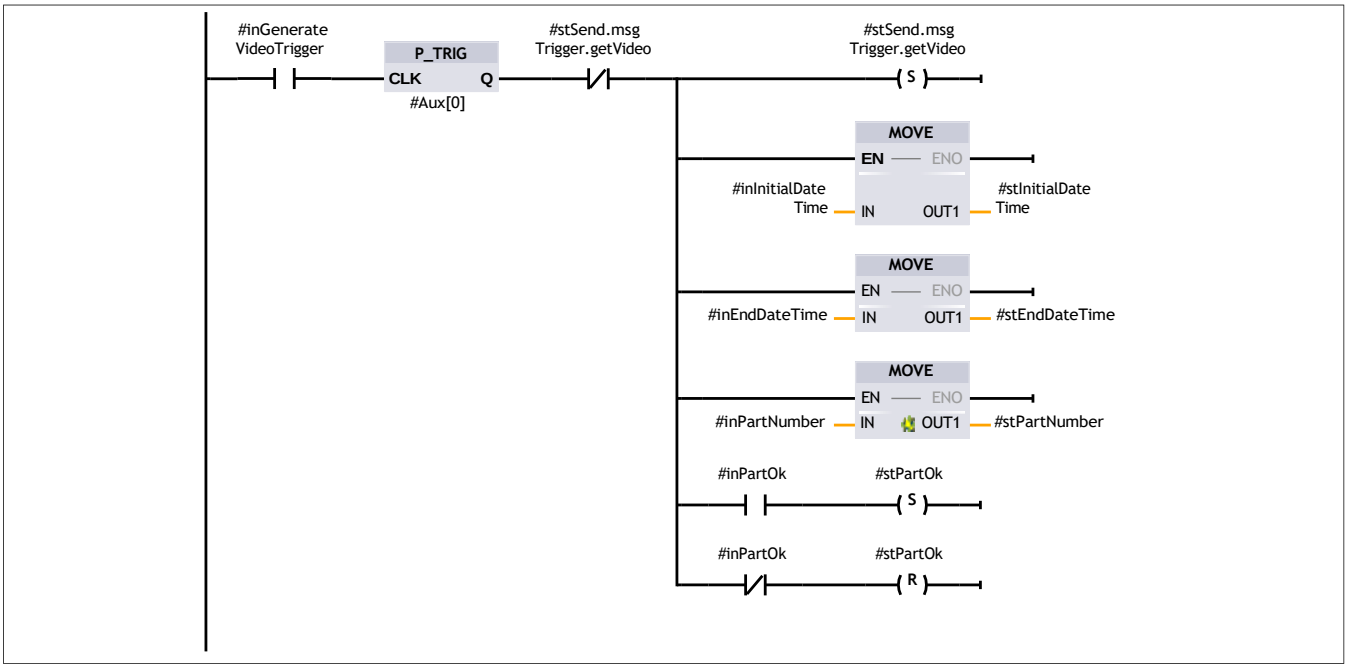
Network 18: Ack received



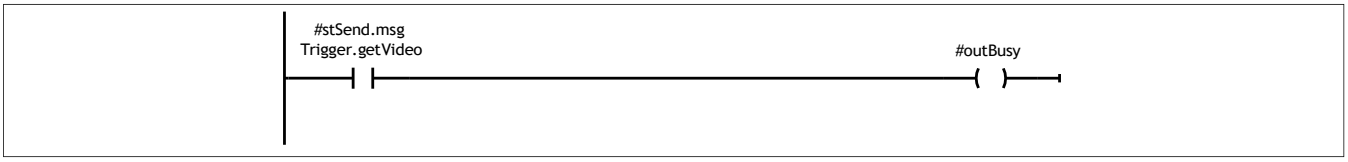
Network 19: ********** GET VIDEO ("1") **********



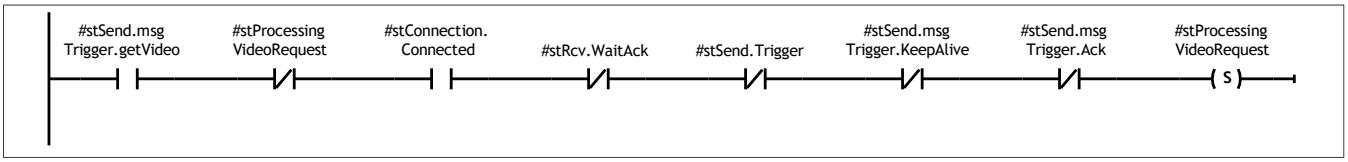
Network 20: Captura request data



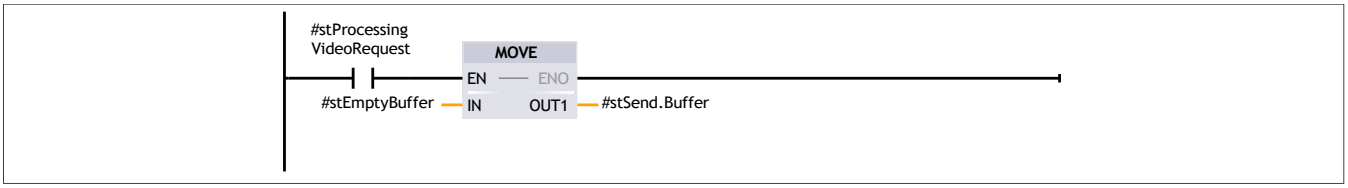
Network 21: out busy channel



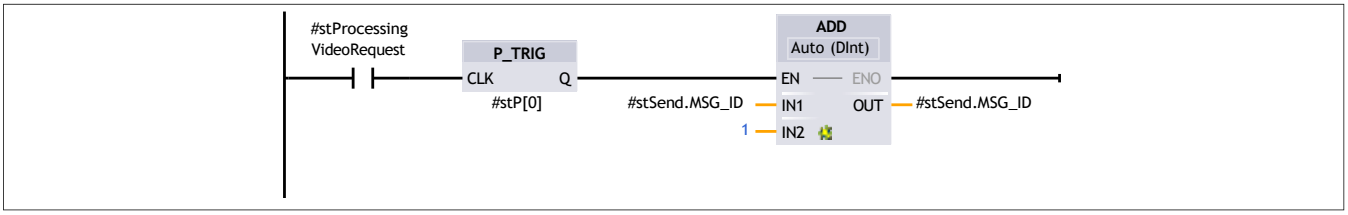
Network 22: Send trigger SET



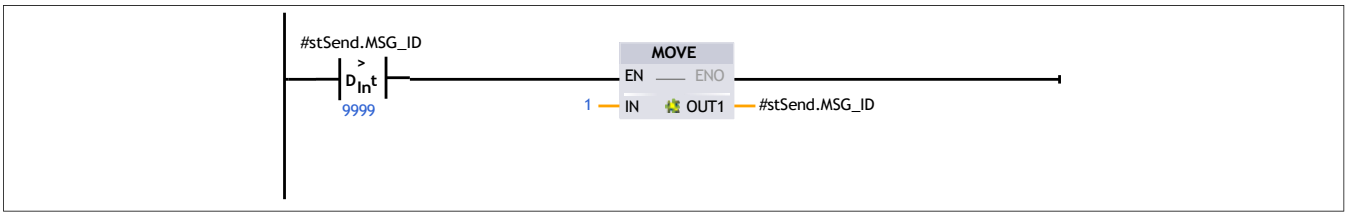
Network 23: Clear buffer



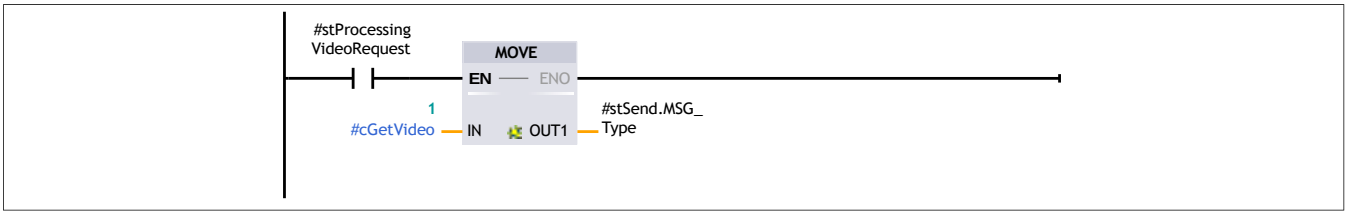
Network 24: Send message ID



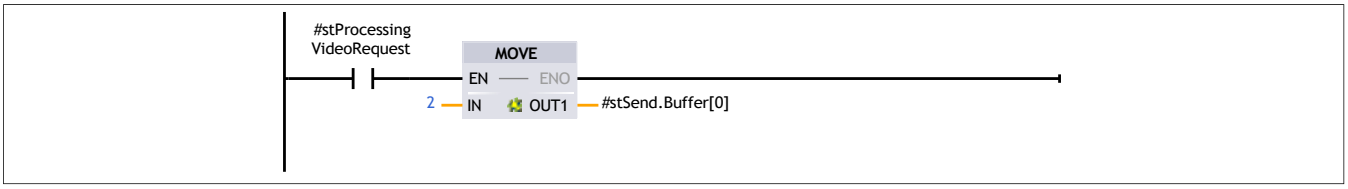
Network 25: Reset message ID



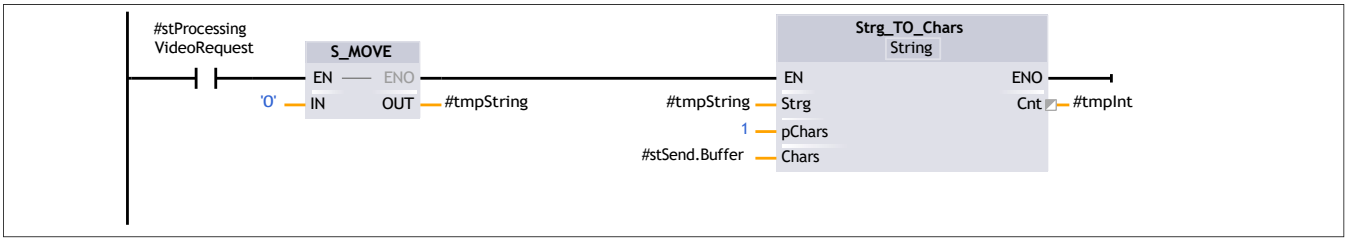
Network 26: Send message type



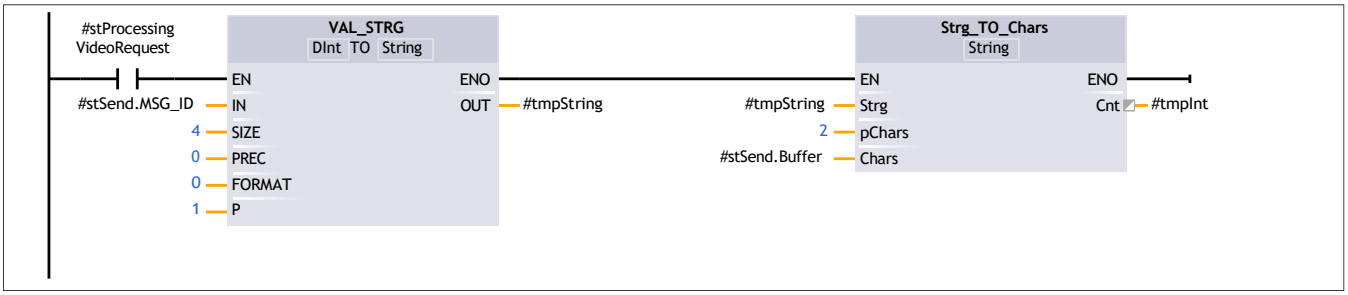
Network 27: Compose frame :: BYTE 0 :: STX (2)



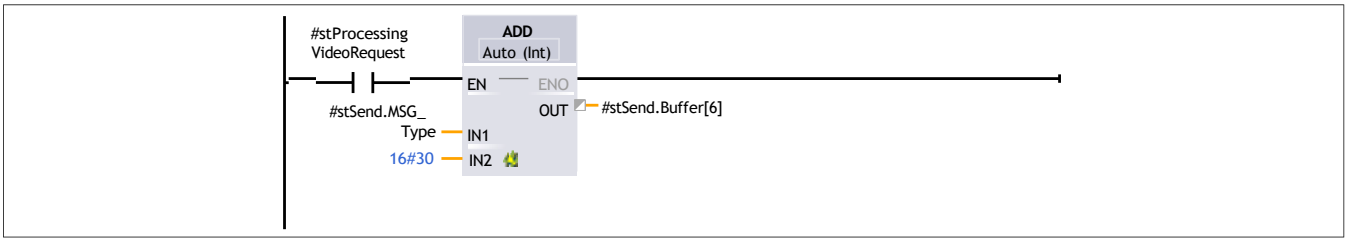
Network 28: Compose frame :: BYTE 1 :: 0



