TELEDYNE LECROY
Everywhereyoulook™

DecoderScript™ Wizard

Frontline® Quick Start Guide

Revision Date: 6/27/2016

Contents

Chapter 1 Purpose

This document is designed to get you started creating protocol decoders using Frontline's proprietary DecoderScript language. DecoderScript can be a bit intimidating at first, but once you understand some basic concepts, it's remarkably simple. In the most direct way, this document and the accompanying wizard will walk you through the mechanics of creating a very simple protocol decoder.

When we say a "simple" decoder we mean it. As a programmer you probably can not count the number of times that some simple business need turned into a major development effort. The worst is when you are working on a project and you just know there is a way to do some simple thing and it takes forever to discover how. Many times all you need is a well directed hint; something to get you started. The hint might be the name of the API needed to do a thing or a simple example.

One of the best hints is a simple, "hello world" program that provides a needed development shell. That's what we want to do here, provide that "shell". By itself, the shell is not going to be very useful. As a framework for your own modifications, however, it could save you a lot of time.

This document is divided into 4 sections:

- Introduction
- Run the Wizard
- Have a look around
- Debug your methods

Chapter 2 Introduction

2.1 Who is this document for?

This document is for the programmer who wants to learn how to write decoders using Frontline's proprietary DecoderScript language. As a proprietary language, DecoderScript is going to be a little different than other procedural programming languages. But it is similar enough that experience with any other procedural computer programming language is going to make your job easier. There is one requirement, however, that you should be aware of. To write methods, which you almost certainly will do, you must have some experience with C/C++.

2.2 What do I need?

- Basic understanding of your protocol
- Installed version of some Frontline analyzer
- Visual Studio 2008
- A text editor. Notepad will work. At Frontline we use UltaEdit and we have developed some very helpful tools to go along with it.
- A capture file to test your protocol decoder

2.3 What will this document DO for you?

- Provide you with samples of commonly used methods
- Help you use the Frontline product to build a methods project
  - Create the project with various methods
  - Debug the methods
  - Navigate the project with Visual Studio 2008

2.4 What will this document NOT do for you?

As we noted earlier, this is not intended to guide you all the way through to a finished decoder for your protocol. Instead you will spend a very short amount of time creating your decoder using a wizard. Then you will use Visual

Studio 2008 and a text editor. You will spend some time getting familiar with debugging the methods and moving your methods to a safer project directory. Once you have completed some basic programming, you will toss this document aside and start some serious programming using the DecoderScript manual as a guide.

The first thing we are going to do is define some basic terms. These include **Decoders**, **Methods** and **Frame Recognizers**.

Chapter 3 Decoder Script Quick Start

3.1 Decoders

3.1.1 What are Decoders?

As you know, most communications data is composed of frames. A decoder, then, is nothing more than a way to break down a communication frame into its component parts. These parts are defined in plain, simple English.

Imagine a frame of one byte of value hex 25.

Let's look at a single line of DecoderScript:

```
FIELD myField (fixed 1 byte) (hex) "This is my Field"
```

The Output from that would be:

```
This is my Field: 0x25
```

Two basic things we did here. First we defined a **FIELD** with a name of 'myField' that is a fixed length of 1 byte. Second, we instructed the Frontline product to display that field in hex with a label of "This is my Field".

Basically, a Decoder is nothing more than a whole bunch of that same sort of stuff. As you can imagine, it's not all that simple, but that's the essence of it.

3.1.2 Where do the Decoders live?

If you are working in Windows XP, your Decoders can be found at

C:\Documents and Settings\All Users\Documents\Frontline Test Equipment\My Decoders

If you are working in Windows 7, your Decoders can be found at

C:\Users\Public\Documents\Frontline Test Equipment\My Decoders

3.1.3 How do I edit a Decoder?

This could not be simpler. You can use Notepad or any other text editor. At Frontline, we really like UltraEdit and have written a language extension for it that will make your life even easier.

Whatever you choose to use, you can edit your decoders while you are running your Frontline product and use the Frontline product to check syntax. Simply reload the decoders and you can see your changes right away.

3.2 Methods

3.2.1 What are Methods?

DecoderScript is based on methods. Let's take, for example, our sample line of DecoderScript from the previous page. There we have two methods.

FIELD myField (fixed 1 byte) (hex) "This is my Field"

In parenthesis, we specified the length of our field as fixed at one byte. **fixed** is a SIZE method. It takes two parameters. The first is a number and the second is an enumerated word (**byte**). It evaluates those two parameters and returns the number of bits that is the size of the field.

