Product File Creator Tutorial

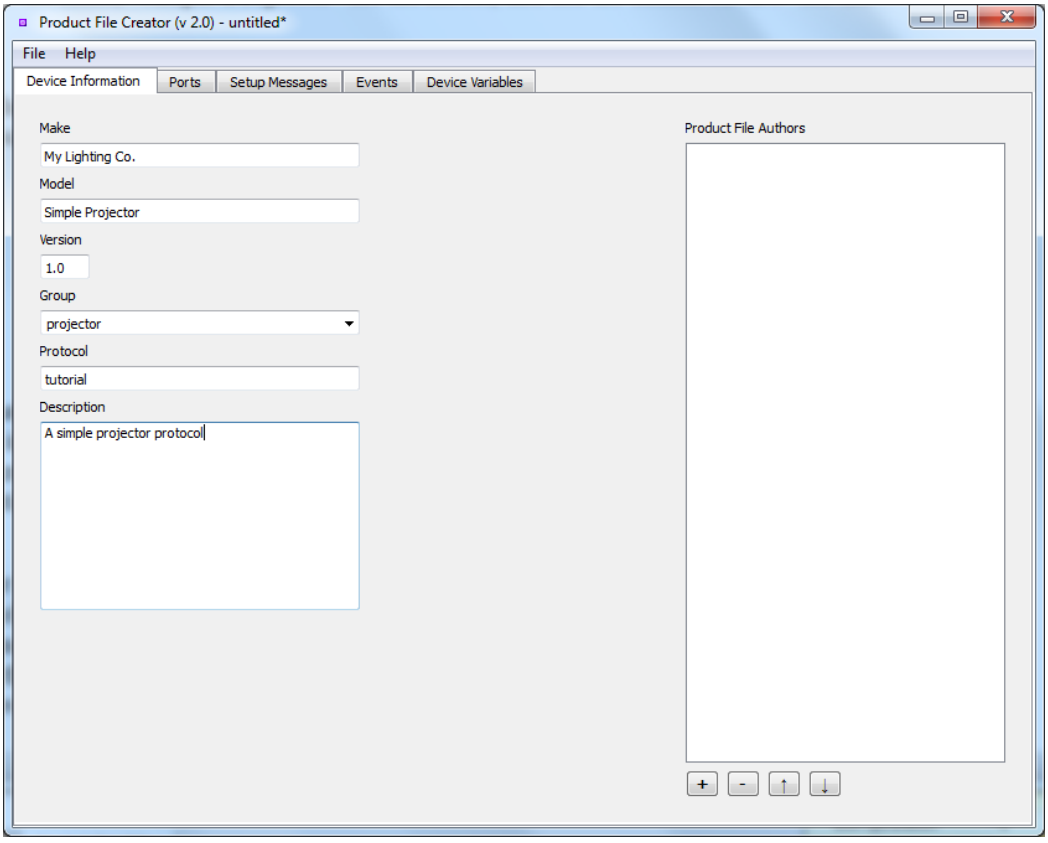
The Product File Creator Tool can be found under the **"Tools" menu of WinScriptLive**. It can be used to create custom product files that can be added to and used in WinScriptLive. In this example we will be creating a simple, limited product file for a Christie HD10K-M projector to allow it to be powered on and off from WinScriptLive and to allow for the online time of its lamps to be gathered into variables.

Getting Started

When launching Product File Creator, you will be presented with a splash screen allowing you to create a new product file or open an existing one. For convenience, a list at the bottom of the window also shows all compatible files in the **"WinScript Live\My Product Files"** subdirectory of your **My Documents** folder.

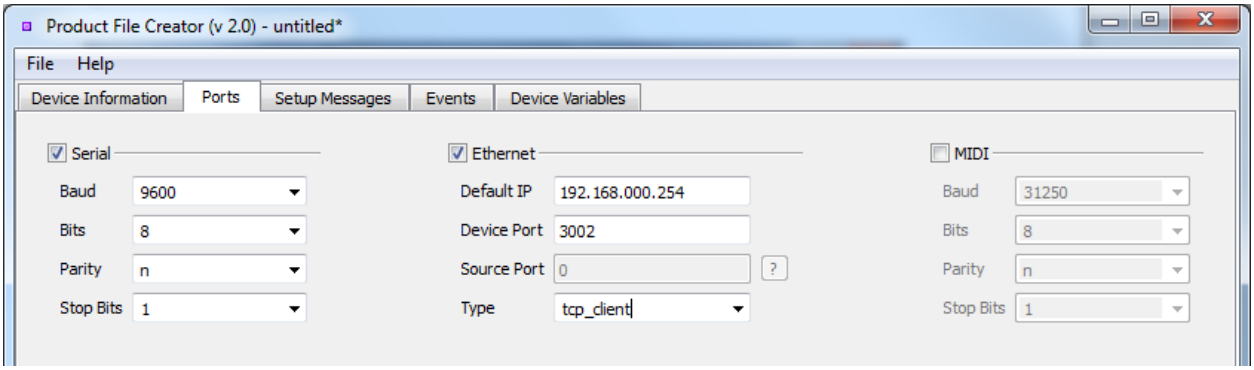
When creating or opening a product file, you will be asked to put in the **Make**, **Model**, and **Version** of the product file you are creating. This info is what WinScriptLive will use to identify and differentiate between different product files and will help us load the product file for our script later on. Since this is a demo, put in "**My Lighting Co.**" for the make, "**Simple Projector**" for the model, and use the default of "**1.0**" for the version number.

Select "projector" from the **Group** dropdown menu so that the device will be classified as a projector. You can create different protocols for the same device if you like, but for this example simply enter "tutorial" for the **Protocol** field.



On the next tab, check that the device has a serial port. You will then see fields for specifiying the communication protocol information for the serial port, which can be edited depending on nature of the device you are trying to create a product file for. Leave the fields as their defaults, as the defaults outline accurately how the Christie HD10K-M commincates with its serial port.

There are also options for the Ethernet port. Here you can fill in the various port fields according to the hardware specifications of your device. For our purposes, fill in the Default IP with "**192.168.0.254**", the Device Port as "**3002**", and the Type as "**tcp_client**". We choose "tcp_client" because our Show Controller will be initiating the TCP connection. Had we chosen "**tcp_server**", we would be indicating that the external device would have to initiate the connection. The projector does not use MIDI so leave that option unchecked.

