AN14169

How to Generate a User-Defined Class USB Device based on i.MX RT Chips

Rev. 1 — 8 March 2024 Application note

Document information

Information	Content
Keywords	AN14169, USB, User-Defined Class
Abstract	This application note describes how to build a user-defined class USB device.

1 Introduction

This application note describes how to build a user-defined class USB device.

This application note is based on the `usb_device_cdc_vcom` demo. It describes in detail how to modify this demo to implement a user-defined class device that only contains two bidirectional Bulk endpoints. It also describes how to create a computer-side host application to communicate with the USB device.

2 USB user-defined class device overview

USB defines class code information that is used to identify the functionality of a device and used to nominally load a device driver based on that functionality. When developing a USB device, the engineer commonly chooses to use a class that the USB-IF defined depending on the function. Whether it is a Windows or Linux-based system, most of the host class drivers have been implemented according to the definitions of the USB-IF. Therefore, for most USB applications, basic functions can be achieved according to the definition, and most of them can be used directly on the personal computer.

The existing class definitions cover most of USB applications, but in the embedded system, besides these functions, the developers have more data transfer tasks to perform through high-speed interfaces such as USB. While defining the class code, the USB-IF also defines an option for a custom class. Developers can select `0xFF` as the value of the class field in the interface descriptor, and then define each field in the interface descriptor according to their own needs.

This application note takes a simple custom class that only contains two Bulk bidirectional transmission endpoints as an example to introduce the implementation of the custom class and the related host application. Based on actual application requirements, developers can add multiple interfaces containing different types of endpoints to achieve more complex data transmission requirements.

3 Implement a user-defined class device

This application note uses the `usb_device_cdc_vcom` demo in the 2.14.0 RT1170-EVKB SDK package as an example. This demo implements two interfaces, the CDC Control interface for transmitting control information and the CDC Data interface for data transmission. In scenarios where only bi-direction data communication is required, the CDC Control interface can be completely removed. So, we can implement a user-defined class device which only contains four descriptors, one device Configure descriptor, one Interface descriptor, and two Endpoint descriptors.

To implement a user-defined device, perform the following steps:

1. Modify the descriptor part.
   In the original code of the demo, a macro `USB_DEVICE_CONFIG_CDC_CIC_EP_DISABLE` is defined. After opening this macro, the endpoints in the CDC Control Interface are disabled. Enable this macro and modify the elements in the array of `g_UsbDeviceConfigurationDescriptor`. Modify the length item according to Figure 1.

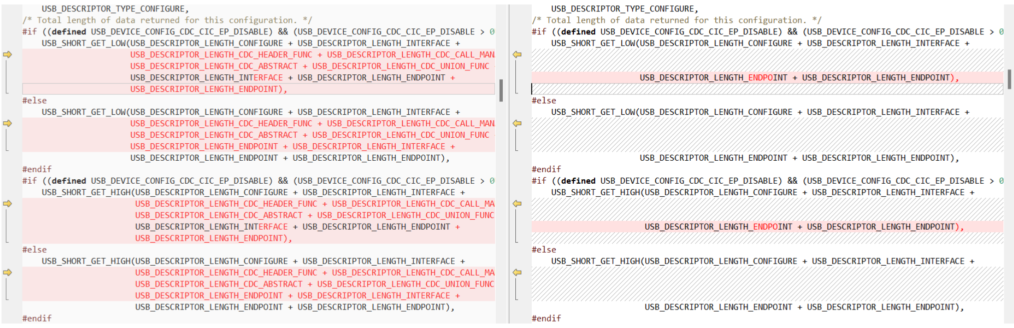

Figure 1. Modify the length

2. In the second half of the array, delete all six descriptors related to the Control Interface and only reserve four descriptors. Then change the Class part of the CDC Data Interface descriptor to a user-defined class, as shown in Figure 2.

Figure 2. Modify interface descriptor

3. After modifying the Interface descriptor part, delete the first group of elements in the `g_UsbDeviceCdcVcomInterfaces` array, which is used to initialize the class driver and change the class definition in this array.