Similarly, in parenthesis, we specified the format of our field as hex. **hex** is a FORMAT method that receives our field data (length defined by the SIZE method) as input and spits out the text '0x25'.

There are many methods that come standard with the language. In addition, you can write your own and Frontline has taken steps to simplify that process. Frontline's environment wants to separate you from the technology and let you concentrate on what you really need.

3.2.2 Where do Methods live?

If you are working in Windows XP, your Methods can be found at

C:\Documents and Settings\All Users\Documents\Frontline Test Equipment\My Methods

If you are working in Windows 7, your Methods can be found at

C:\Users\Public\Documents\Frontline Test Equipment\My Methods

3.2.3 How do I edit a Method?

Here you really need Visual Studio. Frontline products will create a Visual Studio project for you. You will edit the .mth file. You won't touch the .cpp file that really gets compiled. You can reload methods on the fly, but it can be easier just to restart your frontline product.

You can debug methods in Visual Studio and set breakpoints in the .mth file.

3.3 Frame Recognizers

3.3.1 What is a Frame Recognizer?

A Frame Recognizer is a special kind of method. It tells the Frontline product where the start and end of frames are. If you are working with Serial data, you will certainly have to have one. The section in the DecoderScript manual on Frame Recognizers is excellent, but here are the basics.

A Frame recognizer is a function that gets called for every character. It's generally a state machine that keeps track of where it is in the frame. Its only job is to keep track of where it is in the frame and return a code to the Frontline product. Most of the time, it will return eContinue, which means keep going. When it detects the start or end of frame or an error, it will return something else. It is these return codes that tell the Frontline product what to do with each character.

See the DecoderScript manual for a full list of these return codes

Now that you understand a little of the terminology, it's time to creating a simple decoder with the Wizard.

3.4 Run the DecoderScript Wizard

3.4.1 What it is for and what it is not for

This wizard will not produce a final product. What it will do is help you understand what you need to do to get to the final product. It will produce a working decoder and frame recognizer. It will also produce a personality file that will make it look like part of the Frontline product. It's up to you to customize it to your business needs.

There are four main sections to the Wizard. You can follow along with the explanation below as you move through the Wizard.

3.4.2 Collecting data about your computer

This is just the wizard finding its way around. It exists to make the process of finding the proper places to put your decoders and methods as transparent as possible. Also, it could be very useful if you need telephone support.

The wizard supports only the default locations for Frontline files. If you have changed those defaults, your Frontline product won't see the wizard generated files.

The root directory answers a frequent question by customers about where custom decoders and methods live. The working directories section should have four entries. The presence of these indicates that the wizard found the necessary subdirectories.

3.4.3 Tell us about your Decoder

This section is where you start entering information. The wizard will provide you with a Decoder name and Decoder ID. The name can be anything you want. The ID must be less than 0xFFFF.

Warning: If you generate two decoders with different names but the same decoder ID, the frontline product will give you an error. That's why the wizard won't let you do that.

3.4.4 How is your protocol framed?

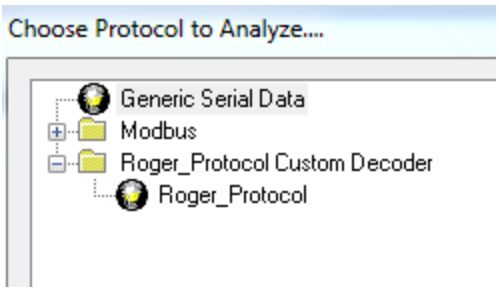
The final section is the most involved. None of the recognizers are meant to be used, 'out of the box'. They are just shells with the right code in them to help you on your way. The decoder will work with any of them. Choose the one that most closely describes your protocol.

Strictly speaking, you do not actually have to do anything but click the Submit button. You can change the name of the protocol later. You can change the Protocol ID later. You will almost certainly want to rename the methods later.

For this initial exercise, let's assume that you just click Submit without changing any of the settings.