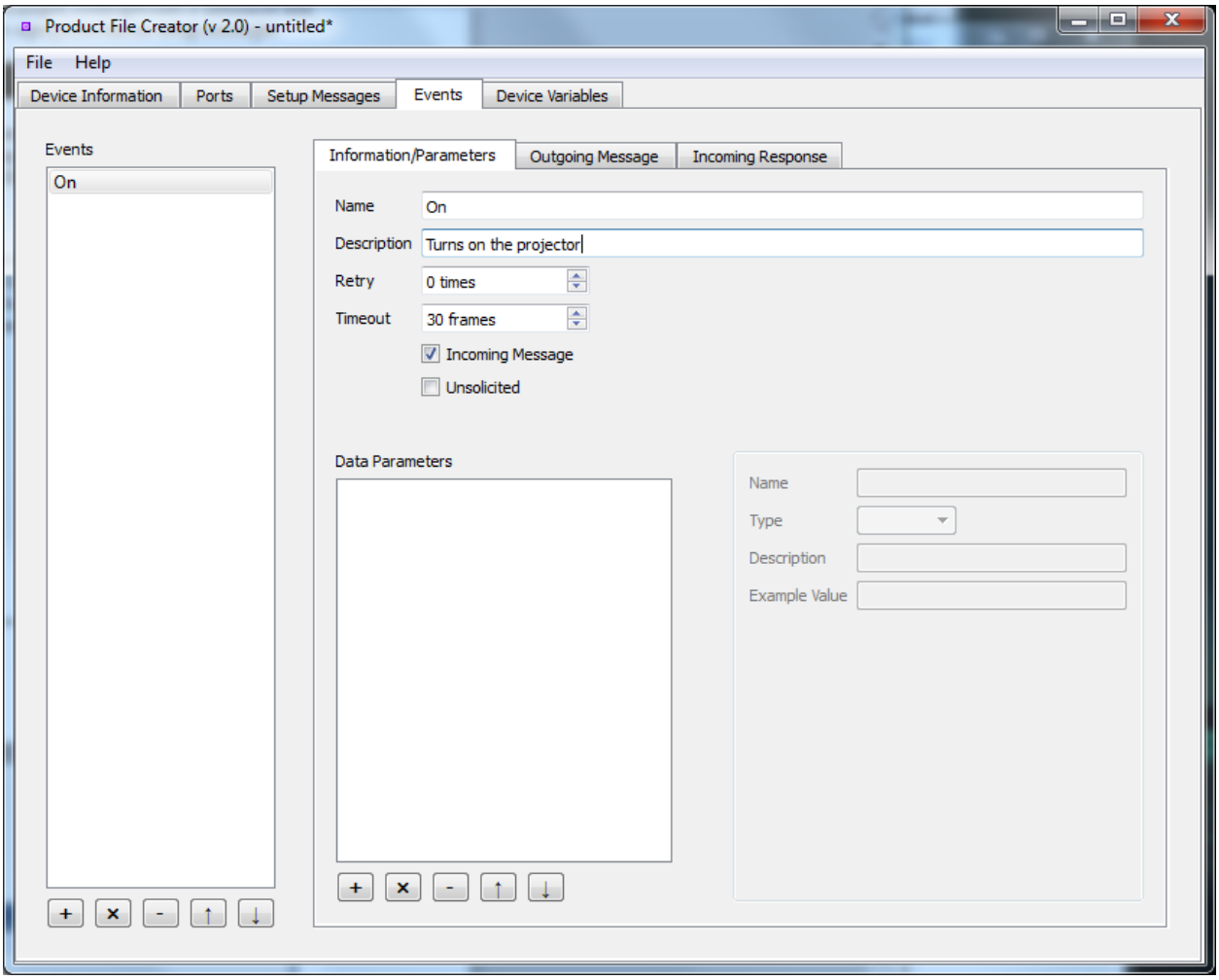


The next tab is "Setup Messages". Some TCP protocols require a series of "login" messages to take place one time before the show controller can send commands to the device. These setup messages take place immediately after a TCP connection has been made. Telnet, for example, often specifies a username and password login process. We will skip this tab in this tutorial since a projector has no setup messages.

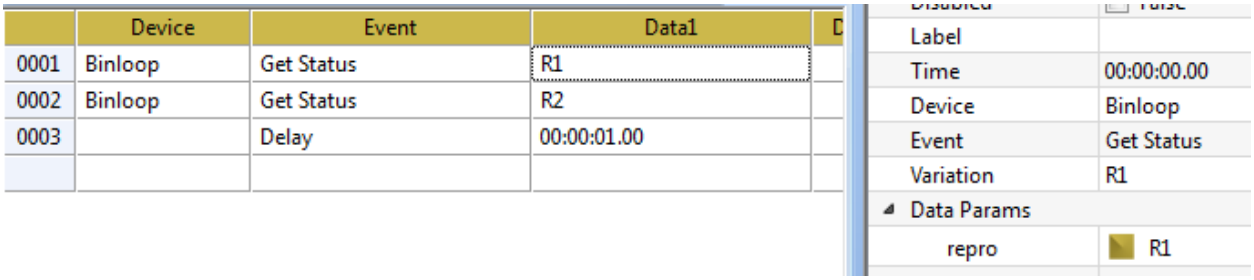
Adding Events

Next go to the Events tab. This tab is used to add the various events a device can execute. These events show up in the "Events" section of WinScript Live when specifying which action you want the device to perform. In this tutorial, we are going to add an "On" event, an "Off" event, and events to get information about how long the projector's lamps have been on.

Click on the **Add Event** button and type "On" for the Name of the Event. Any description you add will be used in WinScriptLive alongside the Event name. According to Christie's documentation, the Power On command can send an acknowledgement of success. This would be useful. Click the "**Incoming Message**" check box so that we can set up a filter for such a response.



Some Events in WinScriptLive have data parameters used to control the contents of an outgoing message (for example, the "Get Status" event in this sequence has a parameter called "repro"):



	Device	Event	Data1
0001	Binloop	Get Status	R1
0002	Binloop	Get Status	R2
0003		Delay	00:00:01.00

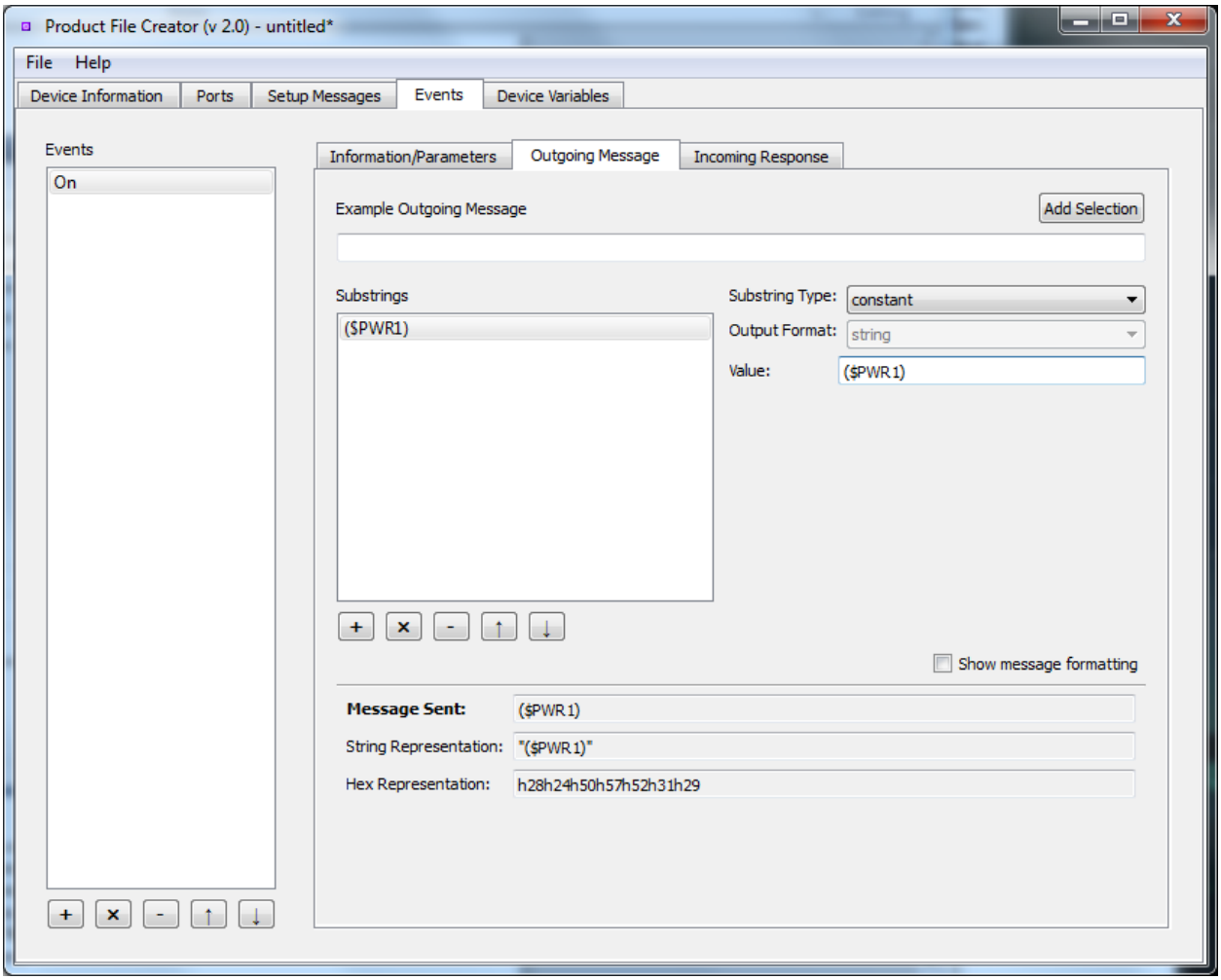
Label	
Time	00:00:00.00
Device	Binloop
Event	Get Status
Variation	R1
Data Params	
repro	R1

