Extending Frontline Products - DecoderScript

Purpose

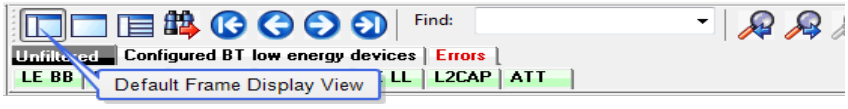
Frontline Test Equipment has a long history of fully supporting all Bluetooth® profiles. Many profiles are supported before the specifications are even fully written. That support has worked well with BR/EDR but *Bluetooth* Smart™ is a much faster moving technology. we fully support all profiles being developed through the *Bluetooth* SIG but what about proprietary work being done outside the auspices of the governing body?

Frontline offers the developer the ability to extend the Frontline product by using DecoderScript™ . Some features of this capability are:

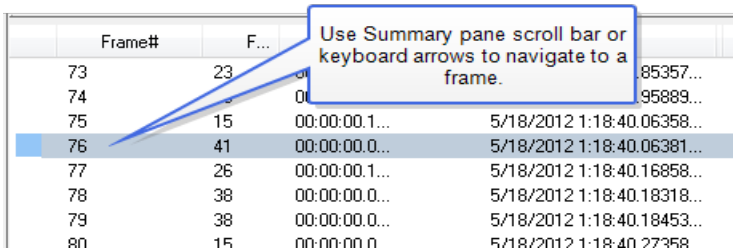
- Access to exactly the same development platform used by Frontline for its own decoders.
- A fully developed and easily learned decoder language.
- Ability to extend the decoder language with C++ methods.
- Custom code can be kept proprietary. That is, the custom code lives in its own directory.
- Custom code is "future proof". Proprietary code will automatically load in any new version of Frontline products.
- DecoderScript code can be used across the full Frontline product line.

Open the Hello Bluetooth® capture file in ComProbe analyzer

You can download the capture file from the Frontline website: http://www.fte.com/downloads/CFA/Hello_Bluetooth.cfa. Save the downloaded file.



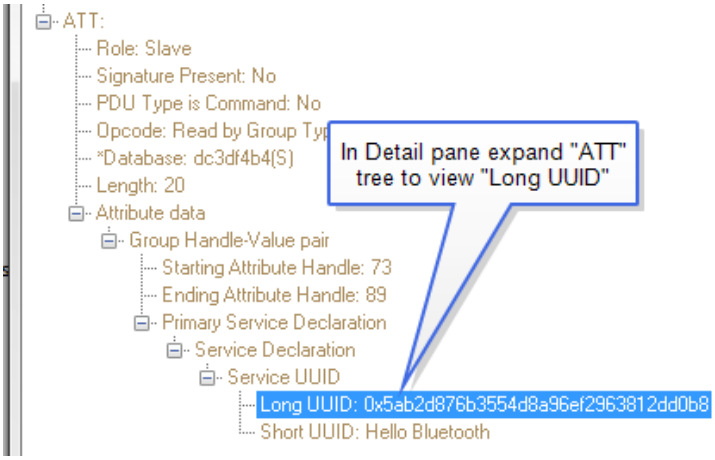
Double click on the .cfa file to open it in an installed ComProbe analyzer, or you can open the capture file from the ComProbe Control Window **File** menu by clicking on **Open Capture File…**. When open, activate the analyzer **Frame Display** with the default panes active. In the Summary pane, navigate to and select frame “76”. In the Detail pane expand Frame 76 “ATT” branch to display the “Service UUID”, “Long UUID”.



In Frame 76, a UUID is mapped to the Hello Bluetooth Service. When the decoder sees the UUID in the frame, it consults a table, “GATT_UUIDs”, to see what it means.

Hello_Bluetooth.cfa in Frame Display

It scans the table and finds the pertinent row.



{ 0x7f000001 "Hello Bluetooth" }

Now it knows that 0x7f000001 means “Hello Bluetooth”. It is that simple.

But wait, where did "0x7f000001" come from? More about that later.

The decoder uses three strategies to recognize a transmitted UUID:

1. For *Bluetooth* SIG adopted UUIDs, only 16 bits must be transmitted. In that case, the value is compared against the table.
2. For *Bluetooth* SIG adopted UUIDs, the full 128-bit UUID can be optionally transmitted. In that case, extract the pertinent 16 bits from that 128-bit UUID and use it.
3. For proprietary UUIDs, the whole 128 bits must be transmitted and evaluated. For various reasons, it is converted to smaller 32-bit value which is used.

This is why we see "0x7f000001". It is a value we created to represent the full 128 bit UUID. For a complete explanation of how that works, see Handling 128-bit UUIDs.