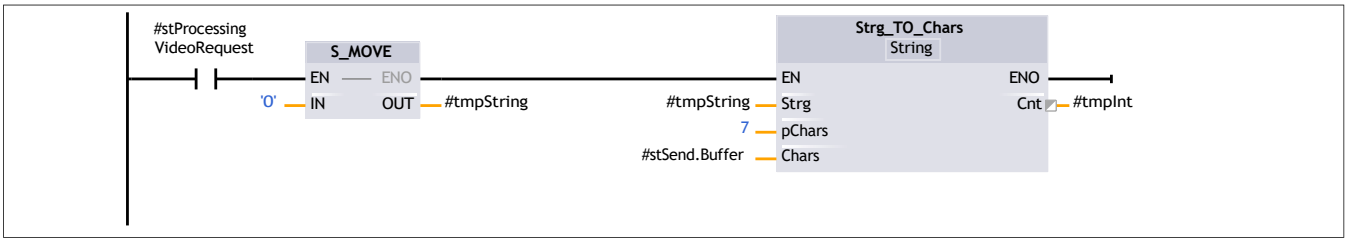
Network 29: Compose frame :: BYTE 2...5 :: Message ID



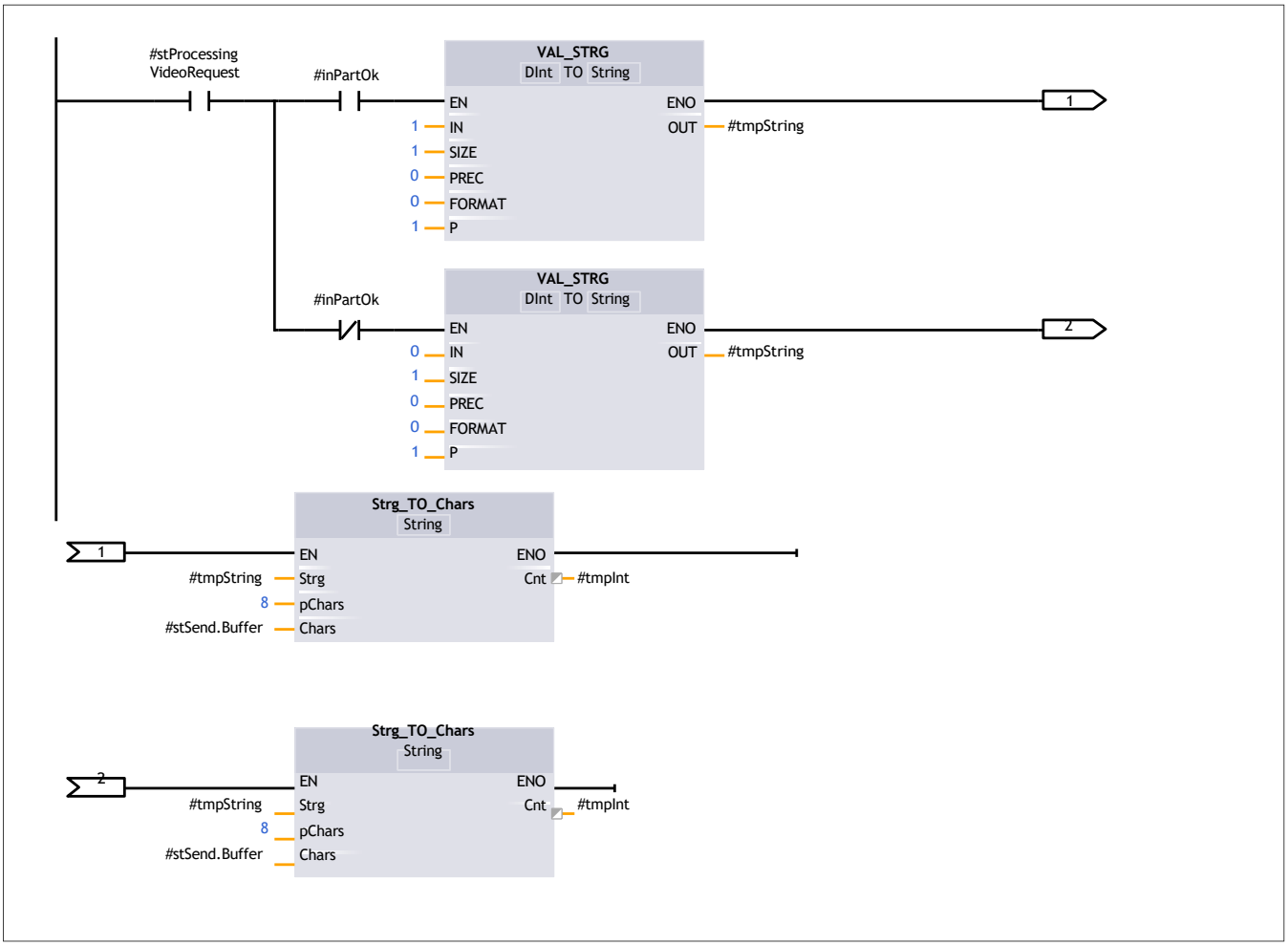
Network 30: Compose frame :: BYTE 6 :: Message type



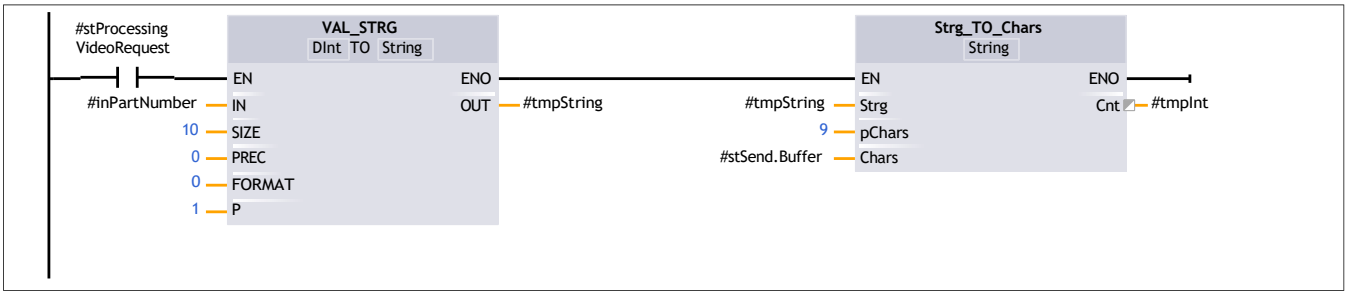
Network 31: Compose frame :: BYTE 7 :: 0



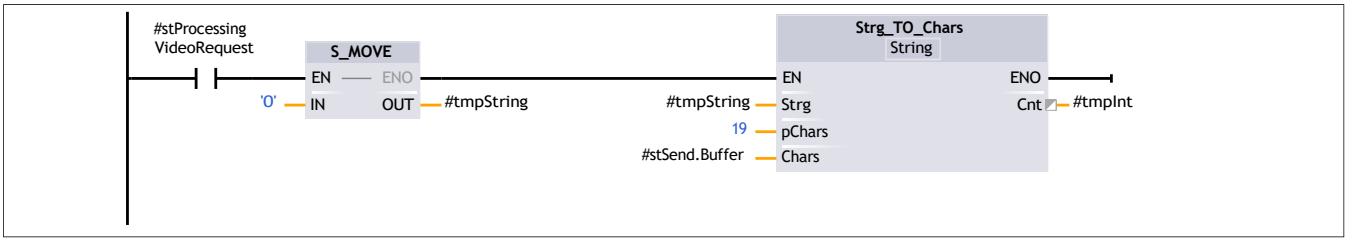
Network 32: Compose frame :: BYTE 8 :: Part status



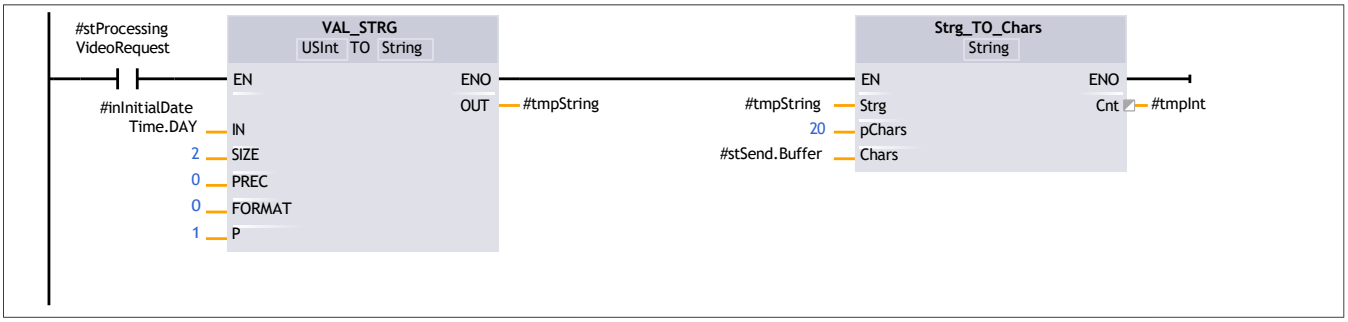
Network 33: Compose frame :: BYTE 9...18 :: Part counter



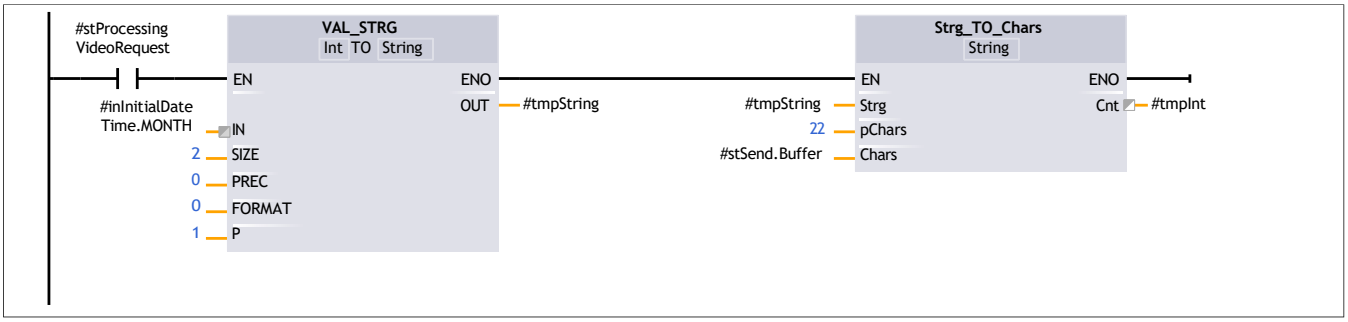
Network 34: Compose frame :: BYTE 19 :: 0



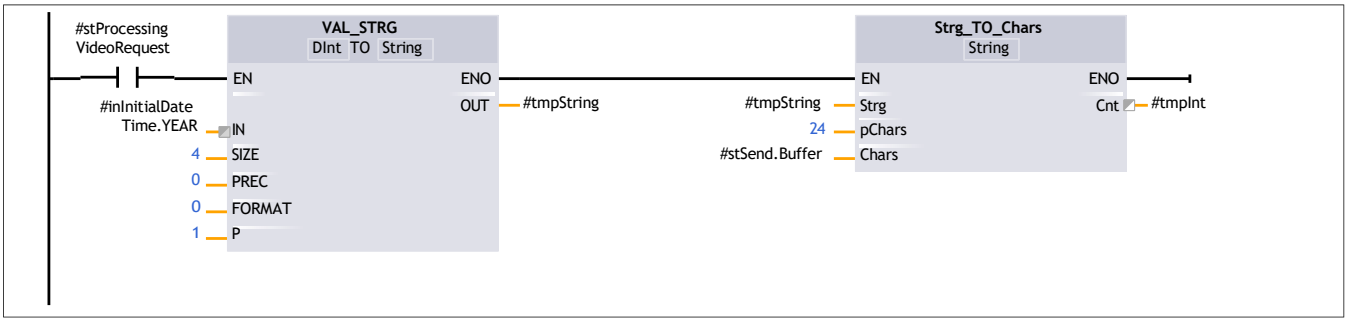
Network 35: Compose frame :: BYTE 20...21 :: Start day



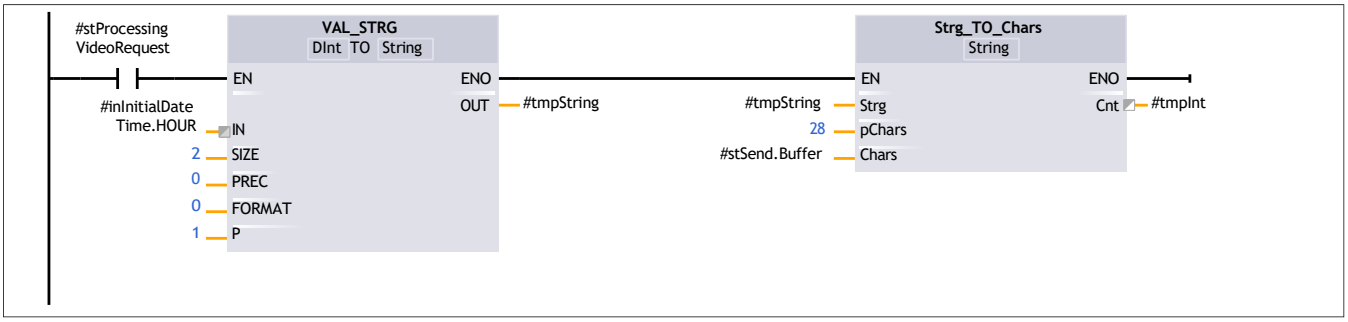
Network 36: Compose frame :: BYTE 22...23 :: Start month