If our Event had any parameters, they would be defined here. Our event does not have any parameters, so go on to the "**Outgoing Message**" section.

Outgoing Messages are sent from your Show Controller into the device being controlled. Some of these commands are complicated combinations of Device Variables, data parameters, and even unprintable characters. For this reason, the Outgoing Command section allows you to build a message as a combination of substrings. The substrings can be of type "constant", "parameter", "variable", or "checksum".

There are two main ways to add to substrings on this tab. You can either: 1) highlight parts of an example Incoming Response and specify what role those parts play in the final message; or 2) directly add substrings to the final message without using an example message. Luckily for us, the command to turn on our projector is relatively simple, so we will use the second method.

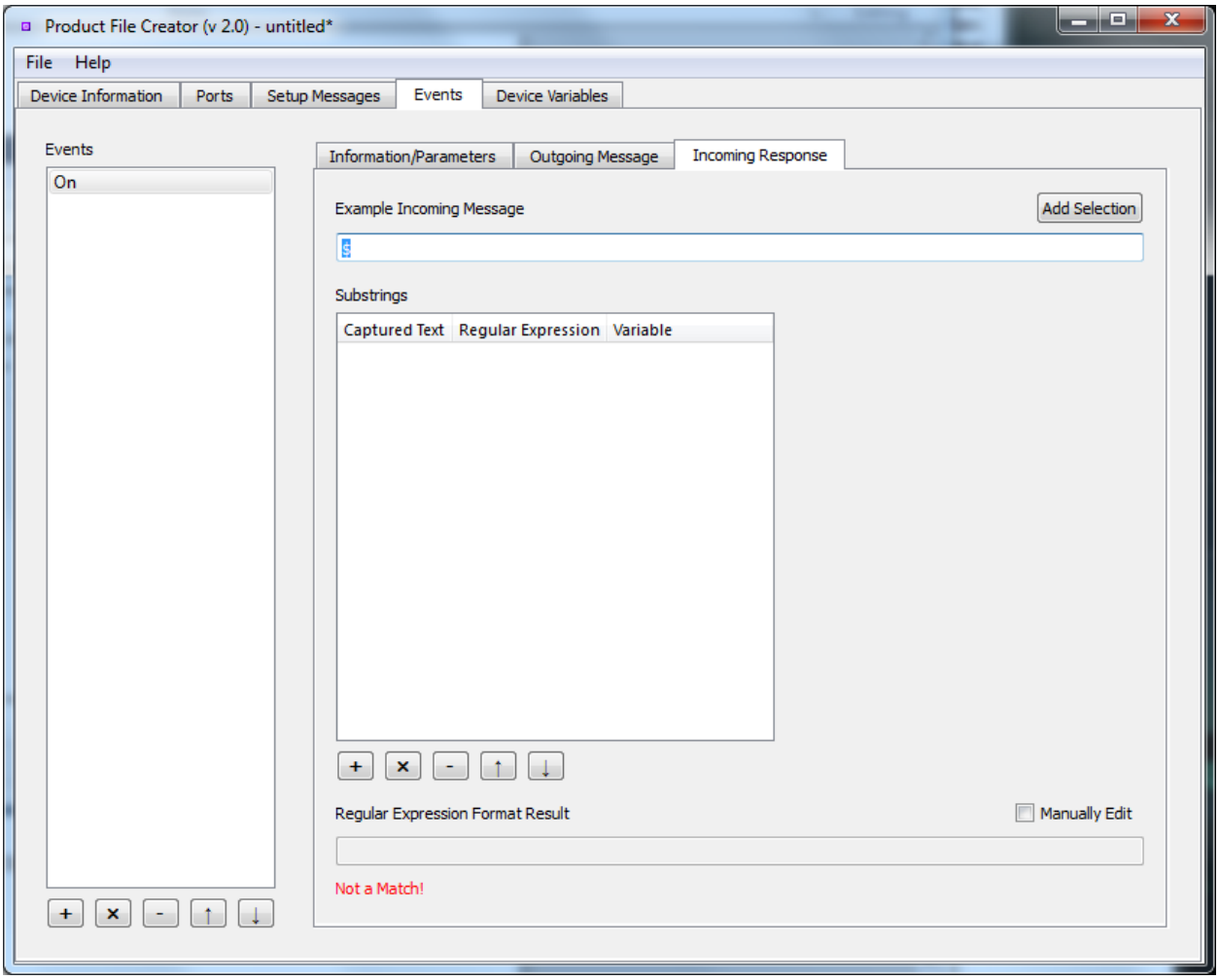
We will make a message from one substring of type "constant". Click on the **Add Substring** button, select "**constant**" as the type of substring to add, and enter "($PWR1)" in the box labeled "**Value**".



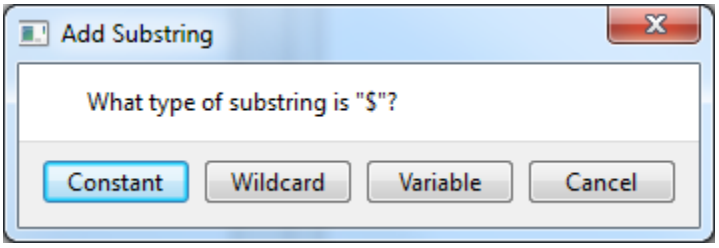
Simple, right? Notice that at the bottom of the window you see both the ASCII String representation and the Hex representation of the Outgoing Message. If our message had any unprintable or escape characters, you would be able to verify them there. Also, had our message contained substrings that used data parameters or variables, the **Message Sent** field would fill in example values to show a complete sample Outgoing Message. If you wanted to, you could click the "**Show message formatting**" box to manually edit the **sprintf** command that is generated for this command.

Now let's specify the form that a valid Incoming Response to this message should take. Click on the **Incoming Response** tab. Like Outgoing Responses, you have the option of building up Incoming Responses as a series of substrings.

The expected Incoming Response to a Power On message to the Christie HD10K-M is simply a "$". Type that into the **Example Incoming Response** box. Example Incoming Responses are usually more complicated than "$", and the reason for having this box will become clearer later. For now, highlight the "$" you just typed and then click the "**Add Selection**" button to the right.



