VCP

Release 6.8

Embention

2024-02-15

CONTENTS

1	1. Introduction	1												
2	2. Requirements													
3	3. Connection with computer													
4	4. Message structure													
5	5. Examples	9												
	5.1 5.1. Common Code	10												
	5.2 5.2. File Configuration	11												
	5.3 5.3. Telemetry	13												

1. INTRODUCTION

VCP is the abbreviation for **Veronte Communication Protocol**. This Protocol allows the user communicate with veronte-based systems.

As we said before, VCP is the protocol used by Veronte-based systems. This protocol allows products to send information and receive commands from the user. This exchange of information can be done using Veronte applications (**1x PDI Builder**, **Veronte Link**, **CEX PDI Builder**, etc.). Nonetheless, it can be used by an own developed application as shown in the following image:



In addition, we call VCP to the library provided by Embention, which makes easier the process to develop an external operation tool.

TWO

2. REQUIREMENTS

To test the VCP library and understand how to use it, we have built some easy examples. But, first of all, the following devices are required to work with VCP:

- VCP library
- A computer with:
 - Windows 10
 - VCP library
 - Visual Studio 2015
- A Veronte-based system
- The harness for the Veronte device (to connect it with a computer)

THREE

3. CONNECTION WITH COMPUTER

Veronte devices can be connected to a computer with Windows via USB.

The following image shows an example of connection:



Where:

- 1. Veronte Harness.
- 2. Veronte Autopilot.
- 3. USB connected to PC.
- 4. Power supply.

Note: Make sure Veronte device is not in maintenance mode.

The PC will assign a COM port to the device, which can be checked on the Windows Device Manager as follows:

File Action View Help Image: Constraint of the state of the stat	×
 Audio inputs and outputs Bluetooth Cameras 	
 Audio inputs and outputs Batteries Bluetooth Cameras 	
> 🐲 Batteries > 🚯 Bluetooth > 👰 Cameras	^
> 🚯 Bluetooth > 👰 Cameras	
> 👰 Cameras	
> 📲 Componentes de software	
> 📃 Computer	
Disk drives	
> 🔙 Display adaptors	
> 📕 Dispositivos portátiles	
> Firmware	
> 🚧 Human Interface Devices	
> 📷 IDE ATA/ATAPI controllers	
> 🔤 Keyboards	
> 🎒 Mice and other pointing devices	
> 🛄 Monitors	
> 🚽 Network adapters	
~ 💭 Ports (COM & LPT)	
u-blox Virtual COM Port (COM9)	
🐺 USB Serial Port (COM3)	
🐺 XDS2xx Emulator CDC Serial Port (COM4)	
🐺 XDS2xx User CDC Serial Port (COM5)	
> Processors	
> 📭 Security devices	
> 🖻 Sensors	
> 📱 Software devices	
Sound video and game controllers	~

Now, the setup is ready to launch examples.

4. MESSAGE STRUCTURE

The structure of the messages between different elements within a Veronte-based system is as follows:

Field:	0xBA	UAV Org	UAV Dest	Encr Flag	IRX	Arg	Len	CRC	Data	CRC
Bytes:	1	2	2	1-bit	7-bit	1	1	1	0-255	2

Important: In VCP, the maximum packet size is 266 bytes. Consider this to ensure proper communication.

- **OxBA:** header of the message, used to identify an incoming communication as a Veronte message.
- UAV Org: ID address of origin.
- UAV Dest: ID address of destination.
- Encr Flag: encryption flag, registers whether the message is encrypted (with 1) or not (with 0).
- IRX: command type, it can be:
 - 0x00: Telemetry messages.
 - **0x0F**: Stick interface.
 - 0x22: Communication Statistics.
 - 0x29: Configuration Commands.
 - 0x2C: Simulated Navigation.
 - 0x2E: Simulated Sensor.
 - 0x05: Configuration Files.
- Arg: argument used by the command.
- Len: length, number of data bytes to be transmitted.
- CRC: Cyclic Redundancy Check, used to check whether data has been corrupted.
- Data: data to be transmitted.