3.4.5 Run the Frontline product to compile the methods

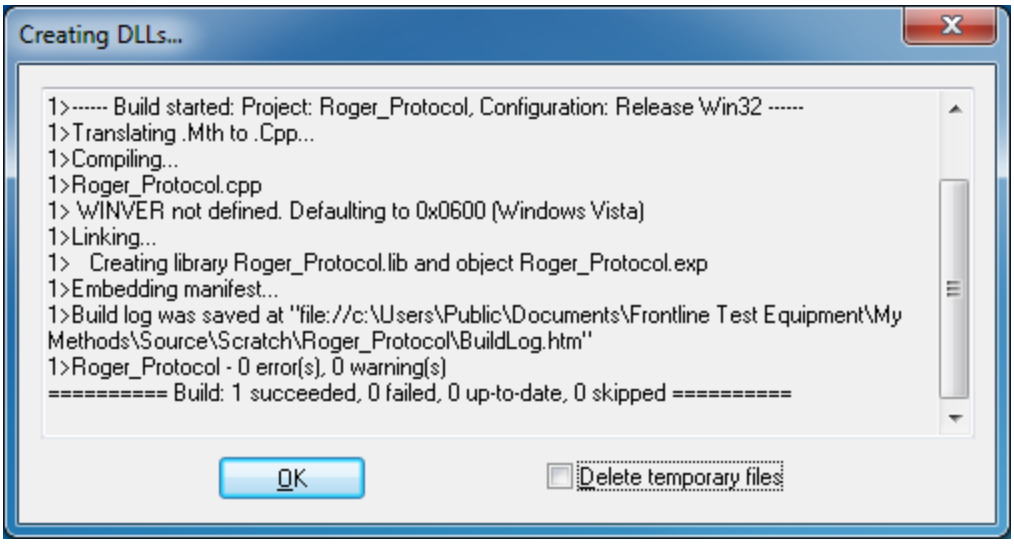
The next step is to run a Frontline product. There should be a new protocol. Because of the Personality file, you see a folder for your protocol on the startup screen.Place your text here.



Expand the folder and you should your protocol. Select that and click Run. Make sure you select the right data capture method.

When you click **Run** you get a dialog asking you to build methods. Click **Yes**.

The Frontline product will build your methods. Uncheck the 'Delete temporary files' checkbox to keep your project.



At this point, go back to the Wizard Results screen. You will see that the 'Edit all.sln' button has been enabled. If you click it, your Visual Studio should start up with the project.

You are done with the wizard. If you like, you can go back and forth between the wizard form and results screens and generate the project as many times as you want.

3.4.6 The Results Screen

Here the wizard tells you what it did. It will have validated the Protocol Name, Decoder ID and the SOF and EOF characters. It also scanned all of the custom decoders for the decoder ID you specified and found no conflicts.

In addition to the validations, it generated three files for you: A decoder, a methods file and a personality file. It also gives you a link to the all.sln file but the link will not be enabled yet. For that, you have to proceed to the next step. Keep the results window open. You will be back.

3.4.7 Have a look around

Now that you have used the Wizard to create your own decoder and methods, let's look at the results.

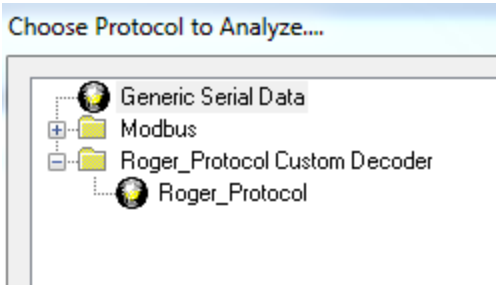
Using UltraEdit, open your decoder file (xxx.dec in My Decoders). At this point, you should use the DecoderScript manual as a guide but here are the basics:

- You will see your decoder name and ID at the top.
- You will see a line instructing the decoder to use your recognizer.
- The word DECODE begins the main Decoder.
- END_MAIN_PATH ends the main decoder.

- Everything below END_MAIN_PATH would be for Groups and Fields that are called by the main path. Think of them as subroutines.

Using UltraEdit, open your personality file (xxx.personality). Again, the DecoderScript manual is your best resource but here are some highlights.

- The personality file defines the way your decoder integrates with the Frontline products. In the screenshot below, you see, “Roger_Protocol Custom Decoder”. Under [Common] in the personality file, you will see a path that defines that text.
- The following entries in the personality file are for various Frontline products. Each modifies the characteristics laid out in [Common]. This file insures that your protocol will appear in each Frontline product.



To see the methods, open all.sln in Visual Studio. Look at the xxx.mth file which is the only one you would edit. Again, the basics:

- You will see a Recognizer with the same name that was in the decoder.
- Notice that it has a private section where you can put variables that would persist between calls.
- It has a RESET section that is called whenever a new capture is started.
- The EACH_BYTE is where the real work is done.
- You will see methods that correspond wit the method call in the decoder. Each of these is as simple as possible. See the DecoderScript manual for making Methods that are useful. In particular, look at Method inputs and outputs.

3.5 Debugging Your Methods

Although the methods will run, they are not of much use. You will need to use the generated methods as a basis for your own methods. When you finish your coding, you will almost certainly need to debug them with Visual Studio. To do that, you will need to make a few changes to all.sln.

