SCO Gateway - Guía de programación

Entorno global

La clase `Entorno` mantiene variables y objetos de acceso global que mantienen información para acceder en diversos puntos del programa.

Es un "singleton".

Agregar a `Entorno` cualquier variable u objeto visible a todo el programa.

Información de configuración

Archivo configuración: `appsettings.json`

La aplicación espera el archivo de configuación con los parámetros de operación. Archivo ubicado en la raíz de carpeta con programa ejecutable.

Parámetros de configuración

En la siguiente lista se presentan los parámetros de configuración definidos.

En la siguiente lista, los parámetros y sus valores.

Sección **GatewayConfig**:

- **POS** - Indica el tipo de POS al que se conecta el gateway. Según el tipo de POS se activan los comandos adecuados. Valores aceptados:
  - 'ECO': usado para pruebas, retorna el mismo mensaje que recibe.
  - 'ECO-POSBC': usado para pruebas, remite el mensaje que recibe a POSBC de pruebas y retorna respuesta sin cambios.
  - 'POSBC': conecta con Toshiba POSBC.
  - 'EVAPOS': conecta con pos EvaPOS.
  - 'GK': conecta a pos GK Ommichannel.
- **IpSCO** string, dirección IP del SCO.
- **PortSCO** string, puerto IP del SCO.

Sección **POSBC**:

- **Ip**: IP del servidor POSBC.
- **Port**: Puerto del servidor POSBC.

Programación de la configuración

La clase `Config.cs` mantienen los valores permitidos a usar como parámetros predefinidos en el archivo de configuración de la aplicación.

Cada nuevo parámetro de configuración debe ser agregado a esta clase.

Otras secciones del programa pueden requerir validar los valores recibidos por la configuración y según estos valores tomar acción. La clase `Config` mantiene los valores permitidos.

Un uso interesante de esta clase es generar un lista o copia de estructura de archivo configuración de ejemplo para ayudar en la configuración de uno de estos y en tener la lista de valores permitidos en sus parámetros.

Procesar solicitudes del SCO

La siguiente es la secuencia de pasos que sigue el programa para procesar una solicitud (mensaje) remitido por el SCO.

- Inicialización de adaptadores y comandos, según el tipo de POS.
- El servidor SCO inicia y queda en modo escucha.
- Para cada mensaje que ingresa, el ciclo es:
    - Se determina el tipo de mensaje.
    - Según el tipo de mensaje se obtiene el comando adecuado y su adaptador. El factory de creación del directorio de comandos se encargga de inicializar los comandos adecuados al tipo de POS.
    - Se ejecuta el comando.

Agregar soporte a una nueva POS

Para soportar un nuevo tipo de POS, los pasos son:

1. Definir parámetro de configuración que indica el tipo de POS. En la clase `Constantes.cs` crear una constante con el valor esperado en el archivo de configuración `appsettings.json`, por ejemplo `gk_test`.
    1. Si es necesario, crear una clase con parémtros particulares de configuración para la nueva POS, por ejemplo, una clase `ConfigGk` para los parémtros propios de la POS Gk.
    2. Crear en el archivo de configuración una sección para estos parámetros de configuración.
2. Editar el archivo `appsettings.json` y asignar al parámetro `GatewayConfig`:`POS` el valor que identifica el tipo de POS, por ejemplo `gk_test`. Agregar la sección particular de la nueva POS si es el caso, según se indica en el paso anterior.
3. Crear una clase factory de directorio de comandos implementando `IDispensaDirectorioCmds`. Nombrar la clase `DispensaDirectorioCmds<tipo_pos>.cs`.
4. Editar la clase factory `DirectorioCmdsFactory` y agregar el nuevo dispensador al método `CreaDirectorioCmds` para que según el valor del parámetro de configuración seleccione el factory adecuado. A las clases factory de directorio de comandos se pasa como parámetro el objeto tipo `Config` para brindar acceso a las clases factory a parámetros en el archivo de configuración que puedan ser requeridos.
5. Crear una carpeta para almacenar las clases de comando. Llamar la carpeta `Comandos<tipo_pos>`, por ejemplo `ComandosGkPruebas`.
6. Crear los comandos en la carpeta. Usar un espacio de nombres dedicado para cada tipo de POS, con el patrón `SCOGateway.Comandos.<tipo_pos>`, por ejemplo `SCOGateway.Comandos.GkPruebas`.
7. Instanciar los comandos en el método adecuado de la case factory.

8. Si se requiere almacenar valores que deben ser manetenidos entres comandos o entre interacciónes, se usa el singleton `Entorno<T>`. Este singleton es una clase que maneja tipos genéricos y mantiene un atributo tipo `T` genérico al cual la implementación de POS puede asociar un objeto con los datos que debe mantener. La implementación de POS particular puede crear una clase con los datos de su necesidad y vincularla al `Entorno<T>` usando los métodos `setDatos` / `getDatos`. Un buen punto para inicializar el entorno con el contenedor de datos apropiado para el tipo de POS es la clase dispensadora de comandos, la cual es invocada por el factory de dispensadores al inicio del gateway. Por ejemplo, para almacenar datos a un tipo_pos se crea una clase que almacen los datos particulares, sea por ejemplo `TipoPOS`, se instancia el entorno con `Entorno<TipoPOS>.Instancia.setDatos(new TiposPOS())`, se puede fijar y leer datos particulares de este tipo de POS con `Entorno<TipoPOS>.Instancia.getDatos().DatoParticularDelTipoPOS = valor;`

Integración con GK Smart POS

Para integrar GK Smart POS se tienen dos implementaciones en el código:

- Una maqueta de servicios web de Smart POS, esta estará identificada por el sufijo `GkPruebas`. S
- La integración con Smart POS propiamente dicha, estará indentificada por el sufijo `Gk`.

Espacios de nombres:

- `SCOGateway.POSGk` contiene el código para la integración oficial con Smart POS Gk.
- `SCOGateway.POSGkPruebas` contiene el código para la integración con el servidor maqueta de los servicios web de Smart POS Gk.

Carpetas de código asociado con GK Smart POS:

- `ComandosGkPruebas` y `ComandosGk` contiene los comandos para ambos tipos de ambientes de Gk.
- `POSGk` contiene código especializado para GK Smart POS.

Integración con POSBC

La clase ClienteServidorPOSBC maneja la conexión con el servidor POSBC.

Pendientes

1. Pasar usuario/clave al archivo de config.
2. Probar respuestas negativas del smart pos y por tanto enviar mensaje excepción al SCO.

Immplementación comandos GK pruebas