FIVE

5. EXAMPLES

To start programming any VCP example or application, specify the assigned COM in the employed IDE. In Visual Studio, the COM port is specified as a project argument following the next steps:

1. Right click on project \rightarrow Properties.

*	Build		ry						
	Rebuild		nal Dependencies						
	Clean		m_hlp.cpp						
	View		_tm.cpp						
	Analyze		_H.cpp						
	Project Only		hetryrx.cpp						
	Project Only Reference CDK Version	'	-fr						
	Retarget SUK Version		ences						
	Scope to This		nal Dependencies						
Ē	New Solution Explorer View		m_hlp.cpp						
	Build Dependencies	1	.cpp						
	Add		_cfg.cpp						
в. *	Class Wizard	Ctrl+Shift+X	-H.cpp						
Ť	Manage NuGet Packages		ettyrx.cpp						
*	Set as Startlin Drojest		ences						
244	Debug		nal Dependencies						
	Debug		m_hlp.cpp						
	Source Control		'cmd.cpp						
ж	Cut	Ctrl+X	_ev						
â	Paste	Ctrl+V							
×	Remove	Del	Team Explorer Class View						
X	Rename								
	Unload Project		ct Properties						
	Rescan Solution								
9	Open Folder in File Explorer								
×	Properties	Alt+Enter	telemetry						
			lencies						

2. Configuration properties \rightarrow Debugging \rightarrow Command Arguments.

emetry Property Pages		?
onfiguration: Debug	 Platform: Active(x64) 	 Configuration Manager
Configuration Properties	Debugger to launch:	
General Debugging	Local Windows Debugger	
VC++ Directories	Command Command Arguments Working Directory Attach Debugger Type Environment Merge Environment SQL Debugging Amp Default Accelerator	S(TargetPath) LLCOMS S(ProjectDir) No Auto Yes No WARP software accelerator

5.1 5.1. Common Code

5.1.1 5.1.1. Serial Port

This example gets the serial COM port used by a Veronte device and initializes it.

```
//Serial port instance
Serial& serial = Serial::get_instance();
//Get default port
const char* port = Serial::get_default_port();
//Checks whether the user has specified a COM port
if (argc == 2)
{
    //Get port as argument
    port = argv[1];
}
//Inits serial port
serial.init(port);
```

5.1.2 5.1.2. Discovery step

In this step the the discovery IRX is added in the command manager, to discover the address ID of Veronte connected to the PC.

<pre>Base::Commgr commgr(myaddr);</pre>	<pre>//Command manager instance</pre>				
<pre>Media::Discovery discovery(commgr);</pre>	//Discovery instance				
<pre>commgr.add_irx(Base::discovery_id, discovery);</pre>	<pre>//IRX addition to Command Manager</pre>				

After this, it is possible to discover the device and its ID:

<pre>discover();</pre>	//Discover action					
<pre>while (!discovery.get_status().received) {</pre>	//Wait until the device has been discovered					
<pre>VCP::send_receive(serial, commgr); discovery.discover();</pre>	//Updating serial bus					
<pre>} Uint16 ver_addr = discovery.get_status().ad</pre>	ddr.id; //ver_addr is the Veronte Address ID.					

This ver_addr will be used by many examples as destination address.

5.2 5.2. File Configuration

With the device ID, it is possible to change its configuration, for example, in this case the routes configurable used by the UAV will be updated. (This example only shows one point update, go to main_cfg.cpp file to check all code).

Feature references creation:

```
Geo::Ftropf fop; //Operation features instance
Base::Feature f0; //Feature reference instance
Tllh v0; //64-bit longitude, latitude and WGS84 height position
// Initialize values - lon, lat, height
v0 = { { -0.01001664582567 , 0.66831910249822 }, 150 };
f0.set_abs(v0); //Sets the coordinates of a waypoint from its absolute coordinates
fop.opg[0] = f0; //Initialize the list of operation generic features
fop.opg.set_enabled(0, true); //Enables the abstract array
```

Route creation:

(continues on next page)

(continued from previous page)

```
// Enable patch
route.route.set_enabled(static_cast<Troute::Pch_index>(0), true);
```