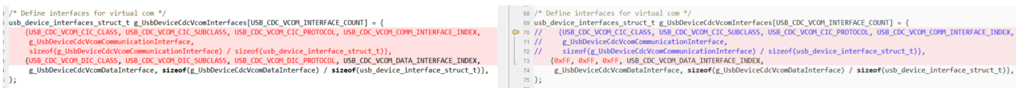

Figure 3. Modify interface array

4. Change the value of the macro `USB_CDC_VCOM_INTERFACE_COUNT` in *usb_device_descriptor.h* to **1**and the value of `USB_DEVICE_CLASS` to **0x00**.
   Besides the descriptor part, change the CDC ACM class driver to adapt the user-defined class. The `USB_DeviceCdcAcmEndpointsInit()` in the CDC ACM class driver first gets the Control Interface and initializes the endpoint belongs to this interface. Since the Control interface has been deleted in the descriptor, delete this part too. The deleted part of the code is from line 217 to line 266 of *usb_device_cdc_acm.c*.
   After that, the function initializes the Data Interface endpoints. Before initializing the endpoints, it checks the class code in the Interface descriptor. Modify this part of the code as shown in Figure 4.

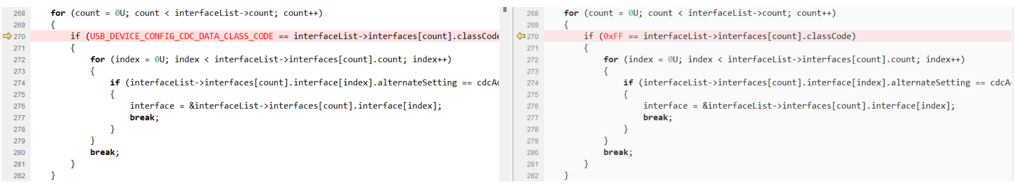

Figure 4. Modify class driver

   This device just contains the bulk transfer and the API call is basically the same as the data transmission part of CDC ACM, so the class drivers of CDC ACM can be reused. For more complex applications, developers must reconstruct their own class drivers. It is convenient to implement custom class drivers referring to the basics of various existing class drivers.
5. Modify the application layer.
   At the application level, the original settings of the demo set up a flag `startTransactions` to start the transmission, and this flag is only set after the host obtains all relevant CDC ACM requests. Therefore, in the *virtual_com.c* file, remove all the judges related to the flag `startTransactions`. And then the function of receiving data and writing back can be used directly.

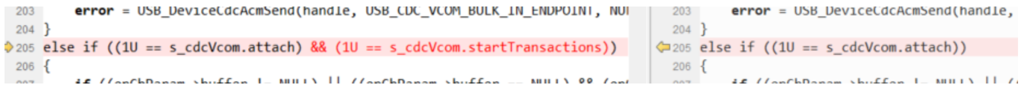

Figure 5. Modify application code

After completing all the above modifications, a user-defined device containing only two Bulk data endpoints is implemented. Compiling and downloading the project, and connecting the device to the personal computer, the device is displayed as other devices in the device manager.

4 Implement host application

Since the personal computer cannot find the related driver to support the user-defined class device, the device appears as other devices after being plugged into the personal computer. The developer must create a host application to interact with this device. The host application introduced in this application note uses `libusb`, a cross-platform library, to operate USB devices.

Firstly, download the relevant library files from https://github.com/libusb/libusb/releases. In this link, select the binary version package, which contains library files compiled by various compilers and the `libusb` header file.

Before using `libusb` to operate USB devices, update the driver for the user-defined class device to a universal USB driver. Zadig software can be used to update the driver to a universal driver, which supports the `libusb` library. Download the latest `Zadig` software from https://github.com/pbatard/libwdi/releases.

Open the `Zadig` software, and the connected user-defined device as shown in Figure 6 is displayed in the software. By default, the driver of this device is displayed as **None**.