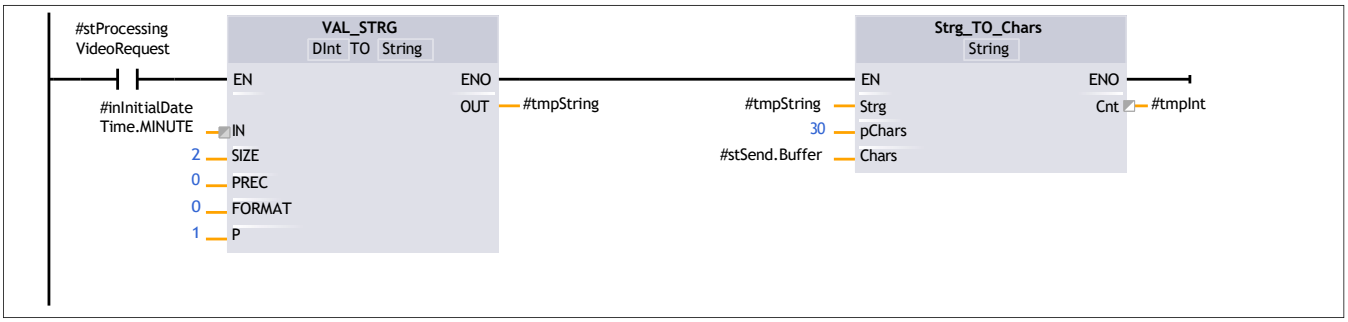
Network 37: Compose frame :: BYTE 24...27 :: Start year



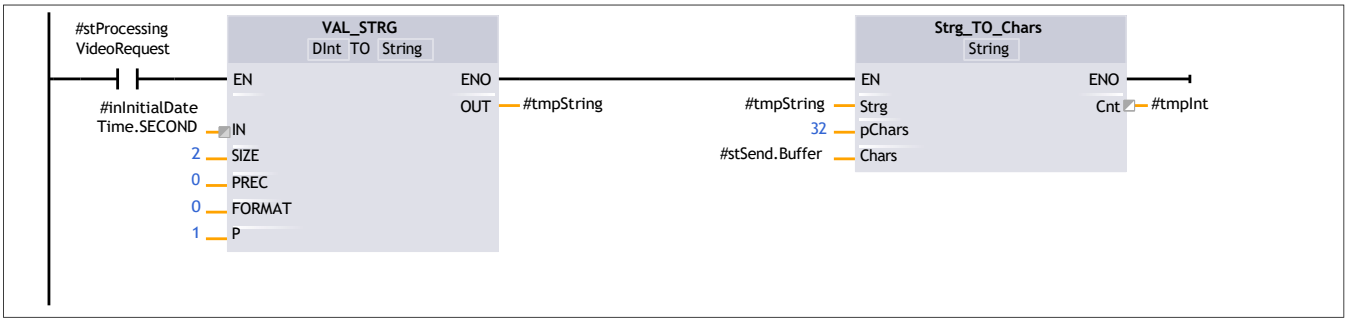
Network 38: Compose frame :: BYTE 28...29 :: Start hour



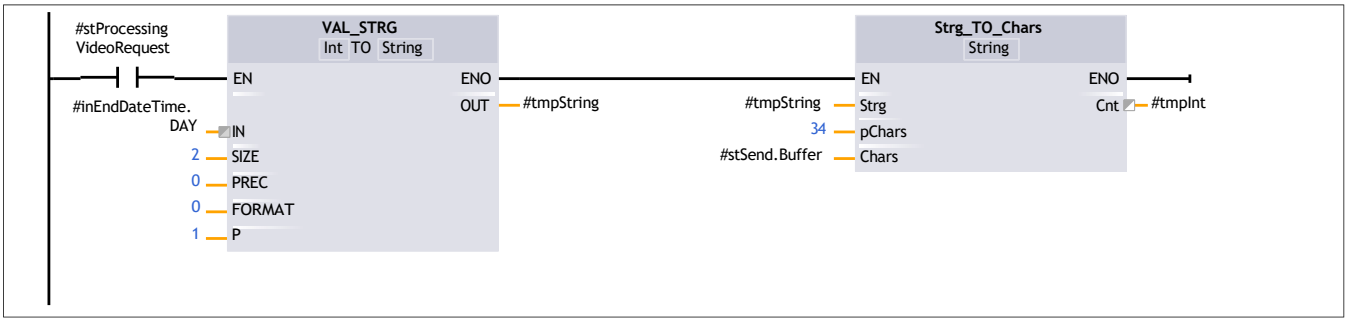
Network 39: Compose frame :: BYTE 30...31 :: Start minute



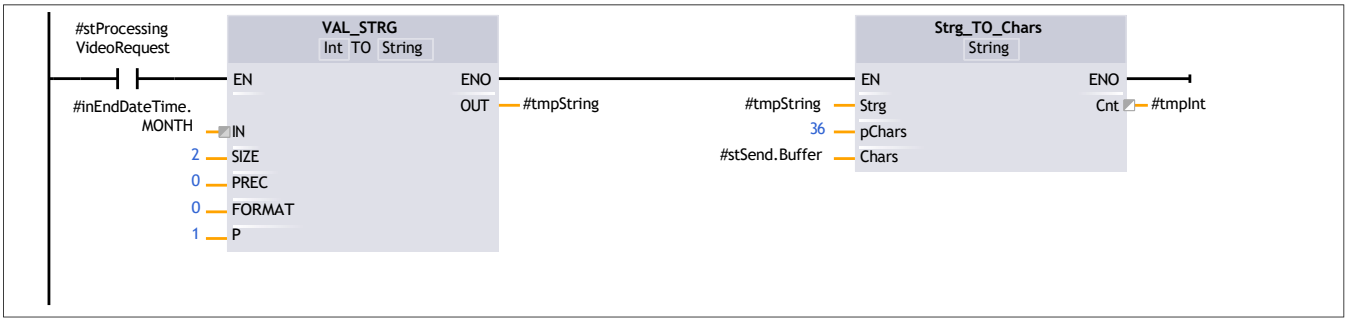
Network 40: Compose frame :: BYTE 32...33 :: Start second



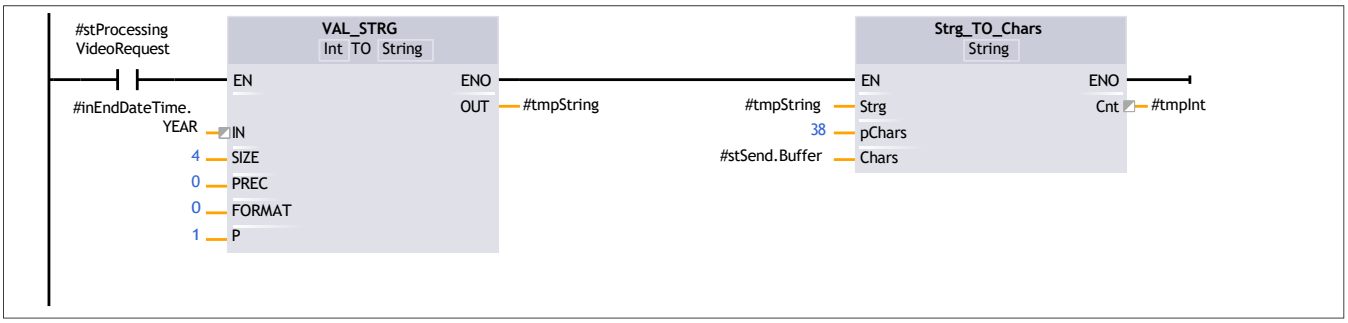
Network 41: Compose frame :: BYTE 34...35 :: End day



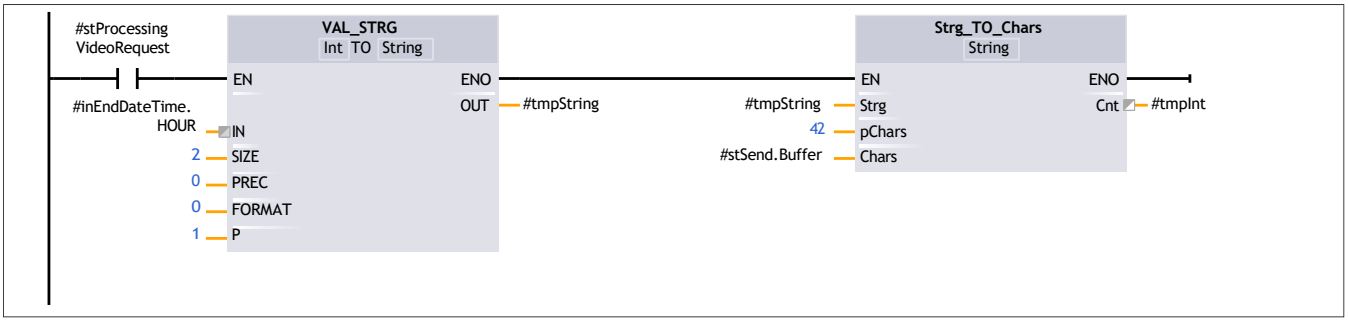
Network 42: Compose frame :: BYTE 36...37 :: End month



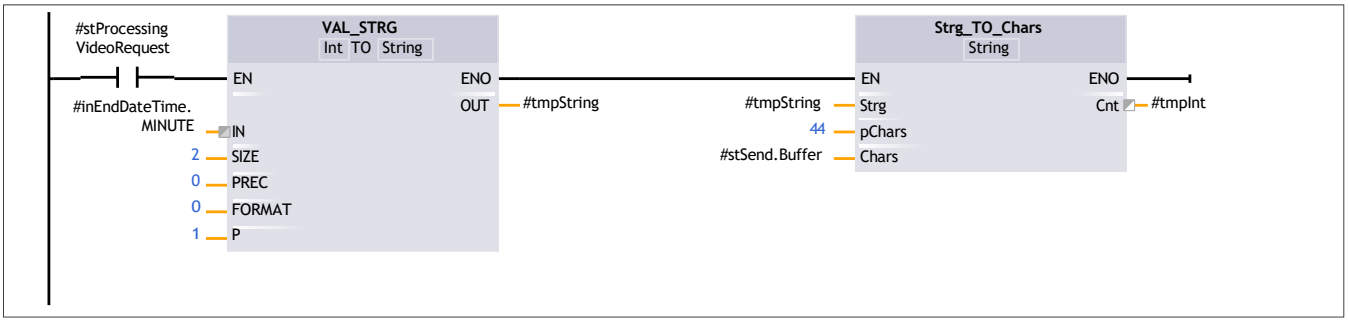
Network 43: Compose frame :: BYTE 38...41 :: End year



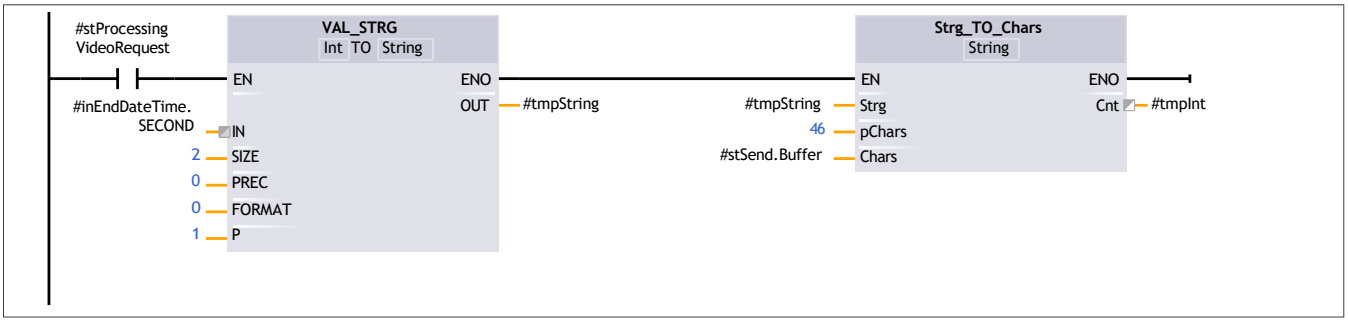
Network 44: Compose frame :: BYTE42..43 :: End hour



