

# Simple demo of Bosch Security SDK

## SimpleWpfClient

The sample program is a WPF client. The implementation that used the SDK is in the class `MainWindowViewModel`, which has very little WPF-specific code in it.

The client does the following

- Accepts connection information from the user.
- Allows the user to connect and disconnect from the panel.
- While connected, lets the user turn on and off an Output.
- Logs any Output status change to the screen.

## Steps to using the SDK

### Instantiate the SessionManager

The `SessionManager` enables the client to manage the life cycle of a Session: Initialize, Connect, Disconnect and Destroy. The `SessionManager` also provide a reference to a `MessageBus`. The `MessageBus` is the communication conduit between the client and the Session.

In the demo application, this is done in the `MainWindowViewModel` constructor:

```
public MainWindowViewModel()  
{  
    _sessionManager = new SessionManager(new TransportProviderFactory());  
    Bus = _sessionManager.MessageBus;  
}
```

### Register for Message Bus Events

Next you want subscribe to various events from the Session you will be creating. In the demo app this is done in the `RegisterForMessageBusEvents` method. We register for 3 connection events:

- `SessionConnectedEvent`
- `SessionConnectFailedEvent`
- `SessionClosedEvent`

Since our simple application is showing information about Outputs, we register for events concerning outputs:

- `OutputAddedEvent`
- `OutputStateChangedEvent`

Note that each listener filters based on a Session Id. That Session id is generated when you initialize a session.

### Initialize the Session

Once you have the connection information from the user, you initialize a Session:

```
_sessionId = _sessionManager.InitializeSession(sdkProfile, null);
```

You must store the session id returned from this command, as you will need to communicate to the session you have created, and to filter messages on the Bus.

## Connect to the Panel

When you connect to a panel, you must tell the SDK what types of events you are interested in. This is done by passing a list of PanelEvents to the SDK. In our example, we are requesting Output changes:

```
_requestedEvents = new List<RequestedEvent>
{
    new RequestedEvent { PanelEvent = PanelEvents.OutputStateChanges,
        IntervalInSeconds = 5, MetadataRetrievalSeconds = 60 }
};
```

The IntervalInSeconds parameter determines how often the SDK polls for state, such as whether or not an Output is on. MetadataRetrievalSeconds is how often the SDK polls for metadata changes, such as Output text.

To connect we make the following call:

```
await _sessionManager.ConnectAutomationSession(_sessionId, _requestedEvents);
```

There is no direct reply from this command. Instead either a SessionConnectedEvent or a SessionConnectFailedEvent will be published by the Session. You would take appropriate actions in those handlers.

## Controlling the Panel

By convention, messages that the client subscribes to are Events, while messages the client publishes are Commands. In the demo, the command to control an Output is issued in

```
private void ControlOutput(int outputNumber, OutputCommands outputCommand)
{
    Bus.Publish(new ControlOutputCommand(_sessionId, outputNumber,
        outputCommand));
}
```

## Disconnecting from the Panel

Disconnecting from the panel is straightforward:

```
_sessionManager.DisconnectSession(_sessionId);
```

## Destroying the Session

This command disposes a session. It can no longer be used once it is destroyed.

```
_sessionManager.DestroySession(_sessionId);
```

## Notes

For simplicity, the demo app creates and destroys a Session each time the user clicks Connect. This is not necessary if the connection information does not change. You could initialize the session, connect and disconnect many times, then destroy the session when you are finished with it. A common use case is implementing reconnect logic.