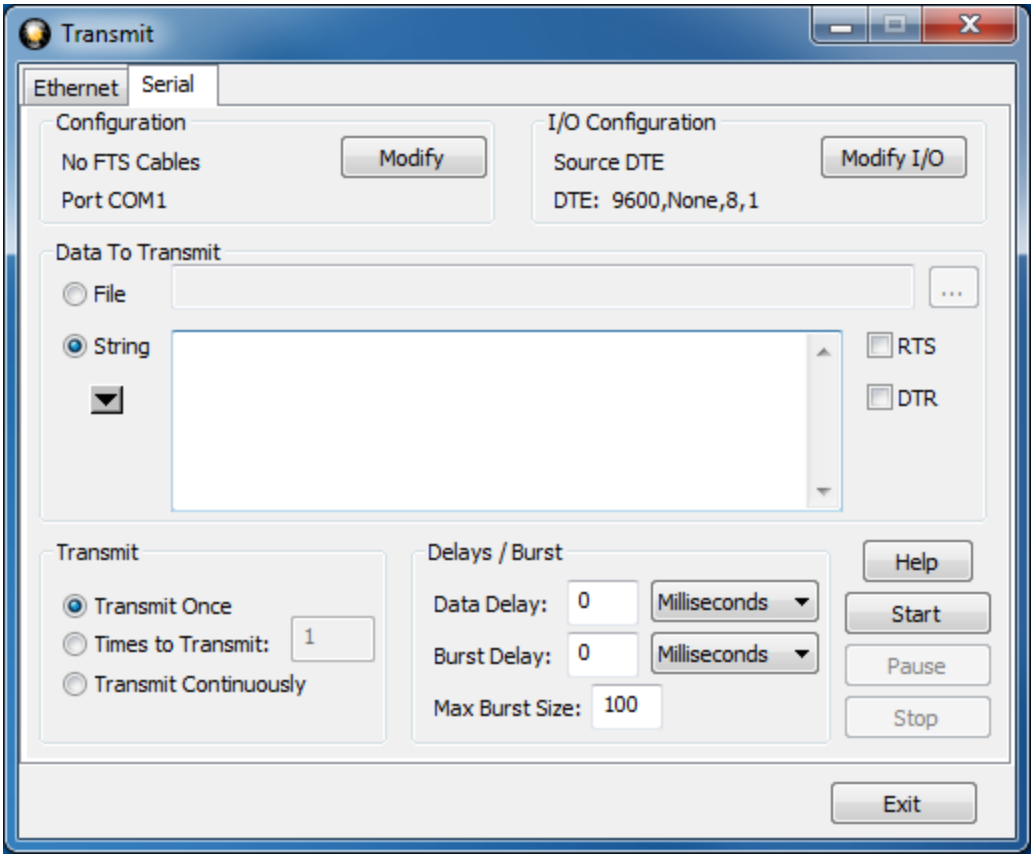
- Edit the Project Properties:
  - Project/Properties/Linker/Debugging/Generate Debug Info = Yes
  - Project/Properties/c/c++/Preprocessor = _DEBUG;_AFXEXT;WIN32;_WINDOWS
  - Project/Properties/c/c++/Browse Information/Enable Browse Information =
  - Include All Browse Information (/fr)
- Recompile your project
  - Run Frontline product

- Attach to fte.exe
- Debug/Attach to Process/ Click on FTS.exe
- Place a breakpoint in your code. Alternatively, use the macro, 'ASSERT(0). Use 'ALWAYS_ASSERT_ONCE;' if you want to just do the assertion one time.
- If you are debugging a Frame Recognizer, Frame Transformer or Data Object, delete the frame (.frm) file from your capture files directory. It's a brute force way to force the application to call your code.
- In the Frontline product, open your capture file.

3.6 Tips and Tricks

3.6.1 Creating dummy data

You probably have real data for your protocol, but you might like to transmit something into the decoder created by the wizard just to see it work. Here we use a Frontline utility to transmit dummy data to Serialtest® Async and view it with the decoder we just created with the Wizard. Serialtest Async ships with an excellent utility called Transmit Data. Start up the program and the default screen should look like this:



- In the **Configuration** section, click on **Modify** button and choose the correct COM Port number that you are using to transmit serial data.
- In the **I/O Configuration** section, click on **Modify I/O** to set the serial transmission configuration to **Source DTE** and **9600 baud,None,8,1**.