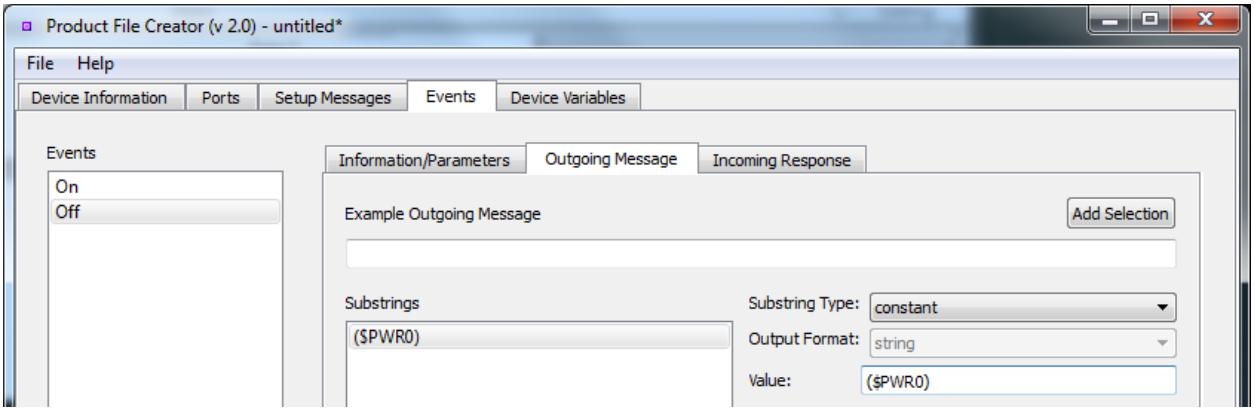
You will be asked what type of substring you are adding. Choose "**Constant**".



At the bottom of the window, you should see "A Match!" underneath the **Regular Expression Format Result**. Incoming Responses are formatted as regular expressions (or "regexp_format"). We will explore this in more detail in a moment, but for now observe that the correct regexp_format was created: "**\$**". If

the Incoming Response from a device does not match the expected regexp_format, your Show Controller's "error" light will come on and the it will attempt to send the outgoing message again.

Now press the **Duplicate (×)** button underneath the list of events. Change the name and description of the newly created event from "On" to "Off", and change the Outgoing Message from "($PWR1)" to "($PWR0)". The Incoming Response will remain the same.



Using Regular Expressions with Event Messages

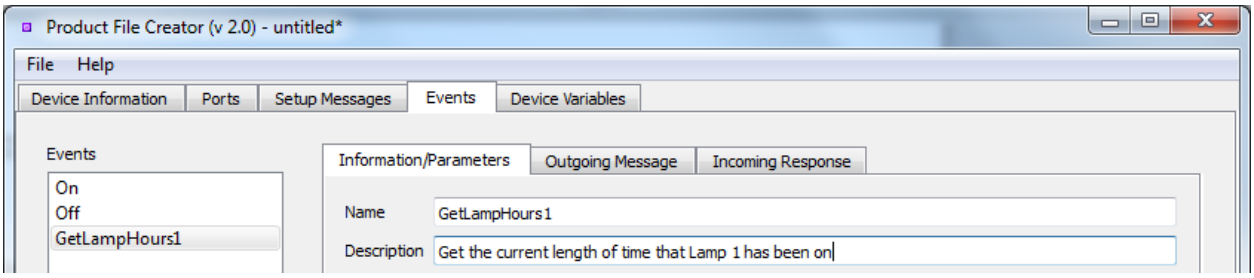
Unlike Outgoing Responses, only the form of a valid Incoming Responses is usually known, not its exact content. For example, you might trigger your Show Controller to send a message to an audio playback device querying the status of a certain track that is currently playing. The Incoming Response from the device might look like any of these:

- "Track 012 – HappyDay3.wav 01:09:59:29; 5m 15s remaining"
- "Track 001 – Song000000001.wav 00:02:14:37; 1h 16m 14s remaining"
- "ERROR 034 - No Track Found".

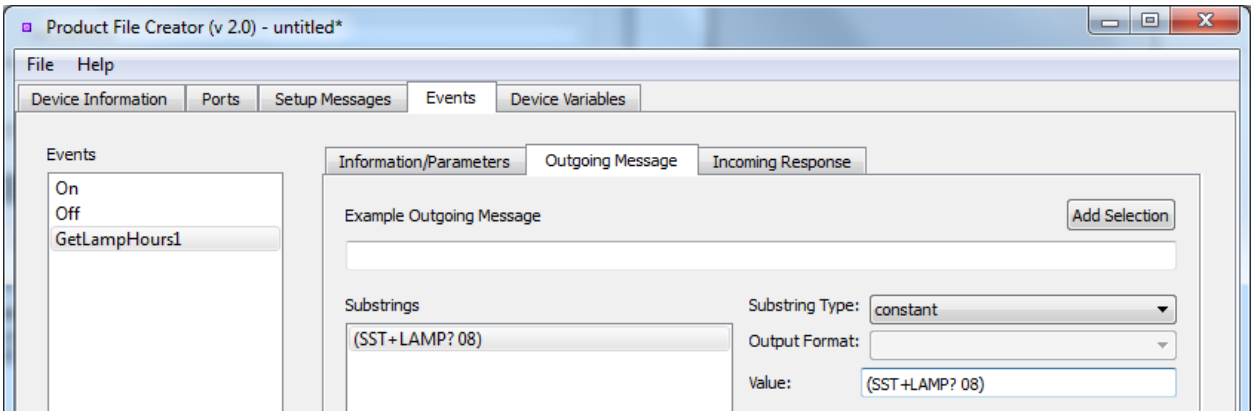
You would want your show controller to know that it got a valid Incoming Response if it got either of the first two messages as a response. You might also want it to match enough of the pattern of the message to store some information from it into device variables or data parameters. In order to provide such capabilities and more, regular expression matching is used for incoming messages in the Product File Creator.

Many great introductions and guides on regular expression matching are available on the internet. We could not do justice to all that is possible with regular expressions in this short tutorial, but the Appendix of this document does have some limited information useful to using regular expression with the Product File Creator and some references to resources. For now, let's illustrate the use of regular expressions by creating an event to gather the number of hours and minutes that the lamps of our projector have been on.

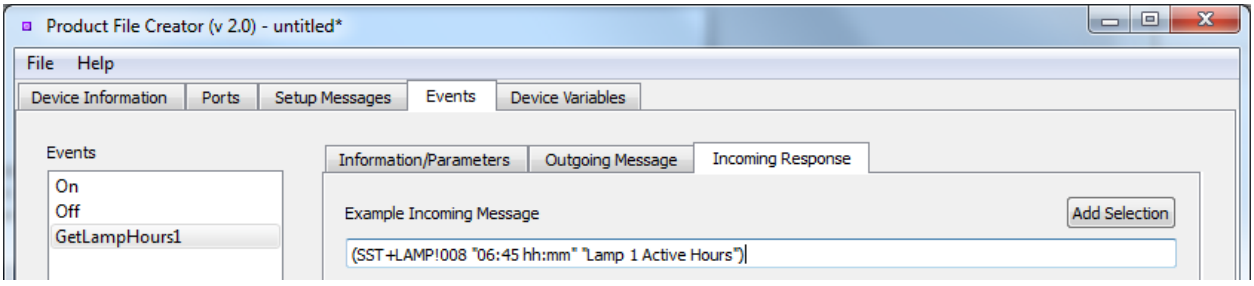
Create an event called “GetLampHours1” and check the box indicating that there is an Incoming Response:



The correct outgoing message is “(SST+LAMP? 08)”. Add this string in the Outgoing Command tab:

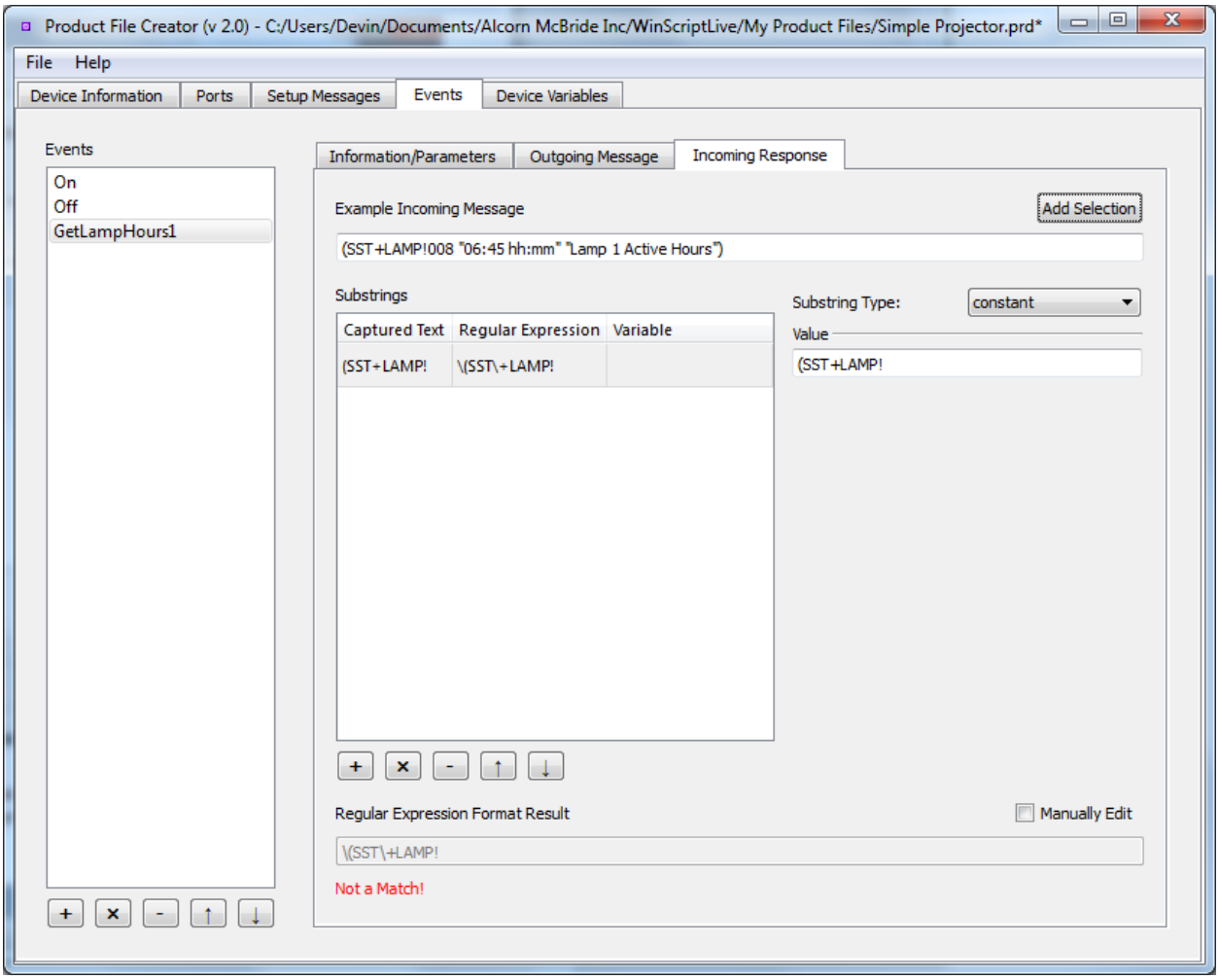


The Incoming Response message is trickier. We’ll use an Example Incoming Response to construct the final regular expression piece by piece. Type “(SST+LAMP!008 "06:45 hh:mm" "Lamp 1 Active Hours")” into the **Example Incoming Response** box:

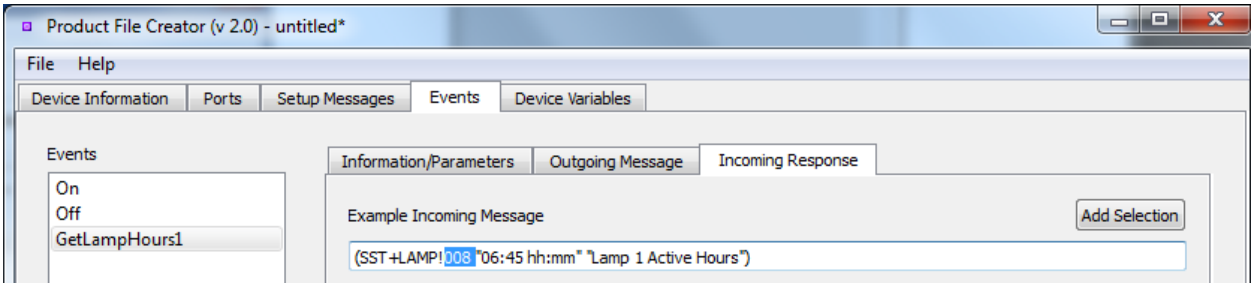


“(SST+LAMP!” is always at the beginning of a valid string so let’s highlight that portion of the Example Incoming Response and use it to add the first part of the final message. Click **Add Selection** and then choose “**Constant**”.

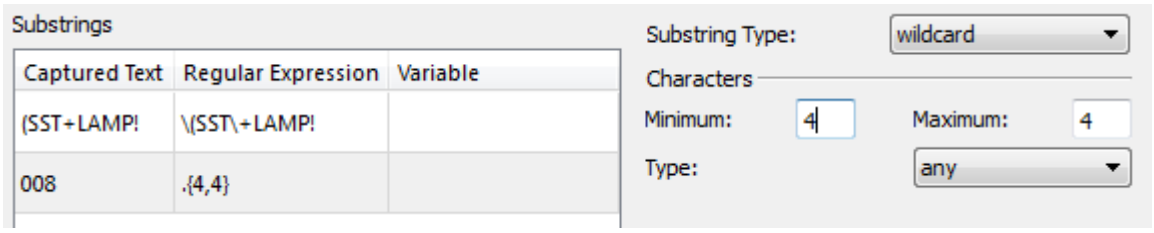
As you can see, “SST+LAMP!” is added as one of the substrings comprising the final message and appended into the **regexp_format**.



After that, the incoming message will have some data we can ignore until we get to the portion that contains the lamp hours. From our Example Incoming Response, this means everything after "LAMP!" and before "06:45 hh:mm". Let's highlight that portion:



Add it as a **wildcard** substring. Using a **wildcard** means that the exact substring does not need to be matched, just something about its form. The Product File Creator tries to decipher what type of text you want to match. Here, because we selected four characters (a space followed by "008"), it defaults to matching any string that is a minimum of 4 characters and a maximum of 4 characters long. That is, the regular expression assumes that the number of characters must be exactly 4 characters.

