Table of Contents

Multiple Wiegand format support

1. Why the multiple Wiegand format support is needed

With a single Wiegand format stored on the device, you need to change the Wiegand configuration of the device every time you want to read another Wiegand format card. With multiple Wiegand formats stored on the device, you don’t need to change the Wiegand configuration for the device to read cards that have different Wiegand formats. Multiple Wiegand format support is critical when the company decides to change the Wiegand format of its access cards to ensure higher security (e.g. from 26-bit Prox to 35-bit Corporate 1000). Since it’s practically impossible to replace the cards of all the employees at the same time, there is always a transition period. This requires the card reader to read cards with different Wiegand formats. The readers that do not support multiple Wiegand formats cannot accommodate this scenario.

2. How the multiple Wiegand format support works

If you use BioStar as an access control management system, the Wiegand format configurations on the BioStar server gets automatically transferred to the devices. Then, on the device detail page, you need to specify which Wiegand formats to use. If you develop your own application using BioStar Device SDK, you need to create and transfer the Wiegand format configurations. You also need to transfer a configuration that contains the indexes of the Wiegand formats to specify which Wiegand formats to use. In the BS2WiegandMultiConfig data structure, you can store up to 15 different Wiegand formats in the formats field.\

```
public struct BS2WiegandMultiConfig
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 15)]
    public BS2WiegandInConfig[] formats;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 32)]
    public byte[] reserved;
}
```

In the BS2WiegandConfig data structure, there are three mask fields where you can specify the indexes of the Wiegand formats to use:

```
public struct BS2WiegandConfig
{
    public byte mode;
    public byte useWiegandBypass;
    public byte useFailCode;
    public byte failCode;
    public UInt16 outPulseWidth; //(20 ~ 100 us)
    public UInt16 outPulseInterval; //(200 ~ 20000 us)
    public UInt32 formatID;
    public BS2WiegandFormat format;
    public UInt16 wiegandInputMask;
```

```
    public UInt16 wiegandCardMask;
    public byte wiegandCSNIndex;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 27)]
    public byte[] reserved;
}
```

- wiegandInputMask: contains a bitmask that indicates the indexes of the Wiegand formats to use for slave devices and Wiegand devices. The slave device works as a dummy device and does not store any Wiegand configurations on it (a slave device is a Suprema device that is connected to another Suprema device through RS485 as a dummy reader). If you want to specify the first and third elements (array index: 0 and 2) of the formats array field of the BS2WiegandMultiConfig data structure, the bitmask should be set to 0000000000001010, because the 0th bit is reserved for the backward compatibility. This wiegandInputMask field is applicable when reading a Wiegand card (iClass or Prox) on the slave device (or the Wiegand device).
- wiegandCardMask: contains a bitmask that indicates the indexes of the Wiegand formats to use for the master device. When the device is working as a master device, it reads the Wiegand cards that match the Wiegand formats specified in this field. If you want to specify the first and third elements (array index: 0 and 2) of the formats array field of the BS2WiegandMultiConfig data structure, the bitmask should be set to 0000000000001010, because the 0th bit is reserved for the backward compatibility. This wiegandCardMask field is applicable when reading a Wiegand card (iClass or Prox) on the master device.
- wiegandCSNIndex: contains the index of the Wiegand formats used to convert the card serial number for both the master device and the slave device. When you flash a non-Wiegand card (Mifare or EM) to the device, it reads the card serial number from the card and converts it into the Wiegand format specified in this field. Only one index can be set to this field. If you want to convert a card serial number into the Wiegand format stored on the 3rd element (array index: 2) of the formats array field of the BS2WiegandMultiConfig data structure, the wiegandCSNIndex should be set to 3. This wiegandCSNIndex field is applicable when reading a non-Wiegand card (Mifare or EM) on both master and slave devices.

When the three mask fields above are all set to 0, the device will try to read the Wiegand cards that match the Wiegand format stored in the BS2WiegandFormat field of the BS2WiegnadConfig data structure:

```
public struct BS2WiegandConfig
{
    public byte mode;
    public byte useWiegandBypass;
    public byte useFailCode;
    public byte failCode;
    public UInt16 outPulseWidth;     //(20 ~ 100 us)
    public UInt16 outPulseInterval;    //(200 ~ 20000 us)
    public UInt32 formatID;
    public BS2WiegandFormat format;
    public UInt16 wiegandInputMask;
    public UInt16 wiegandCardMask;
    public byte wiegandCSNIndex;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 27)]
    public byte[] reserved;
```

```
}
```