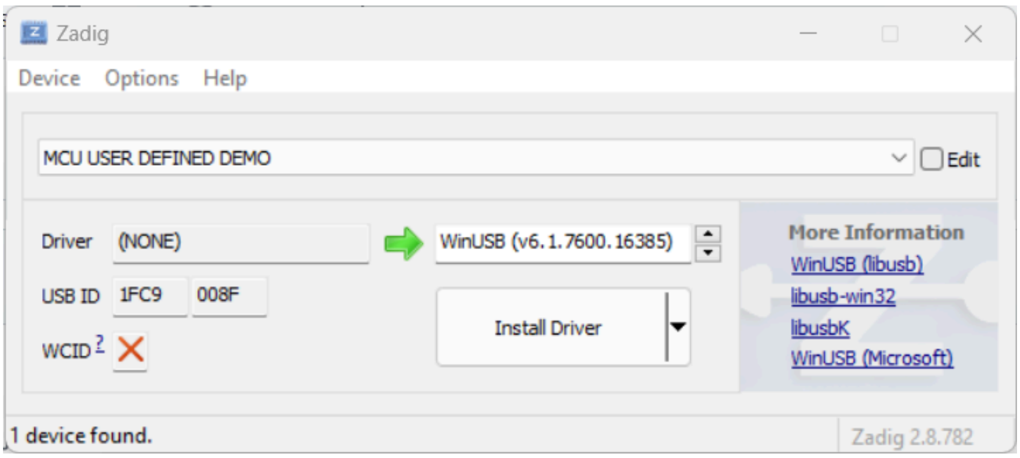

Figure 6. Zadig software

Select WinUSB in the Driver option, click the **Install Driver** button, and wait for the Driver to be installed. After the Driver installation is completed, you can see this device appear in the **Universal Serial Bus Device** list in the Device Manager.

Now, the developers can start to create the host application. The host application of this application note is developed by Visual Studio. Install the Visual Studio 2022 software and confirm that the MFC-related components are installed in the software installation interface.

After the installation is complete, to create a blank project, perform the following steps:

1. Open the software,
2. Create an MFC App project.
3. Select Dialog based on the application type item in the project creation wizard.
4. Click **Finish**.

After the project is created, enter the control editing interface, as shown in Figure 7.

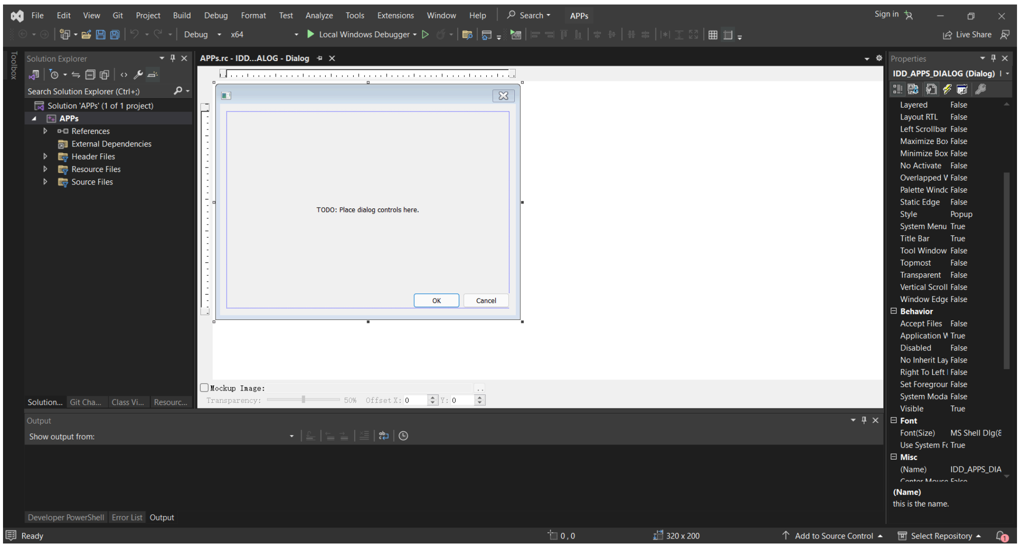

Figure 7. MFC project

To construct a form, perform the following steps:

1. Open the toolbox sidebar in this view, and select the **Button** tool,
2. Place two Button controls in the dialog box as the connect button and the send button.
3. Select the Edit Control tool, and place two Edit Control tools as the sending dialog box and the receiving dialog box respectively, as shown in Figure 8