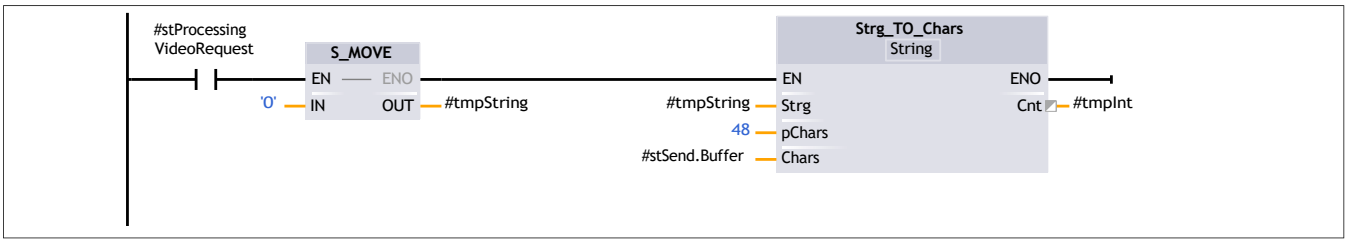
Network 45: Compose frame :: BYTE 44...45 :: End minute



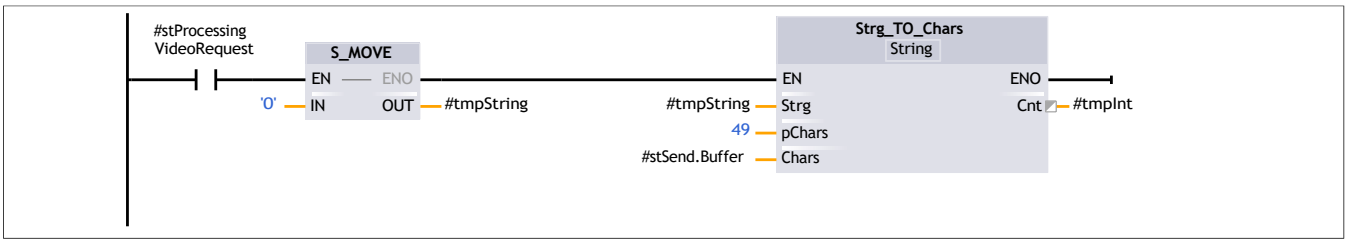
Network 46: Compose frame :: BYTE 46...47 :: End second



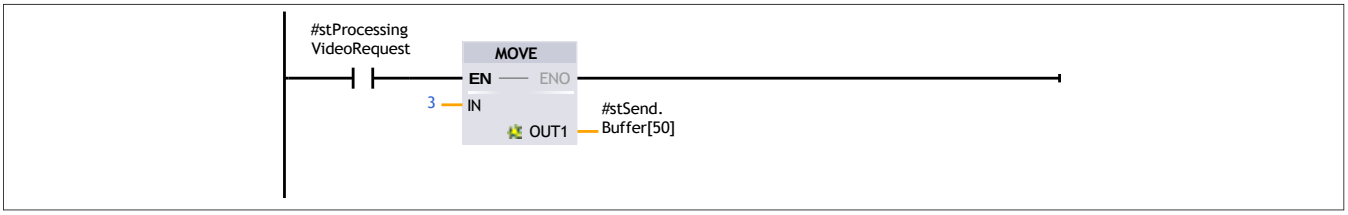
Network 47: Compose frame :: BYTE 48 :: 0



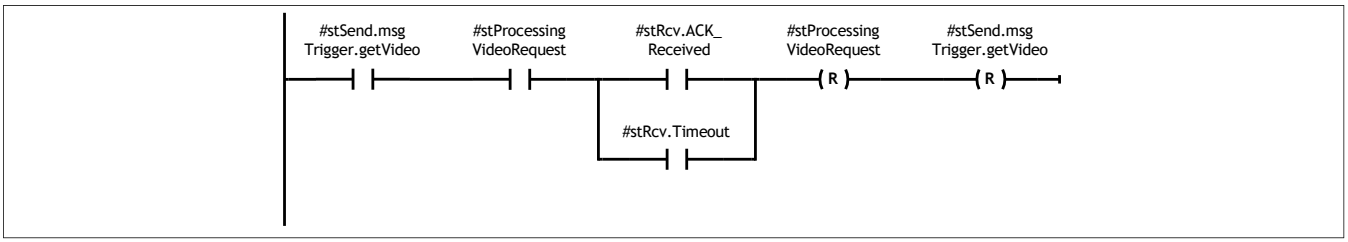
Network 48: Compose frame :: BYTE 49 :: 0



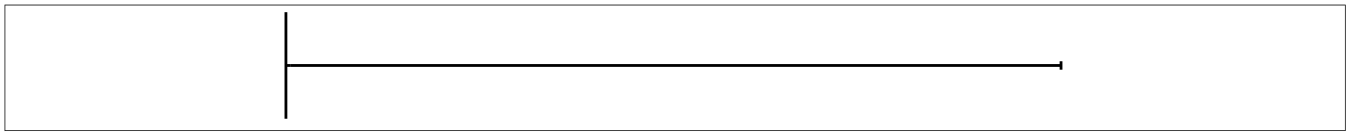
Network 49: Compose frame :: BYTE 50 :: ETX (3)



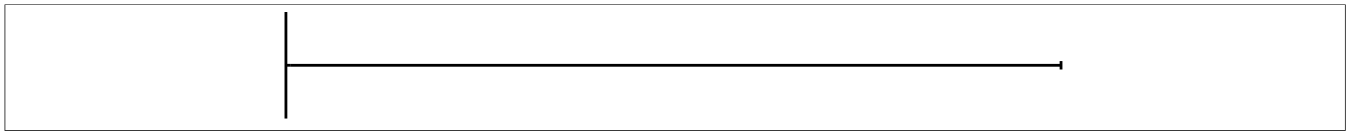
Network 50: Ack received



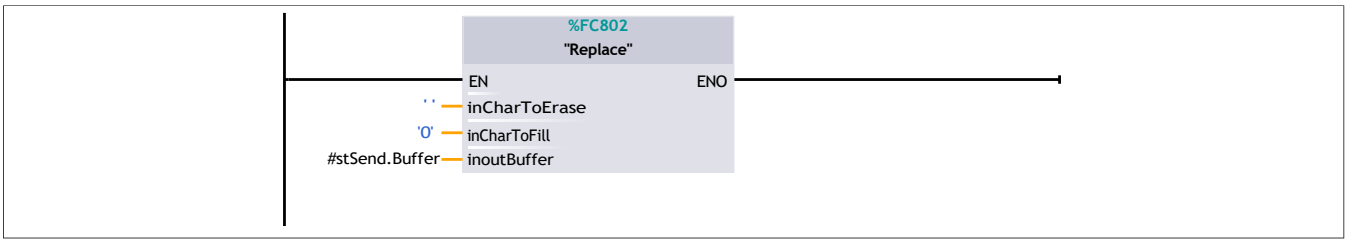
Network 51: ********** ACK ("9") **********



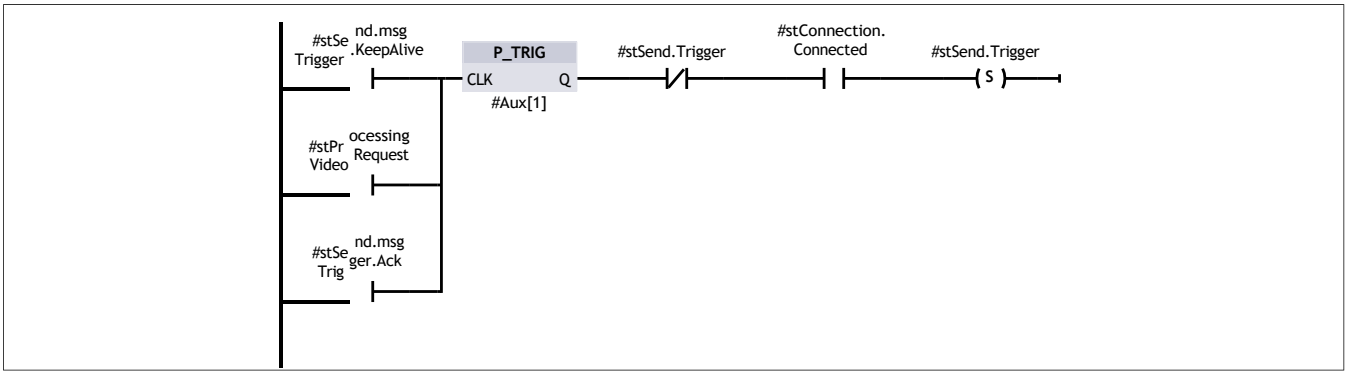
Network 52: ********** SEND DATA **********