- Select the **String** radio button and in the text box, enter a string that looks like this:

  $01$0cHello there$03

  The $01 is the Start Of Frame (SOF) character.

  The $03 is the End of Frame (EOF) character.

  The $0c tells the decoder that there are 12 bytes to follow. This data will work with the SOF/EOF or the SOF/Length recognizer.

Start up Serialtest and choose your decoder. Click the **Capture** button (the red circle) and select the decode pane. Click the start button in the Transmit Data window.

3.6.2 Help! I can not break into my Method

This can be very frustrating. Here are a few tips:

- In the instructions above, make sure you left the underscore before the DEBUG directive.
- Check the myMethods/release directory. I have changed the name of my mth file and it produced two dlls: one with the old name and one with the new name. Each would have the same exported functions. FTS will load both dlls and behavior will be unpredictable.
- Check to make sure that your methods are uniquely named. If your method happens to have the same name as a standard one, FTS might use the standard one and yours will never get executed.
- Make sure your dll is being loaded. Write a simple FORMAT method that returns a string and invoke it in your decoder. At least you will know that your dll is being loaded.

3.6.3 How do Methods become part of a Frontline product?

You edit a .mth file to create and change your methods. If your methods get big, you may have .mh files and include them in your .mth file. None of those files are used directly by the compiler.

If you right click on your .mth file and select properties and look at the custom build step, you will see a call to Mth2Cpp.exe. This application is bundled with every Frontline product. Generally, it's the same thing so if you have multiple products, it doesn't matter much which one you use. When the Frontline application built your project, it used the copy closest at hand.

Mth2Cpp takes your method files and produces the cpp file that is what is really used by the compiler. The output of all.sln is a dll that will go into 'My Methods\Release'. When the Frontline product starts, it looks in that directory and loads whatever dlls it finds.

Warning: Make your method names unique. Consider incorporating your protocol name. Suppose you wrote a FORMAT method called Hex. The Frontline product will happily load it along with its own copy of the Hex method and it's anybody's guess which one will be called.

3.6.4 Viewing parameters in the debugger

You might think that since a parameter can be used as eParam1 in your method code, you can type eParam1 in the watch window in Visual Studio and see it. That's not the case, but there are two ways you can see your parameter data.

1. Declare an intermediate value and assign it the value of the parameter. There is a small performance overhead with that, especially if you are dealing with a frame recognizer where your method will be called

once for every character.

2. Open up the .cpp file and look for your method code. At the top of the function encapsulating your function, you will find a define for eParam1. Put that in your watch window and you should be fine.

3.6.5 Moving your project source to another directory

This procedure is handy when you want to have a Visual Studio project for each methods dll. Moving it away from MyMethods means that you will not accidentally regenerate the project and have to customize it again. It also facilitates Version Control.

1. Copy and paste the whole project to another subdirectory.
2. In the Solution Explorer, remove both XXX.mth and XXX.cpp from the project. Don't delete the files, just remove them.
3. Re-add the files. For the method, you will be prompted to build a rule. Just cancel.
4. Open up the properties for XXX.mth and navigate to the "Custom Build Step"
   a. Under Command Line, enter

      C:\Program Files(x86)\Frontline Test System II\Frontline ComProbe Protocol Analysis System <versioin #>\Executables\Core\Mth2Cpp.exe" "$(InputPath)" "$(ProjectDir)"

      Make sure the directory under Frontline Test System II matches whatever Frontline product you have.

   b. Under Description enter "Translating .Mth to .Cpp…". Don't include the quote marks.

   c. Under Outputs, enter "$(ProjectDir)\XXX.cpp". No quote marks. Make sure to replace 'XXX' with the proper name for the CPP file.

   d. Under project/properties/Configuration Properties/'C/C++'/Additional Include Directories, fix the entry to point to your new path so you can have .mh files. Example:

      C:\Users\Public\Public Documents\Frontline Test Equipment\My Methods\Source

Note: Note: you may have to modify step 4a to reference the correct FTS directory to get the correct version of Mth2CPP.exe.

5. Do a Clean Solution to remove all temporary files.
6. Do a rebuild of the project to insure that the dll shows up under MyMethods.

Technical Support

Technical support is available in several ways. The online help system provides answers to many user related questions. Frontline's website has documentation on common problems, as well as software upgrades and utilities to use with our products.

Web: http://www.fte.com, click Support

Email: tech_support@fte.com

If you need to talk to a technical support representative, support is available between 9am and 5pm, U.S. Eastern time, Monday through Friday. Technical support is not available on U.S. national holidays.

Phone: +1 (434) 984-4500	Fax: +1 (434) 984-4505