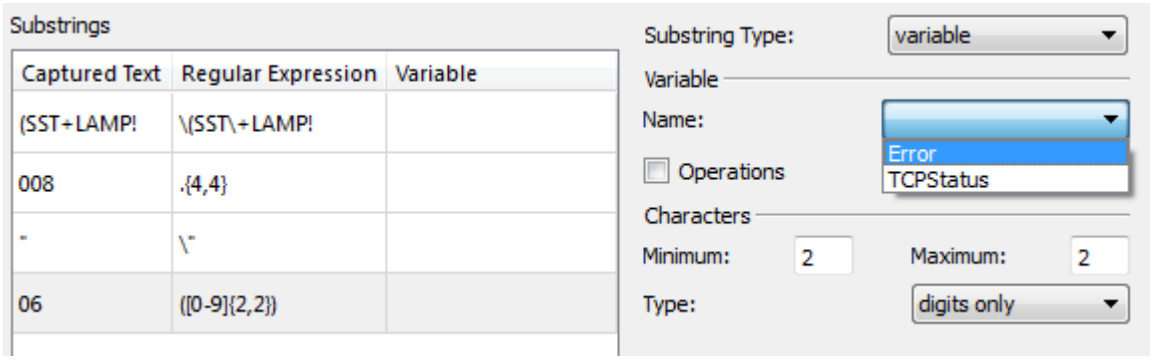


Since our goal is to ignore all characters until we reach the "06:45" string, let's erase the values in **Min** and **Max** and leave those boxes empty. Removing the Min and Max bounds means that everything that follows "SST+LAMP!" is now part of the wildcard substring we created. You can see in the **Response** Substrings section that this is more than just the 4 characters we highlighted. It is every character until the end of the message! That's not quite right, so let's cut the matching off some.

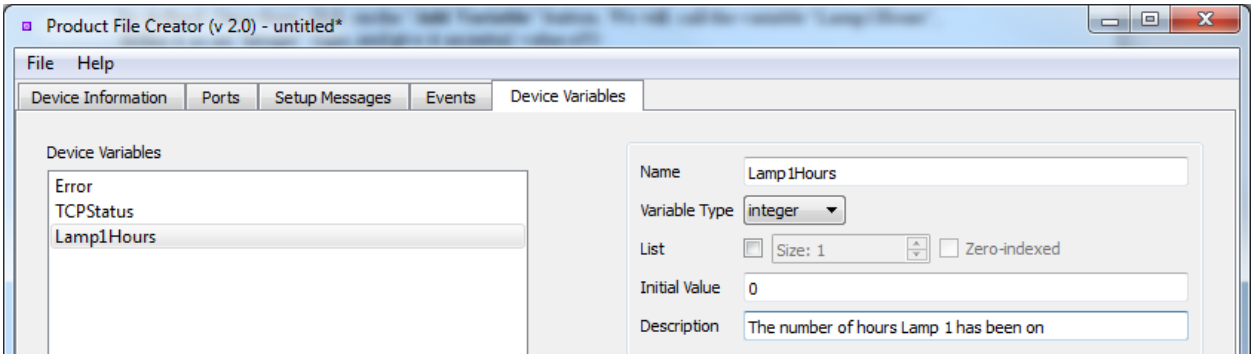
To do this, note that the next distinct portion of the Example Incoming Response is "**"06:45"**". The quotation mark before the 06 lets us know that the data we want (represented by 06 for hours and 45 for minutes) is next in the message. Let's add this quotation mark to the substring. Highlight it, choose **Add Selection**, and select **Constant** as the type.

Next highlight the "06" so that we can store it. Click "**Add Selection**". This time we will add it as a variable. By doing so, we are saying that we do not want to literally add the value "06" to the final regular expression, but rather we want to add a placeholder to store whatever would be in that location in a typical Incoming Response.

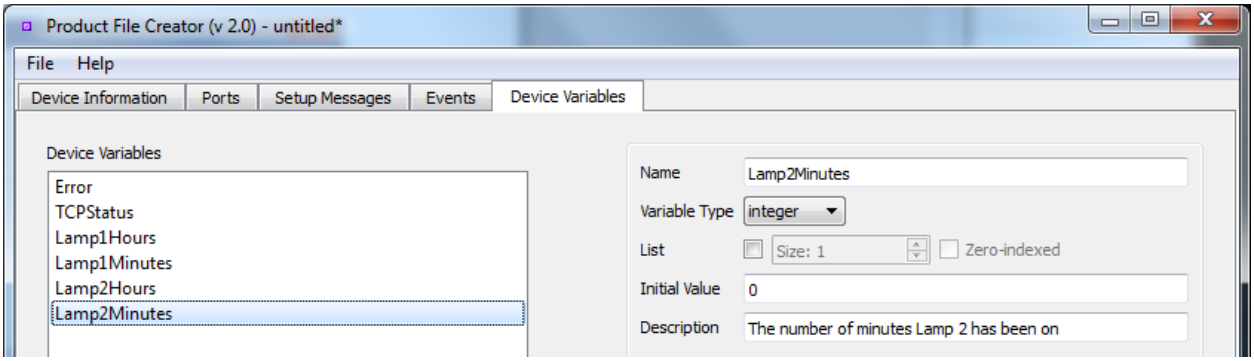
When you choose to add a substring as a variable, you must choose which Device Variable to store the substring in. If you click the dropdown menu next to "**Variable:**", you will see that we have not yet defined a place to store the lamp hours. All you should see in the list are "**Error**", which is a default variable, and "**TCPStatus**", which was automically created because we earlier specified that this device uses TCP/IP with its Ethernet port.



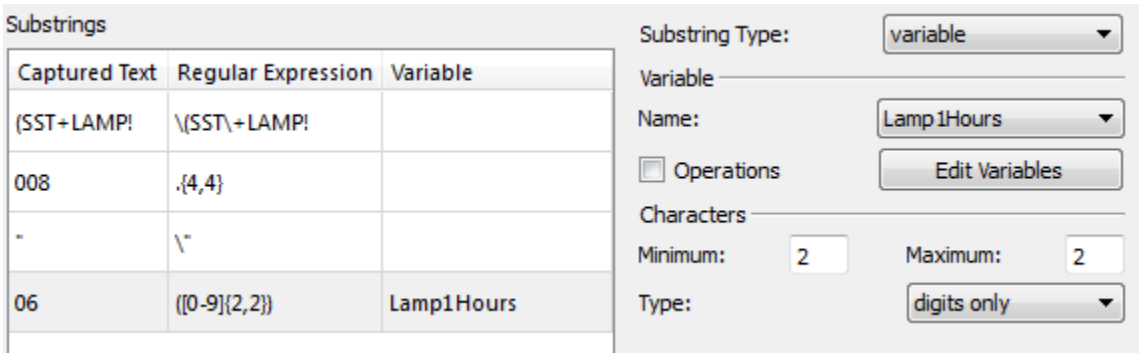
Click “Edit Variables” to be taken to the **Variables** screen. This screen is where Device Variables can be defined. Once there, click on the “**Add Variable**” button. We will call the variable “Lamp1Hours”, define it as an “integer” type, and give it an initial value of 0:



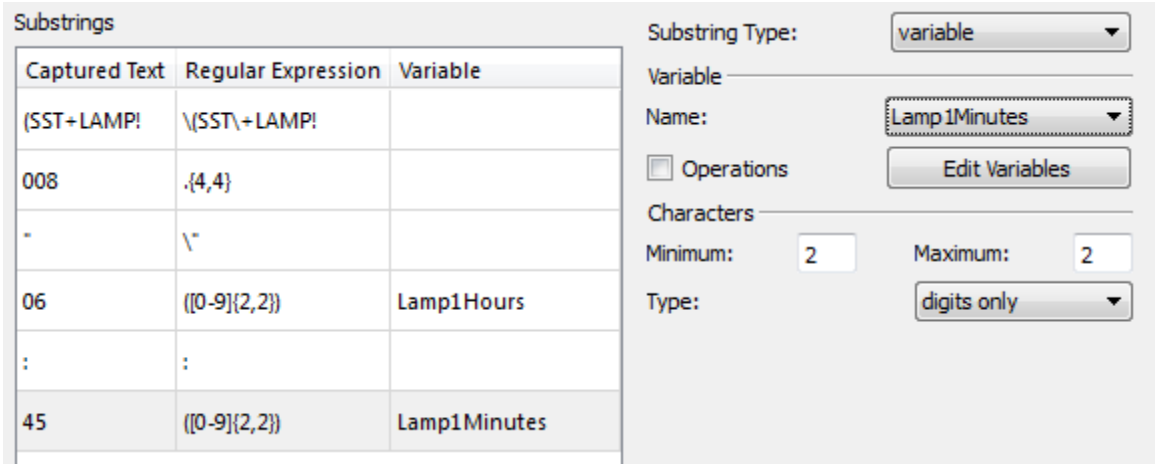
While you are here, go ahead and use the **Duplicate** button underneath the list of variables to define variables for the “minutes” portion of Lamp 1’s online time, as well as the corresponding variables for Lamp 2’s online time. Device variables can be defined at any time as needed.