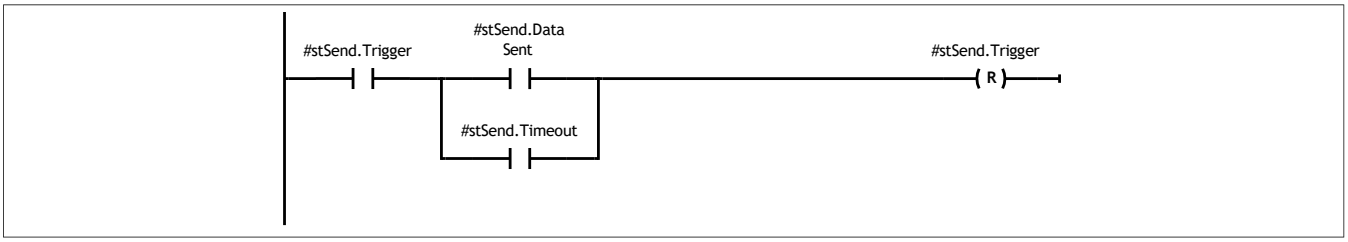
Network 53: Replace spaces by 0



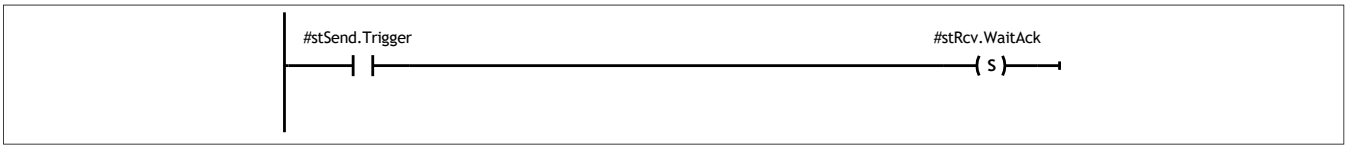
Network 54: Send trigger SET



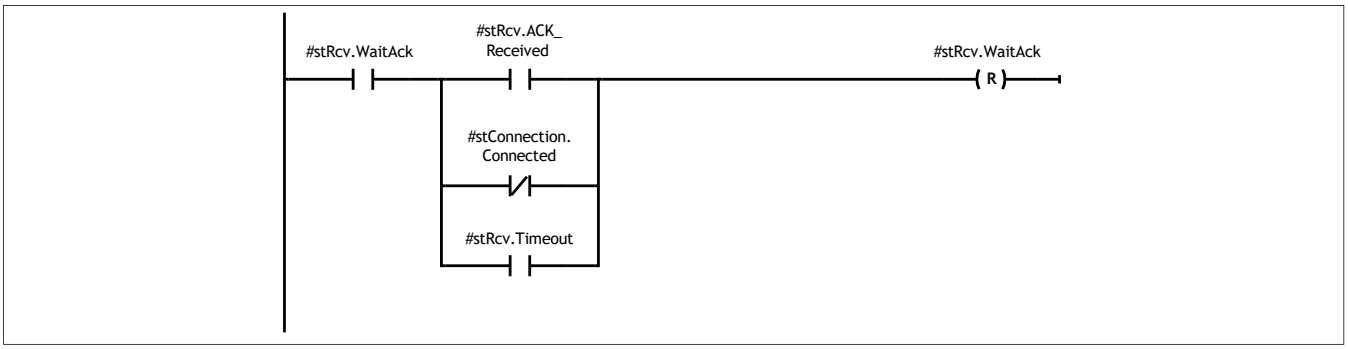
Network 55: Send trigger RESET



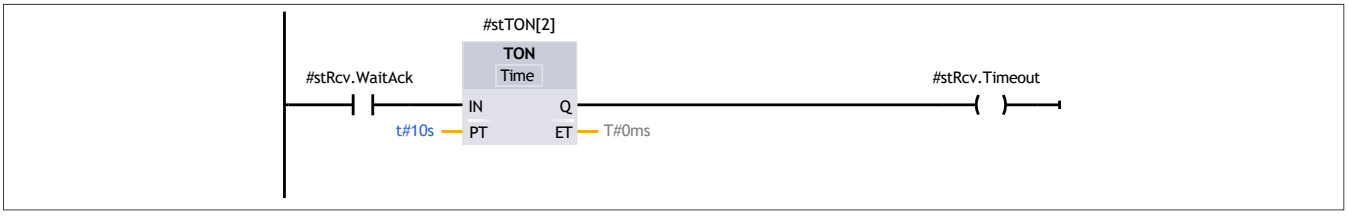
Network 56: Wait for ACK SET



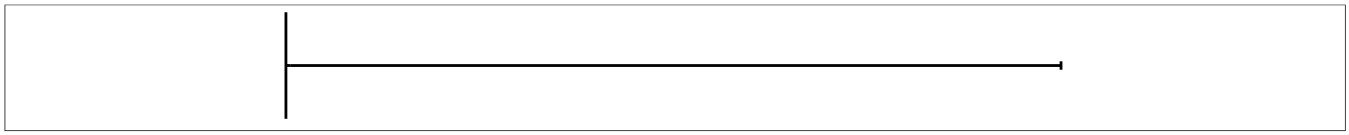
Network 57: Wait for ACK RESET



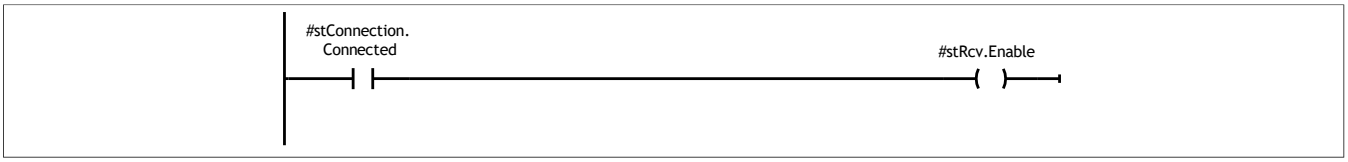
Network 58: Timeout waiting ACK



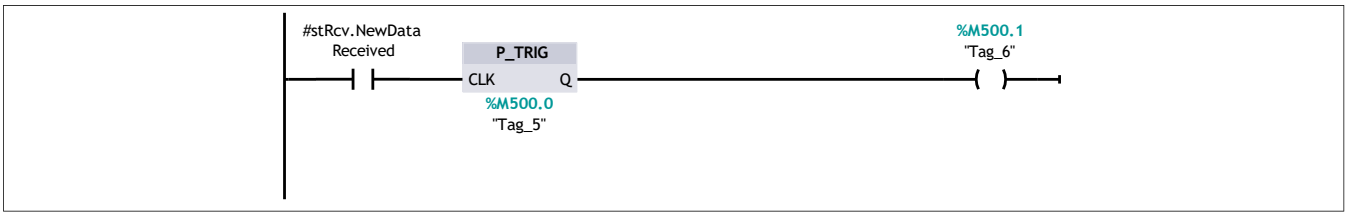
Network 59: ********** RECEIVE DATA **********



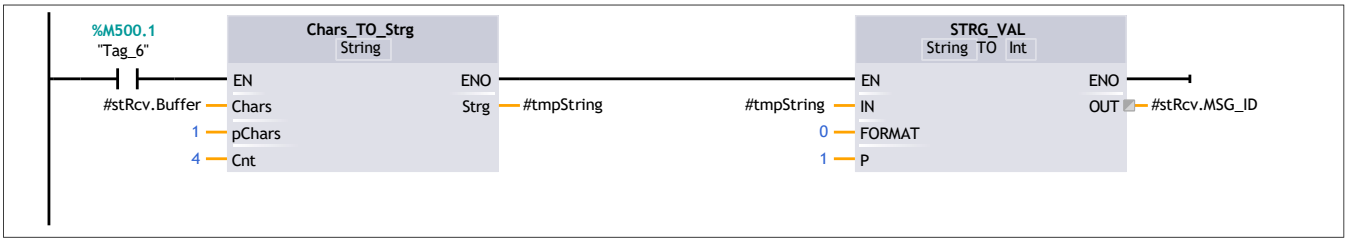
Network 60: Enable receive



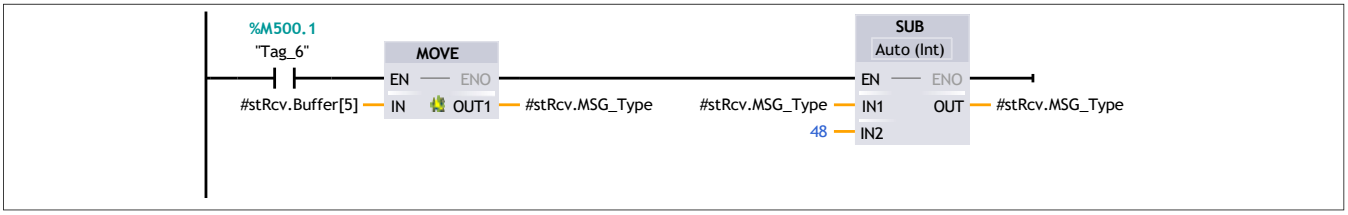
Network 61: Flange reception



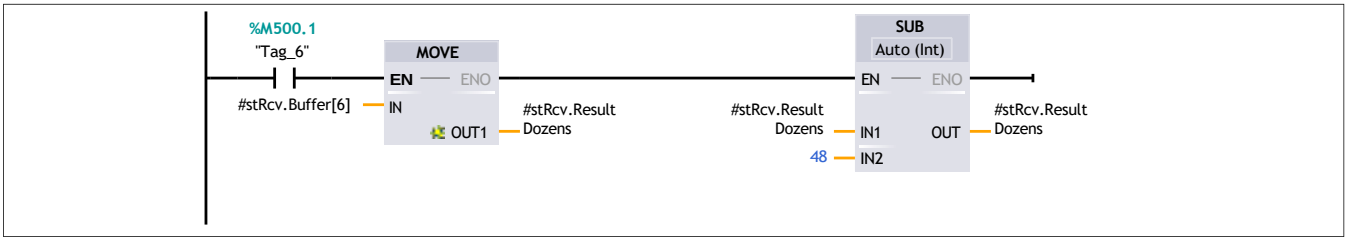
Network 62: New data received :: BYTE 2..5 :: Message ID



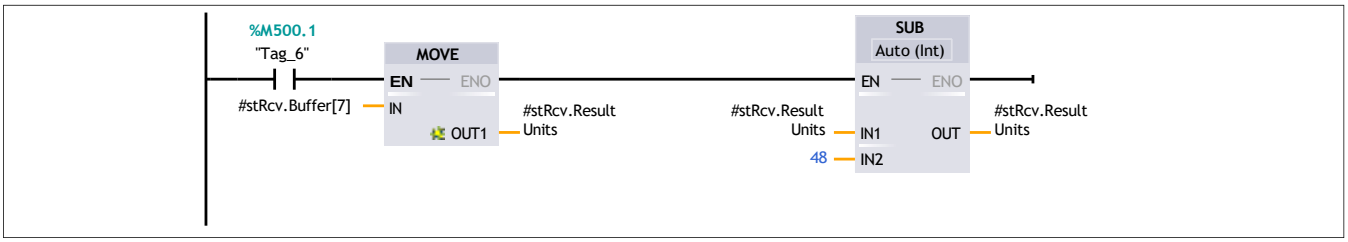
Network 63: New data received :: BYTE 6 :: Message Type



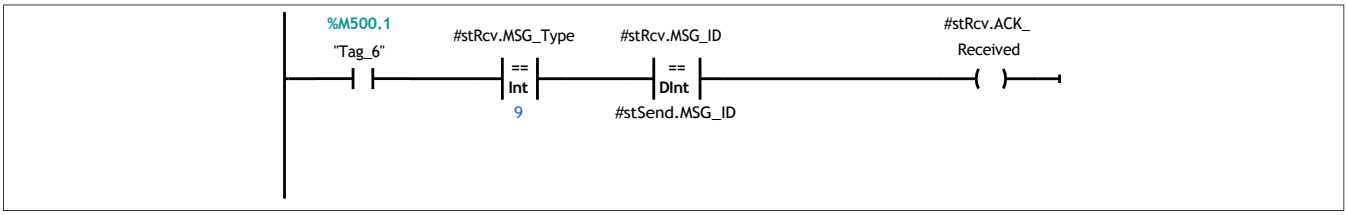
Network 64: New data received :: BYTE 7 :: Result Dozens



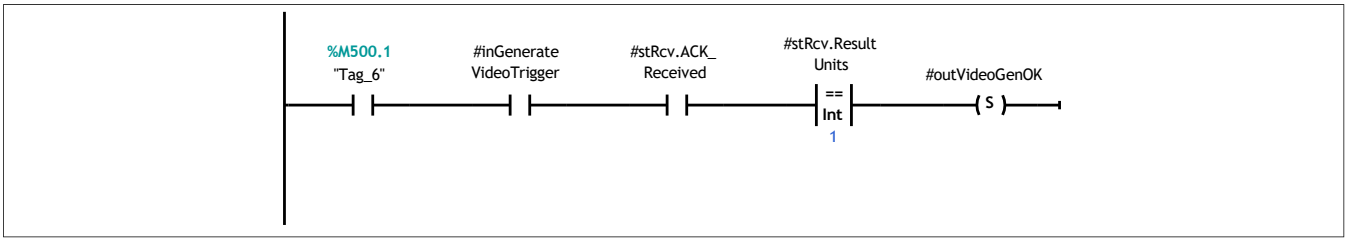
Network 65: New data received :: BYTE 8 :: Result Units



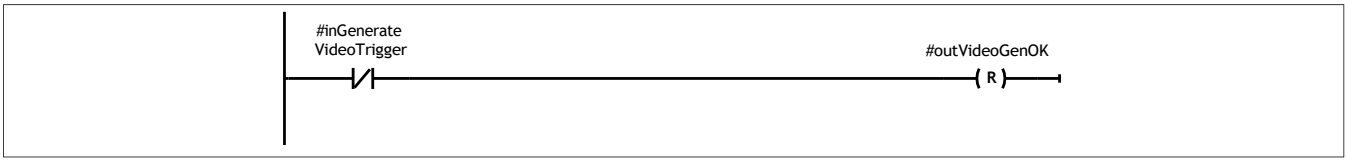
Network 66: Check if received data is ACK of last sent message



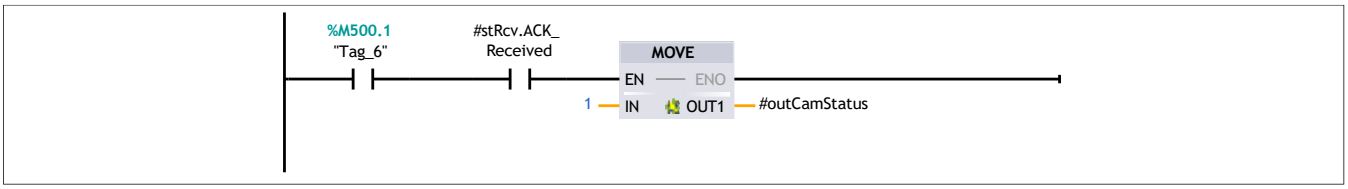
Network 67: Video generated successfully



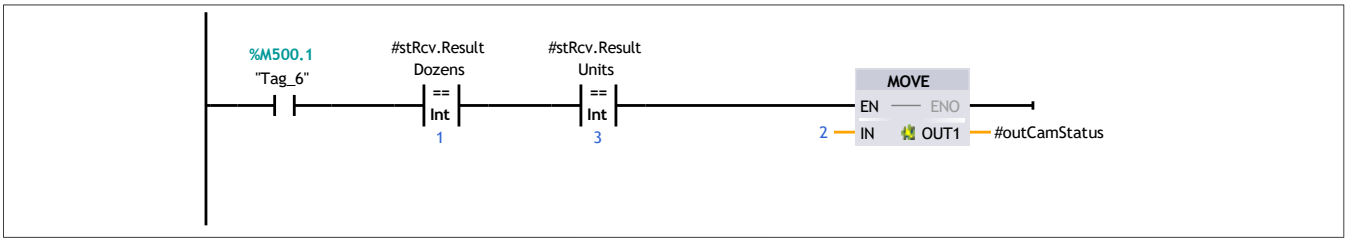
Network 68: Reset check video ok



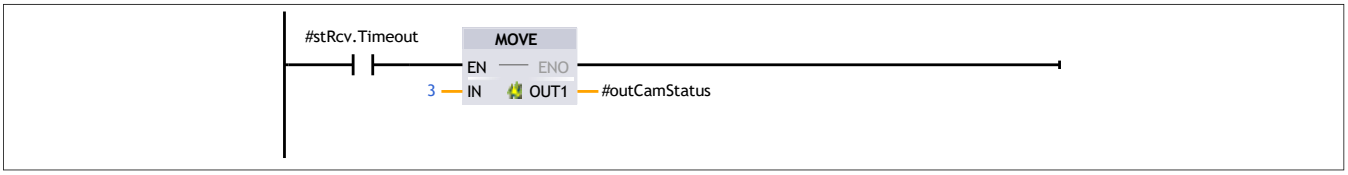
Network 69: Cam status := 1 (Comunication OK)