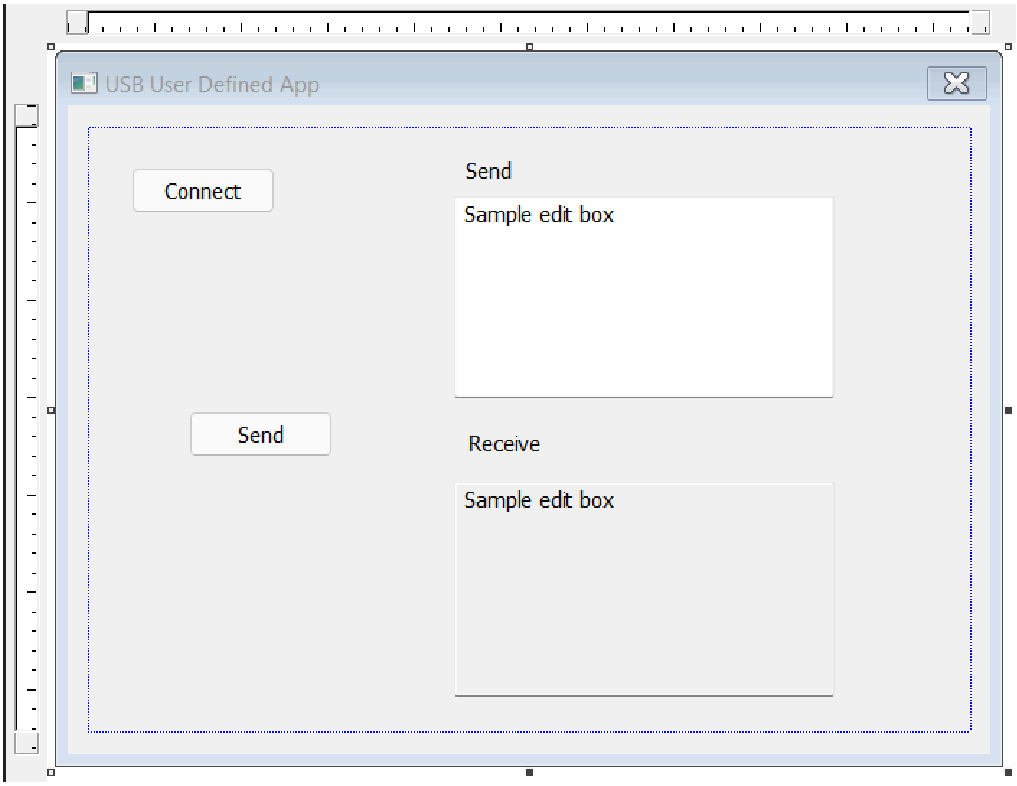

Figure 8. Form design

To add a member function, perform the following steps:

1. Right-click in the form box and select **Class Wizard**.
2. In the pop-up window, select the Connect and Send buttons you created, and choose to add the `BN_CLICKED` handler, as shown in Figure 9.