3. Example (C#)

```
// 26 bit H10301
IntPtr the26bitformatPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(BS2WiegandFormat)));
BS2WiegandFormat the26bitFormat = new BS2WiegandFormat();
the26bitFormat.idFields = new byte[128];
the26bitFormat.parityFields = new byte[128];
the26bitFormat.parityType = new byte[4];
the26bitFormat.parityPos = new byte[4];

the26bitFormat.length = 26;

the26bitFormat.parityPos[] = ;
the26bitFormat.parityPos[1] = 25;

the26bitFormat.parityType[] = 2;
the26bitFormat.parityType[1] = 1;
the26bitFormat.parityType[2] = ;
the26bitFormat.parityType[3] = ;

the26bitFormat.parityFields[28] = 1;
the26bitFormat.parityFields[29] = 255;
the26bitFormat.parityFields[30] = 224;
the26bitFormat.parityFields[62] = 31;
the26bitFormat.parityFields[63] = 254;

the26bitFormat.idFields[28] = 1;
the26bitFormat.idFields[29] = 254;
the26bitFormat.idFields[61] = 1;
the26bitFormat.idFields[62] = 255;
the26bitFormat.idFields[63] = 254;

Marshal.StructureToPtr(the26bitFormat, the26bitformatPtr, false);

// Corporate 1000
IntPtr theCorp1000FormatPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(BS2WiegandFormat)));
BS2WiegandFormat theCorp1000Format = new BS2WiegandFormat();
theCorp1000Format.idFields = new byte[128];
theCorp1000Format.parityFields = new byte[128];
theCorp1000Format.parityType = new byte[4];
theCorp1000Format.parityPos = new byte[4];

theCorp1000Format.length = 35;

theCorp1000Format.parityPos[] = 1;
```

```
theCorp1000Format.parityPos[] = 1;
theCorp1000Format.parityPos[1] = 34;

theCorp1000Format.parityType[] = 2;
theCorp1000Format.parityType[1] = 1;
theCorp1000Format.parityType[2] = 1;

theCorp1000Format.parityFields[27] = 1;
theCorp1000Format.parityFields[28] = 182;
theCorp1000Format.parityFields[29] = 219;
theCorp1000Format.parityFields[30] = 109;
theCorp1000Format.parityFields[31] = 182;
theCorp1000Format.parityFields[59] = 3;
theCorp1000Format.parityFields[60] = 109;
theCorp1000Format.parityFields[61] = 182;
theCorp1000Format.parityFields[62] = 219;
theCorp1000Format.parityFields[63] = 108;
theCorp1000Format.parityFields[91] = 3;
theCorp1000Format.parityFields[92] = 255;
theCorp1000Format.parityFields[93] = 255;
theCorp1000Format.parityFields[94] = 255;
theCorp1000Format.parityFields[95] = 255;

theCorp1000Format.idFields[27] = 1;
theCorp1000Format.idFields[28] = 255;
theCorp1000Format.idFields[29] = 224;
theCorp1000Format.idFields[61] = 31;
theCorp1000Format.idFields[62] = 255;
theCorp1000Format.idFields[63] = 254;

Marshal.StructureToPtr(theCorp1000Format, theCorp1000FormatPtr, false);
Marshal.Copy(the26bitformatPtr, wiegandMultiConfig.formats[].format, ,
             Marshal.SizeOf(typeof(BS2WiegandFormat)));
Marshal.Copy(theCorp1000FormatPtr, wiegandMultiConfig.formats[3].format, ,
             Marshal.SizeOf(typeof(BS2WiegandFormat)));
Marshal.FreeHGlobal(the26bitformatPtr);
Marshal.FreeHGlobal(theCorp1000FormatPtr);

result = API.BS2_SetWiegandMultiConfig(sdkContext, deviceId, ref
wiegandMultiConfig);
if (result != (int)BS2ErrorCode.BS_SDK_SUCCESS)
{
    Console.WriteLine("BS2_SetWiegandMultiConfig failed: " + result);
    API.BS2_DisconnectDevice(sdkContext, deviceId);
    API.BS2_ReleaseContext(sdkContext);
    return;
}

BS2WiegandConfig wiegandConfig;
result = API.BS2_GetWiegandConfig(sdkContext, deviceId, out wiegandConfig);
if (result != (int)BS2ErrorCode.BS_SDK_SUCCESS)
```

```
{
    Console.WriteLine("BS2_GetWiegandConfig failed: " + result);
    API.BS2_DisconnectDevice(sdkContext, deviceId);
    API.BS2_ReleaseContext(sdkContext);
    return;
}
wiegandConfig.wiegandCardMask = 18; // Use the first and fourth Wiegand
formats only
                                    // (0x0000000000010010)

result = API.BS2_SetWiegandConfig(sdkContext, deviceId, ref wiegandConfig);
if (result != (int)BS2ErrorCode.BS_SDK_SUCCESS)
{
    Console.WriteLine("BS2_SetWiegandConfig failed: " + result);
    API.BS2_DisconnectDevice(sdkContext, deviceId);
    API.BS2_ReleaseContext(sdkContext);
    return;
}
```