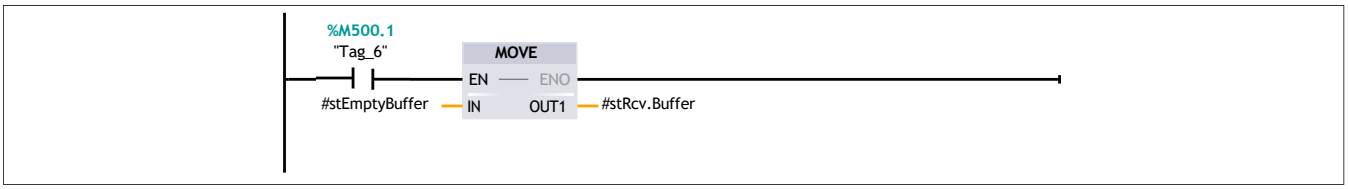
Network 70: Cam status := 2 (Wrong frame, none of the possible cases selected)



Network 71: Cam status := 3 (No response from app)



Network 72: Clear buffer received



Program blocks / Streaming MNG (800)

Streaming MNG_IDB [DB2]

Streaming MNG_IDB Properties					
General					
Name	Streaming MNG_IDB	**Number**	2	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain
▼ Input			
inGenerateVideoTrigger	Bool	false	False
inInitialDateTime	DTL	DTL#2023-10-26-16:45:00	False
inEndDateTime	DTL	DTL#2023-10-26-16:50:00	False
inPartNumber	DInt	0	False
inPartOk	Bool	false	False
inConnection	Struct		False
▼ Output			
outVideoGenOK	Bool	false	False
outBusy	Bool	false	False
outCamStatus	Int	0	False
InOut			
▼ Static			
stSocket_IDB	"Socket"		False
stConnection	Struct		False
stSend	Struct		False
stRcv	Struct		False
Aux	Array[0..20] of Bool		False
stInitialDateTime	DTL	DTL#1970-01-01-00:00:00	False
stEndDateTime	DTL	DTL#1970-01-01-00:00:00	False
stPartNumber	DInt	0	False
stPartOk	Bool	false	False
stProcessingVideoRequest	Bool	false	False
stP	Array[0..10] of Bool		False
stTON	Array[0..10] of TON_TIME		False
stAuxDint	Array[0..10] of DInt		False
stAuxCharArray	Array[0..10] of Char		False
stEmptyBuffer	Array[0..50] of Byte		False

Program blocks / Streaming MNG (800) / Socket (900)

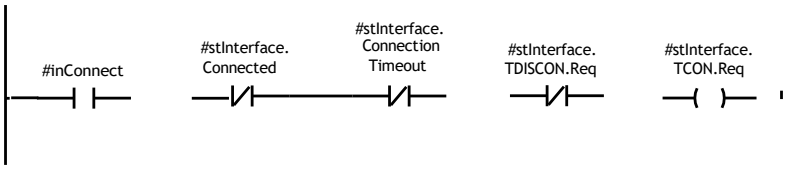
Socket [FB900]

Socket Properties					
General					
Name	Socket	**Number**	900	**Type**	FB
Language	LAD	**Numbering**	Manual		
Information					
Title	Socket management	**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

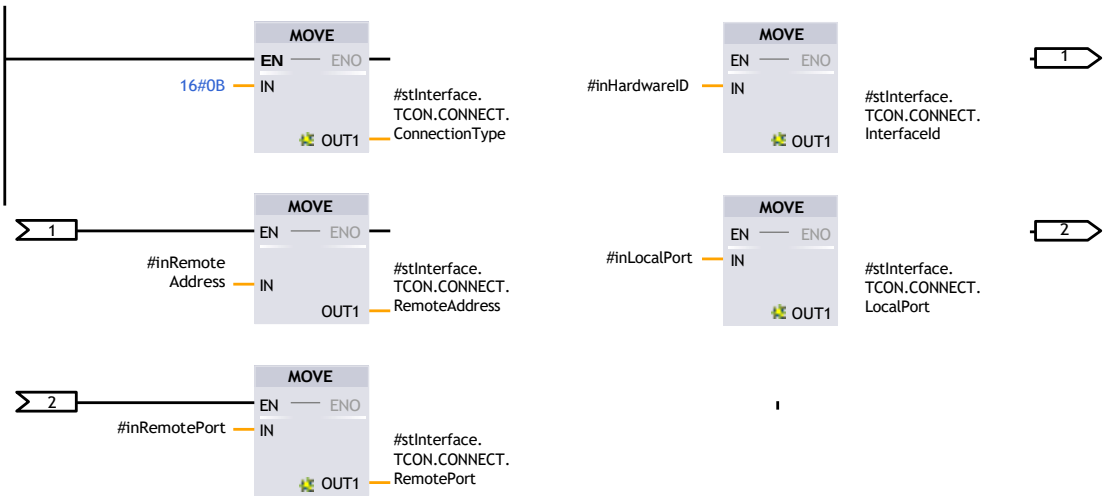
Name	Data type	Default value	Retain
▼ Input			
inConnect	Bool	false	Non-retain
inSendTrigger	Bool	false	Non-retain
inEnableRcv	Bool	false	Non-retain
inHardwareID	HW_ANY	0	Non-retain
inRemoteAddress	IP_V4		Non-retain
inLocalPort	Int	0	Non-retain
inRemotePort	Int	0	Non-retain
inID	CONN_OUC	16#0	Non-retain
inManualDisconnection	Bool	false	Non-retain
▼ Output			
outConnected	Bool	false	Non-retain
outConnectionTimeout	Bool	false	Non-retain
outDataSent	Bool	false	Non-retain
outSendTimeout	Bool	false	Non-retain
outNewDataReceived	Bool	false	Non-retain
▼ InOut			
inoutSendBuffer	Array[0..50] of Byte		
inoutRcvBuffer	Array[0..50] of Byte		
▼ Static			
stTCON	TCON		
stTDISCON	TDISCON		
stTRCV	TRCV		
stTSEND	TSEND		
stInterface	Struct		Non-retain
stTON_ConTimeout	TON_TIME		Non-retain
stTP_ConTimeout	TP_TIME		Non-retain
stTON_SendTimeout	TON_TIME		Non-retain
stTON_DisconTimeout	TON_TIME		Non-retain
Temp			
Constant			