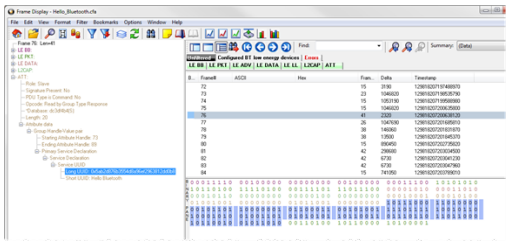
In frame 94, a UUID is mapped to the User Name characteristic. Here the decoder does exactly the same thing. It looks up the UUID in the table and finds a text mapping.

```
{0x7f000002  "User Name"  "User Name  0 chHelloBT_UserName}
```

But there is more in this table entry. It might be necessary to use different text in the ComProbe analyzer **Frame Display Summary** and **Detail** panes, so DecoderScript privides for different fields. The first field is the **Summary** pane text and the second field is the **Detail** pane text. If there is nothing after the **Summary** pane text, the decoder will use that text in both the **Summary** and **Detail** panes as it did for the service name. Here the same text is provided in both fields as a place holder for what comes after. The next field, the"0", is for advanced users and is seldom used. The final field is where things get interesting. This the name of a DecoderScript GROUP that will becalled to decode this data. More on that field a bit later.

So to summarize what is going on with this table entry:

When we see 0x7f000002, "User Name" will go in the Summary pane, "User Name" will go into the Detail pane and we do not care about the "0". Finally, when we get some real data associated with this UUID, we will call chHelloBT_ UserName to decode it.



Hello_Bluetooth Capture File in ComProbe Frame Display

Remap the handling of the Hello Bluetooth® Service UUID

Create a file called "Custom Attributes ATT.dh" in the "My Decoders" directory. In it, paste the text below.

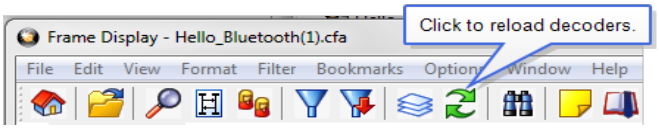
```
TABLE tMyTable APPENDS_AND_OVERWRITES GATT_UUIDs
{ 0x7f000001 "My Service" "My Service" }
ENDTABLE
```

This simple code is remarkably powerful.

- The ATT decoder is configured to look for a file with this exact name. If it exists, whatever is in this file becomes part of the ATT decoder. If it does not exist, the decoder does not care. The file must have a specific name and be in a specific place for this to work.
- Your table may be named anything unique. That is, tMyTable can be whatever you want as long as it does not conflict with any Frontline table names. ComProbe analyzer will tell you if there is a problem.
- APPENDS_AND_OVERWRITES means that whatever is in your table will replace what is in the Frontline table.

  Best practice is to include your company name or initials in your table name which would eliminate any chance of conflict.
- GATT_UUIDs is the name of the Frontline table.

To summarize what's happening here, you are replacing the entry for 0x7f000001 (Hello Bluetooth) with your own interpretation. Reload the decoders and you will see that frame 76 now shows "My Service" instead of "Hello Bluetooth".



Remap the handling of the User Name UUID

Add this line to your table:

```
{0x7f000002  "MyData"  "MyData"  0 gMyCharacteristic}
```

Now you are remapping 0x7f000002 which is the UUID for the User Name characteristic. Remember, you have provided text for (in order) the Summary Pane and the Detail Pane. We don't cover the 0 here. Finally, there is that handler GROUP.

Now add this code to your file below the table.

```
GROUP gMyCharacteristic
{
```

```
    FIELD MyCharacteristic (ToEndOfLayer) (StringOfAscii) "MyData"
}
```

Reload the decoders. You have now taken over complete control of the decoding of the 0x7f000001 characteristic.

At this point, your file should look like this:

```
TABLE tMyTable APPENDS_AND_OVERWRITES GATT_UUIDs
{ 0x7f000001 "My Service" "My Service" }
{ 0x7f000002 "MyData" "MyData" 0 gMyCharacteristic }
ENDTABLE
GROUP gMyCharacteristic
{
    FIELD MyCharacteristic (ToEndOfLayer) (StringOfAscii) "MyData"
}
```

Where can we go from here?

Frontline DecoderScript is a fully functional language for creating custom decoders or for modifying existing decoders. With DecoderScript you can expand and customize the functionality of your ComProbe protocol analyzer.

To complete your extension of the ATT decoder, you may want to add your own error codes. Once you know the name of the table, you are free to do whatever you need. In the sample table below, you can see that we have added an entirely new error code for 0xb0. We have also overwritten the 0xfd error code to flag it as Deprecated for our particular product.

```
TABLE tMyErrors APPENDS_AND_OVERWRITES error_code
    {0xb0 “Some text I want to display”}
    { 0xfd "(Deprecated) Client Characteristic Configuration Improperly Configured for this
        Service" }
ENDTABLE
```

Your next step is to customize and enhance the decoding of your data to fit your unique needs. As always, Frontline Technical Support is standing by to help you.

Author: Roger Feeley, Edited by John Trinkle

Publish Date: February 2016

Revised: March 2017