File Configuration IRX and Config Manager IRX instances, and their addition to the Command Manager:

```
//File manager instance
Media::Filecom filecom(commgr, myaddr, discovery.get_status().addr, 5.0F);
//Configuration manager instance
Base::Cfgmgr cfgmgr(commgr, myaddr, discovery.get_status().addr, 5.0F);
// IRX addition to Command Manager. This action allows to listen messages
// which IRX is Base::filecom_id and Base::config_id, associated to file
// management and configuration management respectively
commgr.add_irx(Base::filecom_id, filecom);
commgr.add_irx(Base::config_id, cfgmgr);
```

Uploading routes configurable:

```
//Random identification number of the file transaction
Uint64 uuid_file = rand();
//File configuration sending and saving configuration
//67 will be interpreted by Veronte as a route change request
if (filecom.upload(route, 67, uuid_file) && cfgmgr.save(uuid_file, 67))
{
  bool sending_file = true;
  while (sending_file)
   {
      VCP::send_receive(serial, commgr);
                                          //Serial bus refresh
     if (!filecom.step() && filecom.is_successful())
        //True when the whole file messages transactions have finished
        sending_file = cfgmgr.step();
      }
  }
}
```

Uploading feature references configurable:

```
//79 will be interpreted by Veronte as a feature references change request
if (filecom.upload(fop, 79, uuid_file) && cfgmgr.save(uuid_file, 79))
{
    bool sending_file = true;
    while (sending_file)
    {
        VCP::send_receive(serial, commgr);
        if (!filecom.step() && filecom.is_successful())
        {
            sending_file = cfgmgr.step();
        }
    }
}
```

Finally, change to phase 0 to test this command using the command manager instanced in *Discovery step*.

```
build_phasecmd(cfgmgr, 0); //Builds change phase command
bool sending_cmd = true;
while (sending_cmd)
{
    VCP::send_receive(serial, commgr);
    sending_cmd = cfgmgr.step();
}
```

5.2.1 5.2.1. Command Prompt

File configuration output:

eronte CMD test on port: \\.\COM7	^
Initializing Discovery	
Success Discovery, UAV Address = 1571	
bening file	
PENED	
ending	
ending	
ending	
losing file	
LOSED	
bading file	
aving file	
aving file	
AVED Discourse and the second s	
SUCCESS	
ANALY AND ANALY AND ANALY AND ANALY	
nening file	
anding	
ending	
ending	
losing file	
LOSED	
bading file	
aving file	
aving file	
AVED	
SUCCESS	
aanging phase	
ending cmd	
SUCCESS SENDING COMMAND	
	×

5.3 5.3. Telemetry

Telemetry is sent by default by Veronte devices, this means it is not necessary to create a particular packet to send. However, it is essential to add the IRX to command manager as in *filecom example*, but in this case the telemetry IRX will be added:

```
// Command manager where add irx (This allows to manage received messages)
Base::Commgr commgr(source_address);
// Telemetry IRX instance
Telemetryrx irx1(0, source_address);
// Add telemetry irx (when a telemetry irx arrives will be routed to
// on_rx function of Telemetry class)
commgr.add_irx(Base::telemetry_id, irx1);
while(true)
```

(continues on next page)

(continued from previous page)

```
// Update serial bus to check if packets are arriving
       VCP::receive_step(serial, commgr);
// Close serial interface
serial.close();
```

Launching this example will show the following message in the command prompt:

Packet received from UAV XX to UAV XX - Hash: XX - Timestamp: XX

or

{

}

Packet ignored from UAV XX to UAV XX - Hash: XX - Timestamp: XX

depending on address destination field of telemetry packet received.

5.3.1 5.3.1. Command Prompt

Telemetry output:

Veronte	e Telemet	ry fro	om: `	\\.\0	COM7							~
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5105.645996	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5105.747070	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5105.848145	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5105.949219	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.050293	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.151367	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.252930	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.354492	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5106.455566	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.557129	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5106.658691	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.759277	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5106.860840	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5106.962402	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.062988	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.164551	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.265137	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.367188	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.468262	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.569336	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.671387	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.772461	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.873047	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5107.974609	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	0xBB3FD528	Timestamp:	5108.076172	
Packet	received	from	UAV	999	to	UAV	2 -	Hash:	ØxBB3FD528	Timestamp:	5108.176758	
												- L.