Network 1: ******** CONNECTION **********

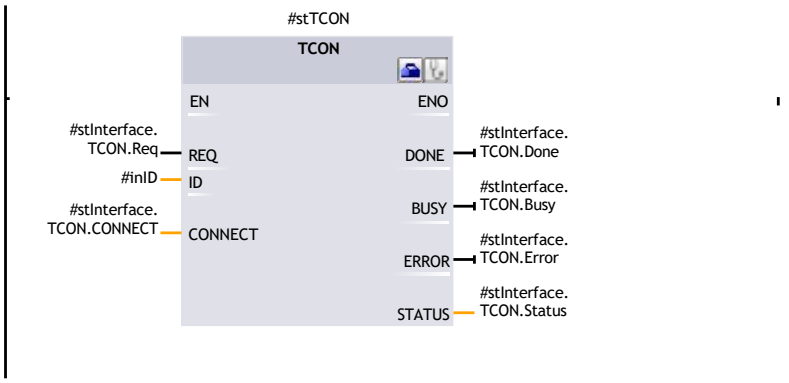
Network 2: Connection request



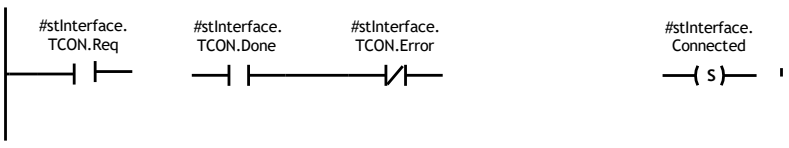
Network 3: Connection settings



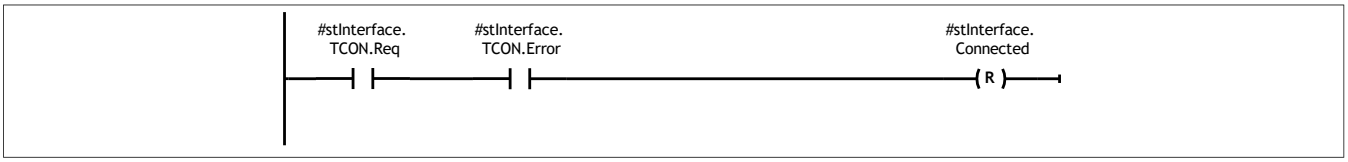
Network 4: Connection block



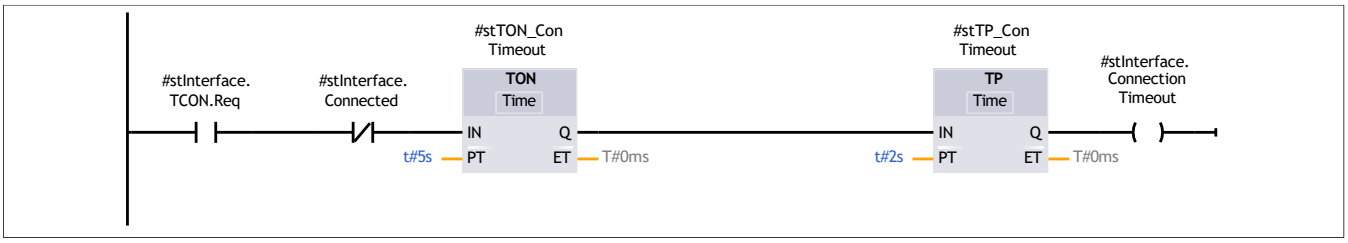
Network 5: Connected



Network 6: Connection error

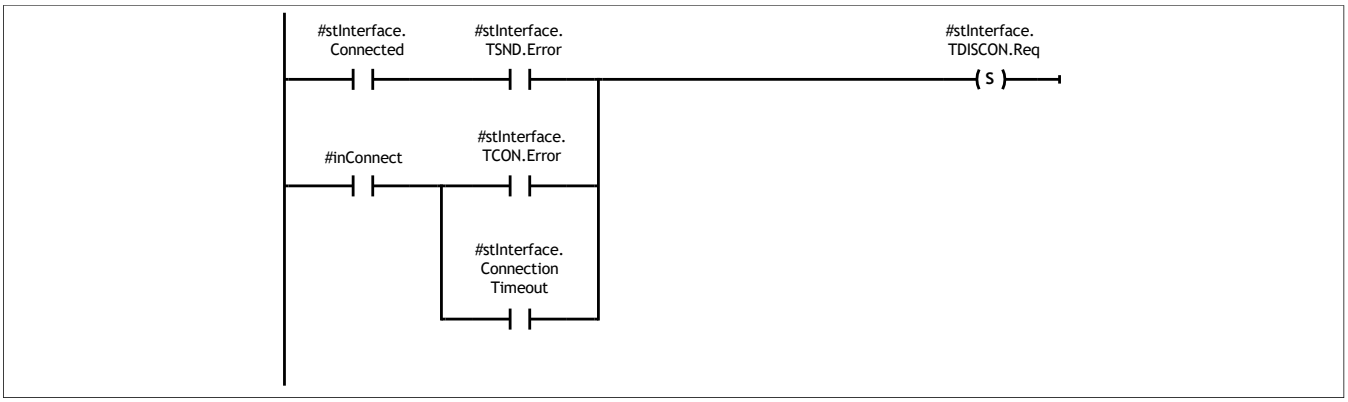


Network 7: Connection timeout

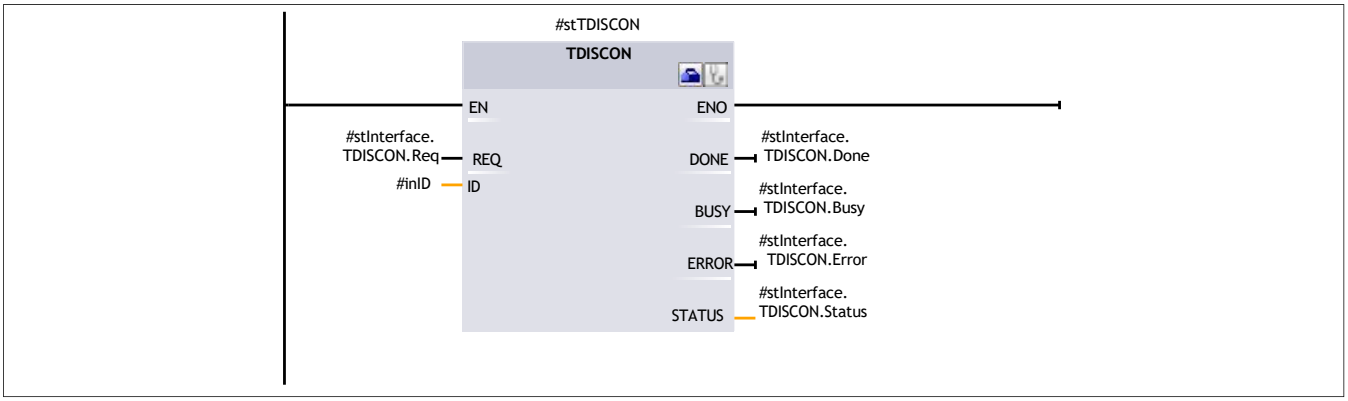


Network 8: ********** DISCONNECTION **********

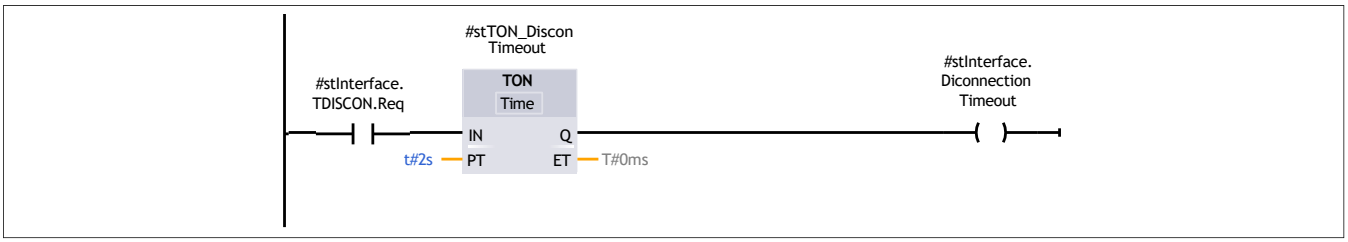
Network 9: Discconection request



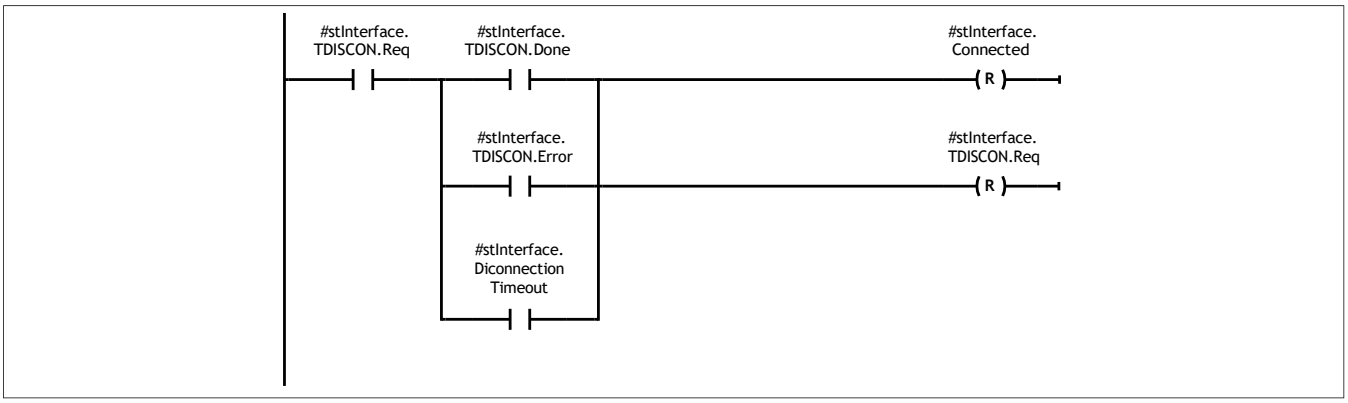
Network 10: Disconnection block



Network 11: Disconnection timeout

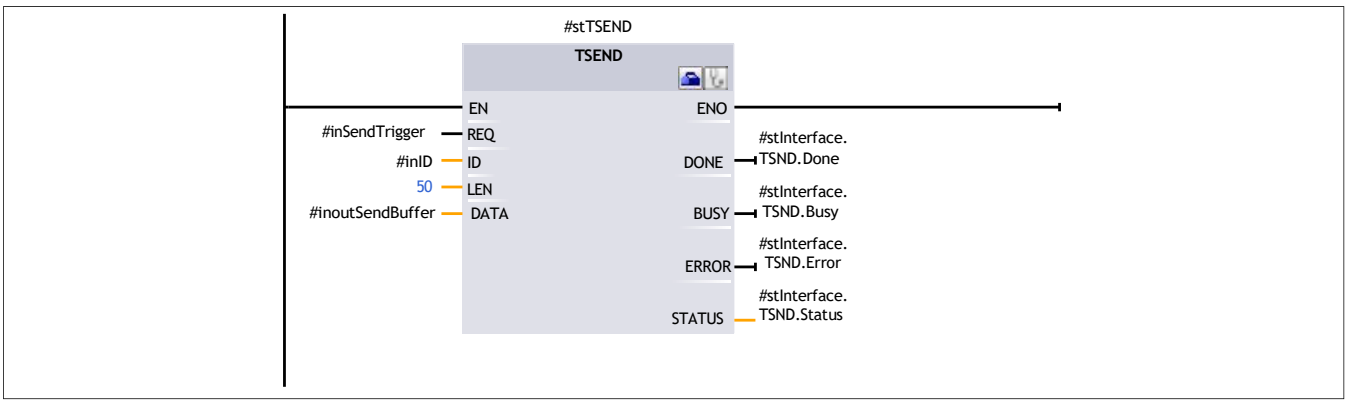


Network 12: Disconnected



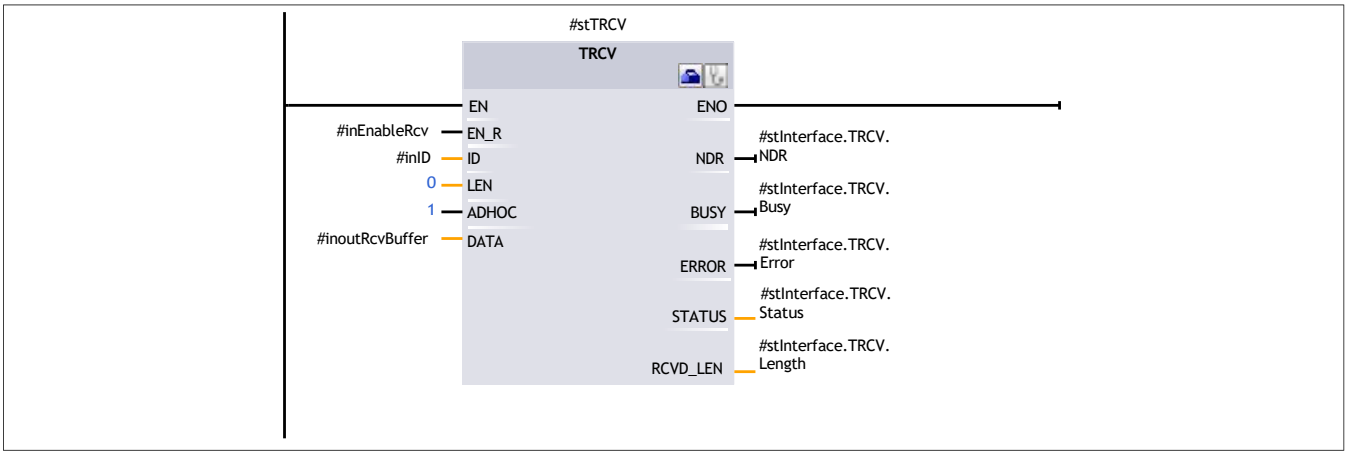
Network 13: ********** SEND **********

Network 14: Send block

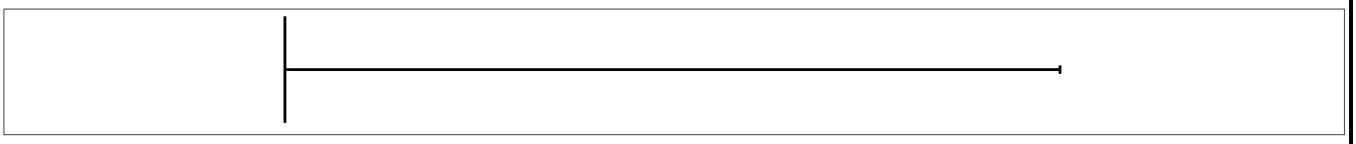


Network 15: ********** RECEIVE **********

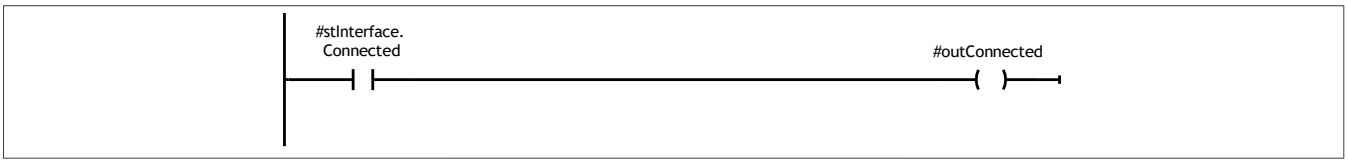
Network 16: Recieve block



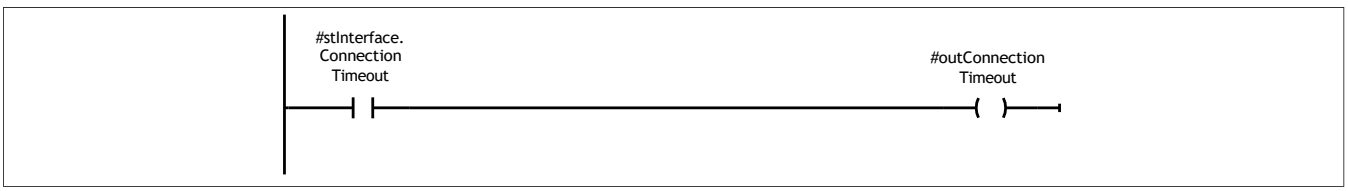
Network 17: ********** OUTPUTS **********



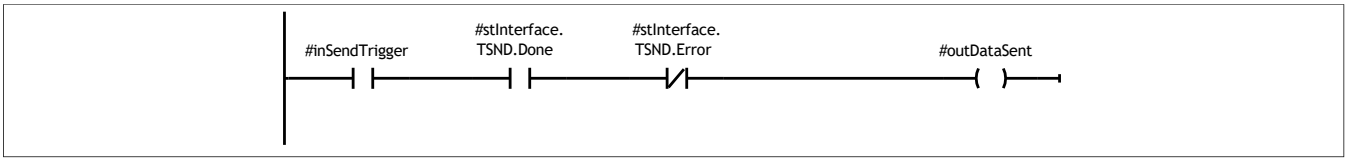
Network 18: Connected



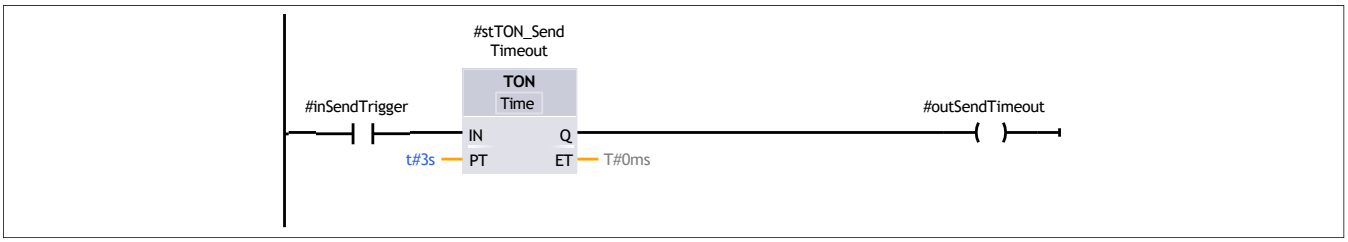
Network 19: Connection timeout



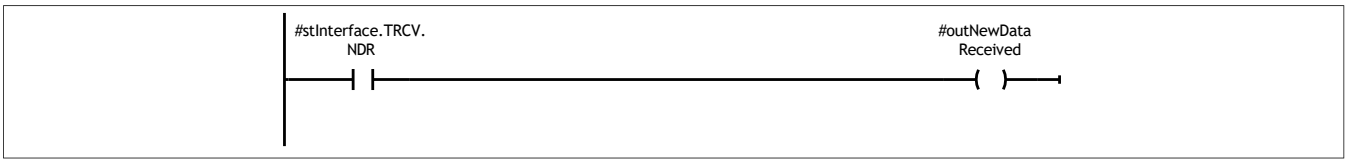
Network 20: Data sent



Network 21: Send timeout



Network 22: New data received



Program blocks / Streaming MNG (800) / Socket (900)

Socket_IDB [DB900]

Socket_IDB Properties					
General					
Name	Socket_IDB	**Number**	900	**Type**	DB
Language	DB	**Numbering**	Manual		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain
▼ Input			
inConnect	Bool	false	False
inSendTrigger	Bool	false	False
inEnableRcv	Bool	false	False
inHardwareID	HW_ANY	0	False
inRemoteAddress	IP_V4		False
inLocalPort	Int	0	False
inRemotePort	Int	0	False
inID	CONN_OUC	16#0	False
inManualDisconnection	Bool	false	False
▼ Output			
outConnected	Bool	false	False
outConnectionTimeout	Bool	false	False
outDataSent	Bool	false	False
outSendTimeout	Bool	false	False
outNewDataReceived	Bool	false	False
▼ InOut			
inoutSendBuffer	Array[0..50] of Byte		False
inoutRcvBuffer	Array[0..50] of Byte		False
▼ Static			
stTCON	TCON		False
stTDISCON	TDISCON		False
stTRCV	TRCV		False
stTSEND	TSEND		False
stInterface	Struct		False
stTON_ConTimeout	TON_TIME		False
stTP_ConTimeout	TP_TIME		False
stTON_SendTimeout	TON_TIME		False
stTON_DisconTimeout	TON_TIME		False

DOCUMENTO Nº2: PRESUPUESTO

Índice

1. Introducción

El presente documento tiene como objetivo la descripción detallada de todos los elementos que se han requerido para la realización del proyecto.