We can now go back to the Events tab and use the variables we have just defined to capture part of the Incoming Response. Making sure "06" is selected from the "Response Substrings" section, choose "Lamp1Hours" from the Variable dropdown list. The regexp_format is updated accordingly. Notice that "**Character type**" now says "digits only" to let the regular expression know to expect 2 integer digits, not just any type of character.

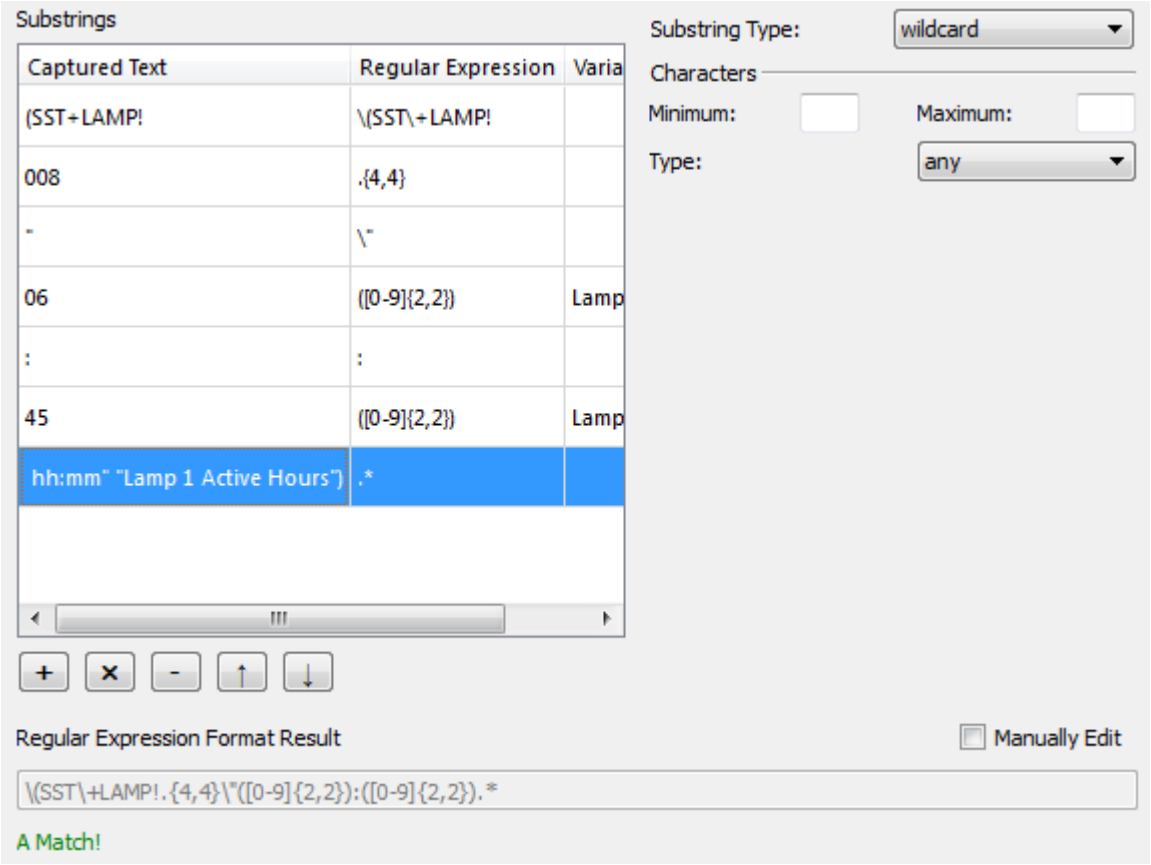


Next, add the "**:**" as a constant substring. Then, highlight the "**45**" from the Example Incoming Response and add it as a "variable" substring into the "**Lamp1Minutes**" variable:

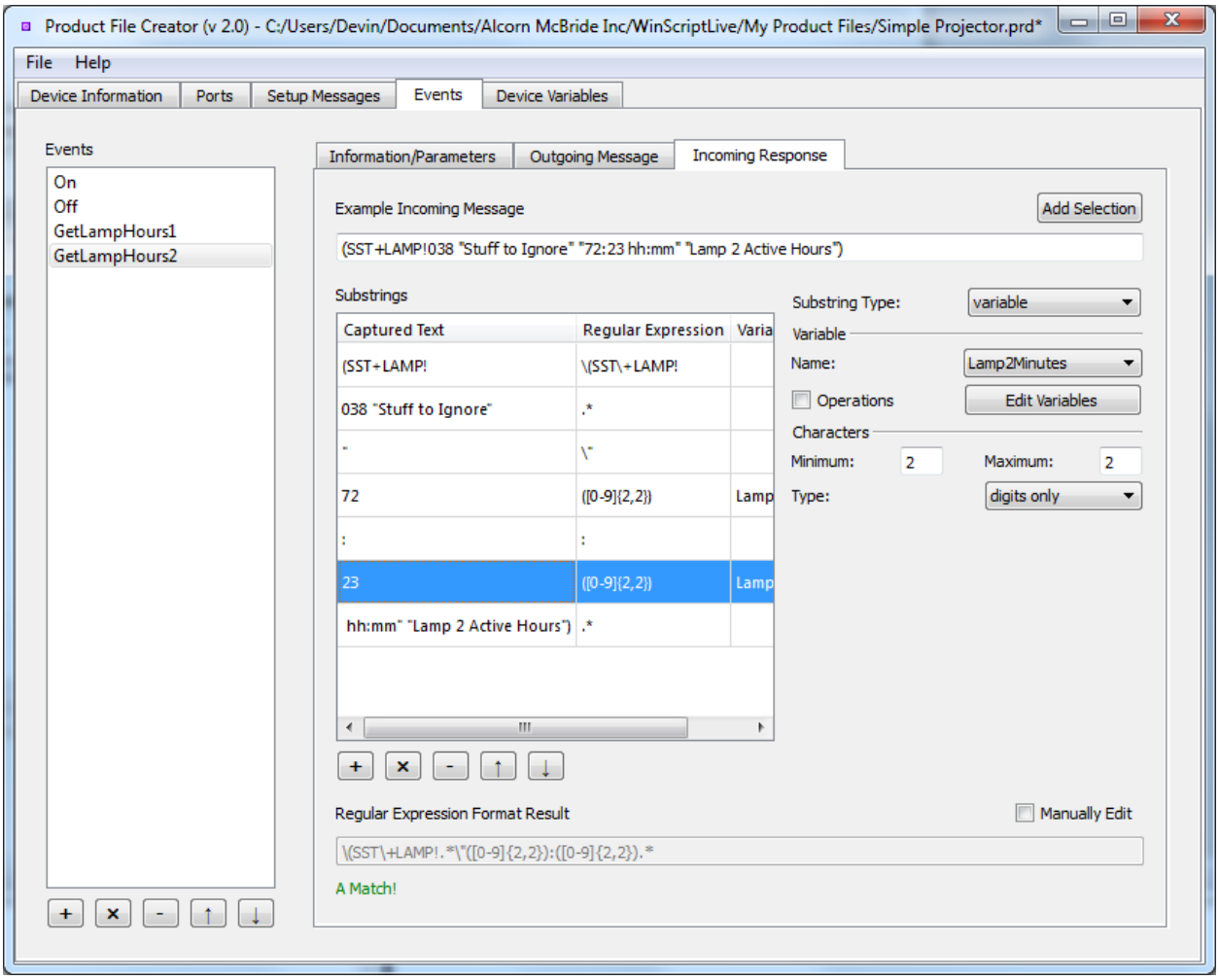


Once we add we have stored the Lamp 1 minutes, nothing else remaining in the Example Incoming Response matters to us. Highlight everything to the right of "**45**", and add it as a wildcard substring.