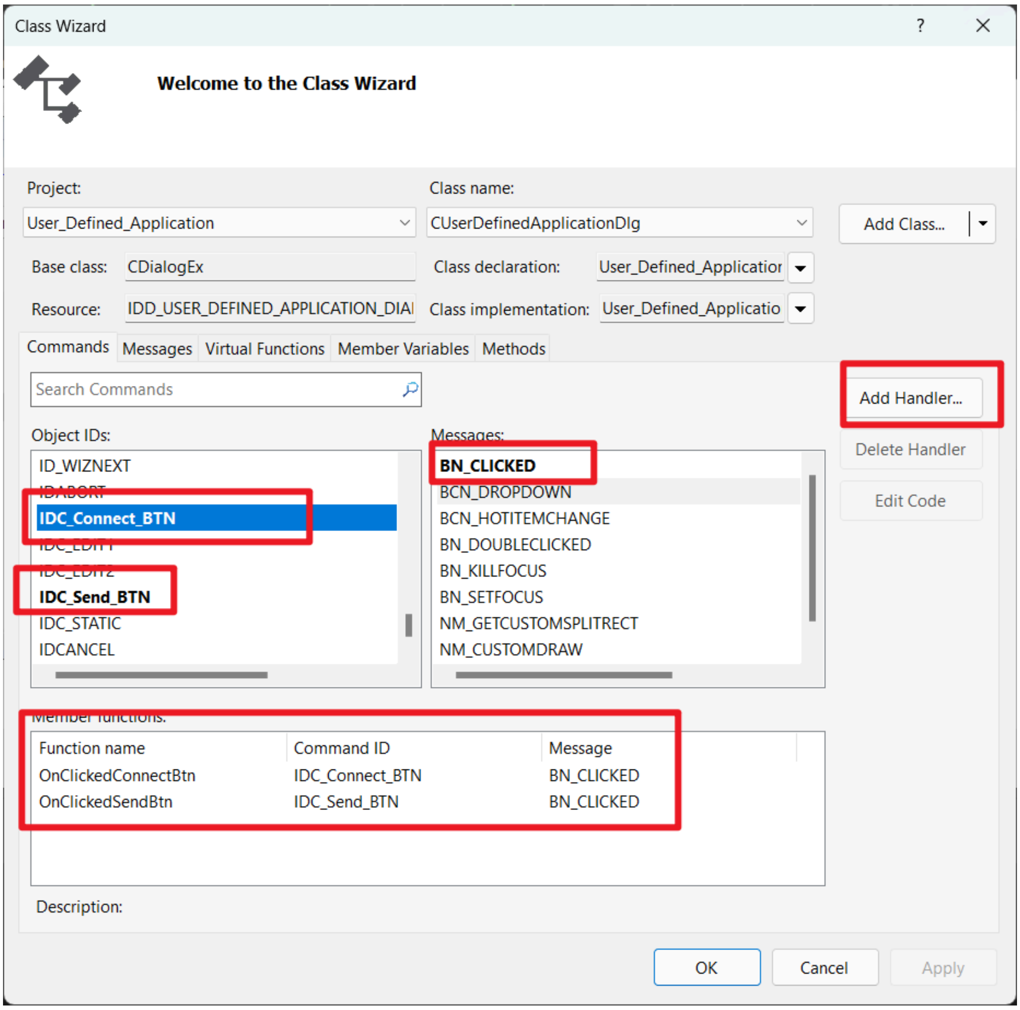

Figure 9. Add a member function

After the functions are added, switch to the solution explorer view and add the *libusb.h* file to the header file. Include the header to the form source code file of the project. Copy the *libusb-1.0.dll* and *libusb-1.0.lib* files in the *libusb-1.0.26-binaries\VS2015-x64\dll* directory to the VS solution directory.

In the properties of the solution, select the VC++ Directories item under the **Configuration Properties** entry, and add the path where *libusb-1.0.lib* is located to the Library Directories entry. Then in the Input item under the Linker entry, enter *libusb-1.0.lib* in the Additional Dependencies option. After saving the settings, click **Debug** once to generate the Debug directory. VS may report an error in this step. Ignoring the error, copy *libusb-1.0.dll* to the *\x64\Debug\ directory* in the solution directory.

Now, you can start to add code. Add two member functions in the `CUserDefinedApplicationDlg` class, which are used to connect USB devices and send data. Create the following `connect_device()` function.

```
libusb_device_handle* handle;
libusb_device* dev;
struct libusb_config_descriptor* conf_desc;
```

```
uint8_t endpoint_in = 0, endpoint_out = 0;  // default IN and OUT endpoints
int CUserDefinedApplicationDlg::connect_device()
{
    const struct libusb_endpoint_descriptor* endpoint;
    int i, j, k, r;
    int iface, nb_ifaces, first_iface = -1;
    struct libusb_device_descriptor dev_desc;
    libusb_init(NULL);
    handle = libusb_open_device_with_vid_pid(NULL, 0x1fc9, 0x0094);
    if (handle == NULL) {
        MessageBox(TEXT("Connect Failed!"), TEXT("message"), MB_OK);
        return -1;
    }
    dev = libusb_get_device(handle);
    CALL_CHECK_CLOSE(libusb_get_config_descriptor(dev, 0, &conf_desc), handle);
    nb_ifaces = conf_desc->bNumInterfaces;
    if (nb_ifaces > 0)
        first_iface = conf_desc->interface[0].altsetting[0].bInterfaceNumber;
    for (i = 0; i < nb_ifaces; i++) {
        for (j = 0; j < conf_desc->interface[i].num_altsetting; j++)
        {
            for (k = 0; k < conf_desc->interface[i].altsetting[j].bNumEndpoints;
 k++) {
                struct libusb_ss_endpoint_companion_descriptor* ep_comp = NULL;
                endpoint = &conf_desc->interface[i].altsetting[j].endpoint[k];
                if ((endpoint->bmAttributes & LIBUSB_TRANSFER_TYPE_MASK) &
 (LIBUSB_TRANSFER_TYPE_BULK | LIBUSB_TRANSFER_TYPE_INTERRUPT)) {
                    if (endpoint->bEndpointAddress & LIBUSB_ENDPOINT_IN) {
                        if (!endpoint_in)
                            endpoint_in = endpoint->bEndpointAddress;
                    }
                    else {
                        if (!endpoint_out)
                            endpoint_out = endpoint->bEndpointAddress;
                    }
                }
                if (ep_comp) {
                    libusb_free_ss_endpoint_companion_descriptor(ep_comp);
                }
            }
        }
    }
    libusb_free_config_descriptor(conf_desc);
    r = libusb_set_auto_detach_kernel_driver(handle, 1);
    for (iface = 0; iface < nb_ifaces; iface++)
    {
        r = libusb_claim_interface(handle, iface);
        if (r != LIBUSB_SUCCESS) {
            MessageBox(TEXT("Connect Failed!"), TEXT("message"), MB_OK);
            return -1;
        }
    }
    MessageBox(TEXT("Connect Successfully!"), TEXT("message"), MB_OK);
    return 0;
}
```