El coste total del proyecto de divide en los siguientes 3 apartados:

- **Elementos hardware:** Formados por los aparatos electrónicos físicos necesarios para llevar a cabo el proyecto.
- **Elementos software:** Formados por los programas necesarios para desarrollar el proyecto y sus licencias.
- **Mano de obra.**

2. Desarrollo del presupuesto

Elementos hardware.

Nº	DESCRIPCIÓN	IMPORTE		
		Precio (€)	Unidad (n)	Total (€)
1	ET 200SP Open Controller CPU 1515SP PC	2.815,79 €	1	2.815,79 €
2	SIMATIC Field PG M6	7.509,74 €	1	
2.1	SIMATIC Field PG M6 Programador	28,17 €	1	28,17 €
3	IR EYEBALL NETWORK CAMERA H.265+	130,00 €	1	130,00 €
4	SSD SanDisk Extreme Portable 1TB	104,99 €	1	104,99 €
				3.078,95 €

Tabla 1: Presupuesto elementos Hardware.

Los precios mostrados en la Tabla 1 se han obtenido de las siguientes fuentes:

- https://www.securame.com/domo-ip-dahua-hdw2431tass2-4mp-ir30m-28mm-h265-poe-sd-wdr-audio-p-5052.html?gclid=Cj0KCQjwlN6wBhCcARIsAKZvD5j-k_Bvlz78RhVOufvMyaZyp7wKzB3wHJydqaTYAZsW0ayzRDpM3FEaAhRxEALw_wcB&gad_source=1 [Accedido 16/04/2024]
- https://www.amazon.es/SanDisk-port%C3%A1til-resistente-orificio-mosquet%C3%B3n/dp/B08GTYFC37/ref=asc_df_B08GTYFC37/?tag=googshopes-21&linkCode=df0&hvadid=469263912841&hvpos=&hvnetw=g&hvrand=11256714707857685585&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=20297&hvtargid=pla-978531831726&psc=1&mcid=6a12dd26d26835f38ccbb478302d58af [Accedido 16/04/2024]
- https://relepro.com/programadora-pg/4454/siemens_simatic-field-pg-m6-advanced_6es7718-1cc00-0ac1.html [Accedido 16/04/2024]
- https://es.wiautomation.com/checkout [Accedido 16/04/2024]

El precio desglosado en el apartado 2.1 de la Tabla 1 es la parte proporcional del uso del SIMATIC Field PG 6 para la realización del proyecto (Se ha tenido en cuenta el uso de ambos programadores).

Para una vida útil de 7 años -> 61320 horas útiles

2.1 $Con\ un\ uso\ de\ 80h + 150h\ de\ programación \rightarrow \quad \frac{230(h)*7509{,}74(€)}{61320(h)} = 28{,}17€$

Elementos software.

Nº	DESCRIPCIÓN	IMPORTE		
		Precio (€)	Unidad (n)	Total (€)
1	SIMATIC S7, STEP 7 PROFESSIONAL V17 TIA PORTAL	3.292,83 €	1	
1.1	SIMATIC S7, STEP 7 PROFESSIONAL V17 TIA PORTAL Programador	86,46 €	1	86,46 €
2	Microsoft Visual Studio Professional 2022 (PC)	24,90 €	1	24,90 €
				111,36 €

Tabla 2: Presupuesto de elementos Software.

Los precios mostrados en la Tabla 2 se han obtenido de las siguientes fuentes:

- https://www.codigi.es/caja/?_gl=1*1f3pzps*_up*MQ..&gclid=Cj0KCQjwlN6wBhCcARIsAKZvD5gcfTjz7Exhf9PpBkji_o-OUl1H2w9gYq7B0xveb00Q2_GX_fAgrz0aAmDcEALw_wcB [Accedido 16/04/2024]
- https://relepro.com/plc-simatic-s7-1200/4547/siemens_simatic-s7-step-7-professional-v17-version-descargable_6es7822-1ae07-0ya5.html [Accedido 16/04/2024]

El precio desglosado en el apartado 1.1 de la Tabla 2 es la parte proporcional del uso de la licencia SIMATIC S7, STEP 7 PROFESSIONAL V17 TIA PORTAL para la realización del proyecto (se ha tenido en cuenta el uso de ambos programadores).

Licencias con una vida de 365 días -> 8760 horas de licencia

1.1$Con\ un\ uso\ de\ 80h + 150h\ de\ programación \rightarrow \quad \frac{230(h)*3292{,}83(€)}{8760(h)} = 86{,}46€$

Mano de obra.

DESCRIPCIÓN	IMPORTE		
	Precio (€/h)	Cantidad (Horas)	Total (€)
Programador Senior de PLCs	19,73 €	80	1.578,40 €
Programador Junior de PLCs (Prácticas)	5,07 €	400	2.028,00 €
			3.606,40 €

Tabla 3: Presupuesto mano de obra.

Los sueldos fijados en la Tabla 3 se basan en:

Descomposición Programador Senior	
Salario Base (Brutos)	35.000,00 €
Horas Anuales	1774,00
Salario €/h	19,73 €

Tabla 4: Presupuesto salario programador senior.

Descomposición Programador Junior	
Salario Base (Brutos)	9.000,00 €
Horas Anuales	1774,00
Salario €/h	5,07 €

Tabla 5: Presupuesto salario programador junior.

3. Presupuesto final

DESCRIPCIÓN	IMPORTE		
	Precio (Euros)	Cantidad (Horas)	Total (Euros)
Programador Senior de PLCs	19,73 €	80	1.578,40 €
Programador Junior de PLCs (Prácticas)	5,07 €	400	2.028,00 €
			3.606,40 €

DESCRIPCIÓN	IMPORTE		
	Precio (€)	Unidad (n)	Total (€)
Mano de obra	3.606,40 €	1	3.606,40 €
Elementos hardware	3.078,95 €	1	3.078,95 €
Elementos software	111,36 €	1	111,36 €
Total presupuesto de Ejecución Material			6.796,70 €

Gastos Generales 15%	1.019,51 €
Beneficio Industrial 6%	407,80 €

Total presupuesto sin IVA			8.224,01 €

IVA (21%)	1.727,04 €

Total presupuesto			9.951,05 €

Tabla 6: Presupuesto final.

El presupuesto asciende a la cantidad de: NUEVE MIL NOVECIENTOS CINCUENTA Y UN EUROS CON 5 CÉNTIMOS.

DOCUMENTO Nº3 PLIEGO DE CONDICIONE

Índice

1. Objeto

El objeto del presente pliego de condiciones es establecer los requisitos y especificaciones técnicas que deben cumplir todos los componentes y funcionalidades del sistema de control y supervisión para la máquina industrial. Este pliego tiene como objetivo proporcionar una base clara y detallada para el desarrollo, implementación y evaluación del proyecto.

Se busca definir de manera precisa las características que deben tener tanto el hardware como el software que compondrán el sistema, así como los criterios de rendimiento y calidad que se esperan alcanzar. Además, el pliego de condiciones establece los procedimientos de prueba y validación que se utilizarán para verificar el cumplimiento de los requisitos y garantizar el correcto funcionamiento del sistema en condiciones reales de operación.

2. Condiciones de los materiales

En este apartado se recogen las especificaciones técnicas del material necesario para llevar a cabo el proyecto.

2.1 Ordenador personal

- **Procesador:** Se recomienda un procesador Intel Core i5 de séptima generación o equivalente.
- **Memoria RAM:** Mínimo 8 GB de RAM DDR4.
- **Almacenamiento:** Disco duro SSD de al menos 256 GB.
- **Sistema Operativo:** Windows 10 Professional (64 bits).
- **Puertos de conexión:** Puerto Ethernet LAN, USB 3.0, HDMI.
- **Conectividad:** Conexión a Internet por Ethernet o Wi-Fi.
- **Pantalla:** Monitor de al menos 15 pulgadas con resolución Full HD (1920x1080).
- **Periféricos:** Teclado, ratón y otros dispositivos periféricos necesarios para el desarrollo del proyecto.
- **Software:** Visual Studio 2022 y TIA Portal v17 con licencia profesional.

2.2 PLC

El PLC seleccionado para el proyecto es el modelo Siemens ET 200SP Open Controller CPU 1515SP PC, si se sustituyese por otro modelo contactar con GH Induction o escoger un modelo compatible indicado por el fabricante con estas especificaciones mínimas.

- **Versión Firmware:** 21.9.
- **Procesador:** Dual-Core 1GHz, AMD G Series APU T40E.
- **Memoria principal:** 4Gbyte RAM
- **Disco Flash:** 30GB
- **Sistemas operativos:** Windows Embedded Standard 7 E 32 bits.
- **Tensión alimentación:** 24V.
- **Consumo corriente entrada (valor nominal):** 1,5 A.
- **Consumo eléctrico:** máx. 36 W.
- **Tarjeta de memoria CFast:** Sí; memoria Flash 30 GB.

- **Número de interfaces Industrial Ethernet**: 2.
- **Número de interfaces PROFINET:** 1
- **Número de interfaces USB:** 3; 3 USB 2.0 en el frente de 500 mA cada uno; de ellos, 2 de 500 mA y 1 de 100 mA simultáneamente.

2.3 Cámara

La cámara seleccionada para el proyecto es el modelo IR EYEBALL NETWORK CAMERA H.265+ de la marca Dahua, si se sustituyese por otro modelo contactar con GH Induction o escoger un modelo compatible indicado por el fabricante con estas especificaciones mínimas.

- **Fiabilidad:** Resistente ante entornos industriales adversos, altas temperaturas, humo, agua, etc.
- **Compatibilidad:** Compatible con los protocolos de comunicación estándar utilizados TCP/IP.
- **Software de Gestión:** Posibilidad de utilizar software de gestión proporcionado por el fabricante para configurar y controlar la cámara de manera eficiente.

2.4 Software

Si se requiere alguna actualización o cambio del programa desarrollado se debe contactar con GH Induction para evaluar su viabilidad técnica. Cualquier cosa que el cliente modifique por su cuenta, GH Induction no se hace responsable.

En caso de no seguir las directrices resaltadas y no consultar con la empresa las posibles modificaciones, la empresa GH Induction, no se hace responsable de los posibles daños.

3. Condiciones de la ejecución

La forma adecuada y recomendada, por GH Induction, sobre la ejecución del sistema de visualización viene explicada en el apartado **5. Manual de usuario** del documento de la memoria. Si el cliente actúa de una forma distinta a la mencionada GH Induction no se hace responsable de los posibles daños o problemas que puedan surgir.

Para garantizar el correcto funcionamiento de la cámara y del sistema en su conjunto, es importante establecer las condiciones adecuadas que minimicen los riesgos de fallos o problemas. Todo lo que no cumpla con los siguientes requisitos GH Induction no se hace responsable de los problemas que esto pueda ocasionar.

- **Temperatura:** La temperatura de trabajo debe estar entre los -40ºC y los 60 ºC.
- **Humedad relativa:** Se debe tener una humedad relativa inferior al 95% para prevenir la condensación de humedad dentro de la cámara.
- **Protección contra la intemperie:** Uso de carcasas protectoras que proteja a la cámara de lluvia, nieve o la exposición directa al sol.
- **Estabilidad estructural:** Se debe instalar en una estructura sólida y estable que garantice su sujeción y evite vibraciones o movimientos que puedan afectar la imagen.

ANEXO I: INFORME DE ALINEAMIENTO CON LOS OBEJTIVOS DE DESARROLLO SOSTENIBLE

Objetivos de desarrollo sostenible	Alto	Medio	Bajo	NP
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructura.	X			
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumos responsables.				X
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

La implementación exitosa de los objetivos específicos también se traducirá en una mejora en el punto 9 de los Objetivos de Desarrollo Sostenible “Industria, innovación e infraestructura” en los siguientes términos.

- **Promoción de la innovación tecnológica:** La aplicación de tecnologías como la comunicación por sockets TCP/IP y el futuro uso de servidores OPC UA demuestran un enfoque innovador para mejorar los procesos industriales.
- **Desarrollo de infraestructuras digitales:** Al integrar sistemas de control basados en PLC y pantallas HMI, el proyecto está contribuyendo al desarrollo de infraestructuras digitales en entornos industriales. Estas infraestructuras son esenciales para mejorar la eficiencia, productividad y la capacidad de respuesta en las operaciones industriales.
- **Mejora de la eficiencia y la sostenibilidad:** Al optimizar los procesos de control y monitoreo en el contexto industrial, el proyecto contribuye a la mejora de la eficiencia operativa de reducción del consumo de recursos y, por tanto, reducciones de desperdicios y emisiones.