And create a `send_data()` function to send, receive, and display the data.

```
int CUserDefinedApplicationDlg::send_data()
{
```

```
    int r = 0;
    int size = 0;
    CString send_content;
    CString receive_content;
    CString trans;
    int len = 0;
    unsigned char data_trans[512];
    unsigned char data_recv[512];
    Send_Box.GetWindowTextW(send_content);
    len = send_content.GetLength();
    for (int j = 0; j < len; j++)
    {
        data_trans[j] = (unsigned char)send_content[j];
    }
    r = libusb_bulk_transfer(handle, endpoint_out, data_trans, len, &size, 0);
    if (r != LIBUSB_SUCCESS) {
        printf("   Failed\n");
        return -1;
    }
    int send_size = libusb_bulk_transfer(handle, endpoint_in, data_recv,
 sizeof(data_recv), &size, 2000);
    if (send_size < 0)
    {
        printf("   Failed\n");
        return -1;
    }
    for (int i = 0; i < size; i++)
    {
        trans.Format(_T("%c"), data_recv[i]);
        receive_content += trans;
    }
    Receive_Box.SetWindowTextW(receive_content);
    return 0;
}
```

Now, you can compile and run the project.

5 Running the demo

The user-defined class device and the host application are implemented respectively, the device and the application can test together now.

Download the IAR project into the RT1170-EVKB board, connect the USB OTG1 port to the personal computer, open the host application software on the personal computer, click Connect, and the connection success dialog box pops up. Type characters in the **Send** dialog box and click the **Send** button. The host reads the contents in the **Send** dialog box and sends it to the Device. When the Device receives the data, it sends them back to the Host. After the Host receives the data, it displays the data in the **Receive** dialog box. Figure 10 shows the test result.

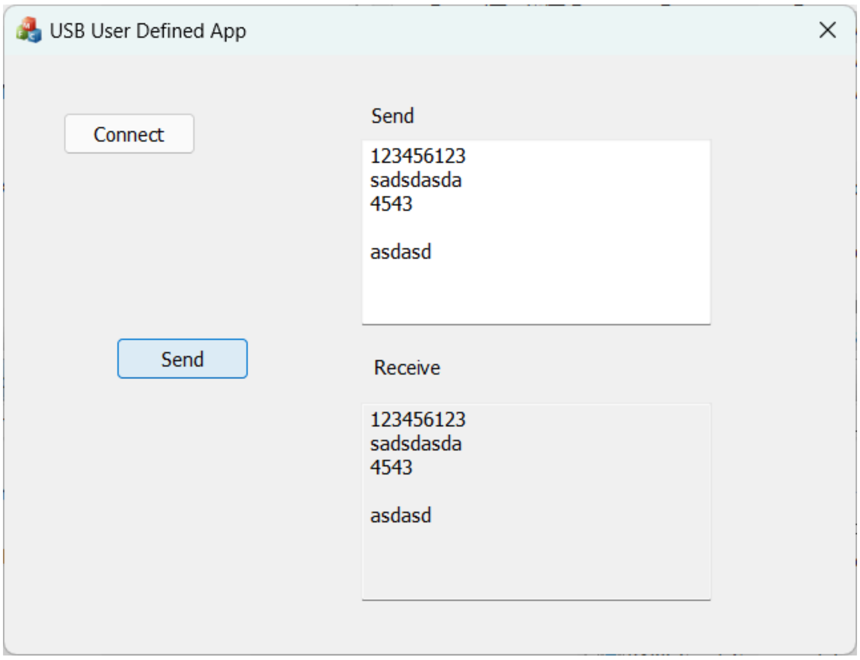

Figure 10. Test result

This application note creates a simple user-defined class device, and a simple host application as reference. For further functions, developers can add more interfaces with different types of endpoint to implement.

6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 1 summarizes the revisions to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14169 v.1	08 March 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

IAR — is a trademark of IAR Systems AB.

i.MX — is a trademark of NXP B.V.

Contents

Date of release: 8 March 2024
Document identifier: AN14169