Finally, make sure to erase the **Min** and **Max** number of characters for this Wildcard substring and you should end up with a final regular expression of: "**\(SST\+LAMP!.*"([0-9]{2,2}):([0-9]{2,2}).***"

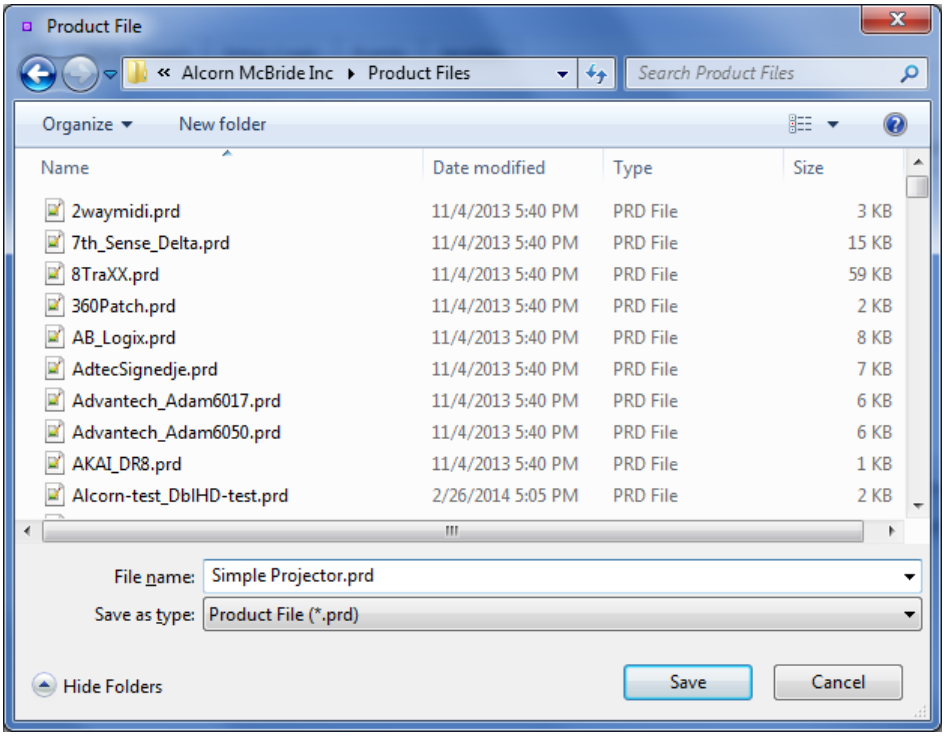


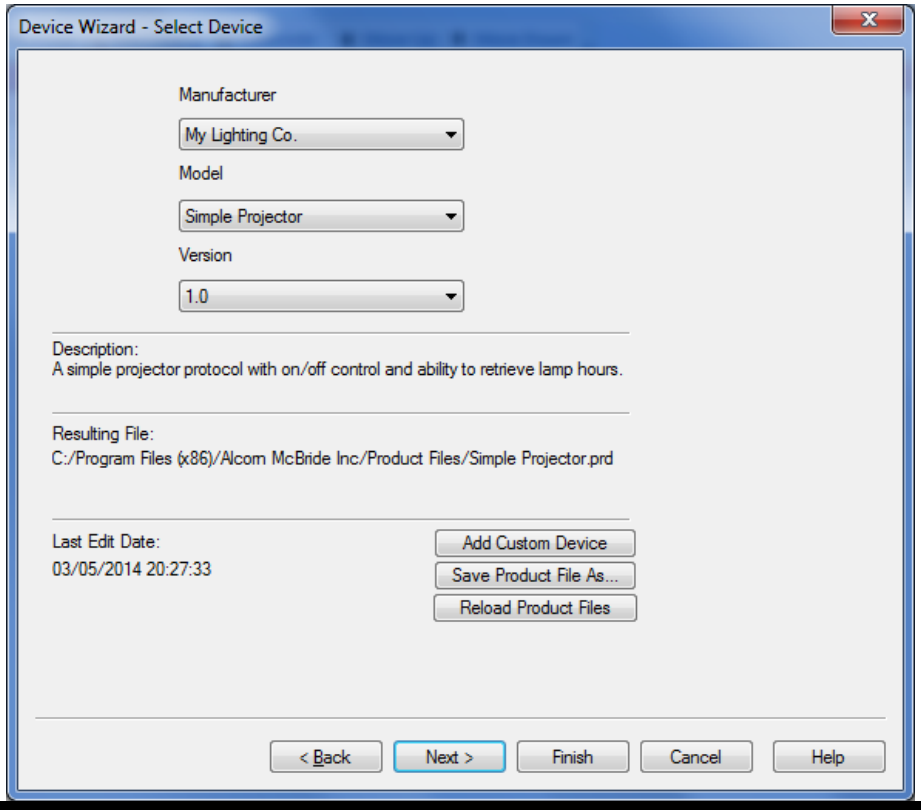
Now you can try it. Add an event for getting Lamp 2's online time. Use "**(SST+LAMP ?38)**" as the Outgoing Command and "**(SST+LAMP!038 "Stuff to Ignore" "72:23 hh:mm" "Lamp 2 Active Hours")**" as the example Incoming Response. Your screen should look similar to the following:



Using Custom Product Files in WinScriptLive

Once you save your product file, you can load it into WinScriptLive and use it like any other device. It is recommended that you save product files in the **"WinScript Live\My Product Files"** subdirectory of your **My Documents** folder. This will make them easily accessible from both WinScript Live and Product File Creator.




Device Wizard - Select Device
Manufacturer
My Lighting Co.
Model
Simple Projector
Version
1.0
Description:
A simple projector protocol with on/off control and ability to retrieve lamp hours.
Resulting File:
C:/Program Files (x86)/Alcorn McBride Inc/Product Files/Simple Projector.prd
Last Edit Date:
03/05/2014 20:27:33
Add Custom Device
Save Product File As...
Reload Product Files
< Back
Next >
Finish
Cancel
Help

Appendix A: Local Variables

When constructing outgoing and incoming messages for protocol commands, you may sometimes need to temporarily store the result of an operation for later use. For example, an incoming message may contain multiple bytes which need to be reordered or otherwise transformed before being stored in an actual device variable.

Local variables allow you to assign temporary names to the results of operations or bits of captured text from incoming messages so that they can be referenced by other substrings.

Consider the above scenario in which bytes in an incoming message need to be swapped before being stored to a device variable. Local variables allow this to be done without requiring a new device variable to be created for every byte in the message.

For example, imagine we have an incoming message consisting of four bytes “ABCD” which we want to store to a device variable in the form “DCBA”. This can be done using device variables like so:

Create a new event and enter “ABCD” as the example incoming message.

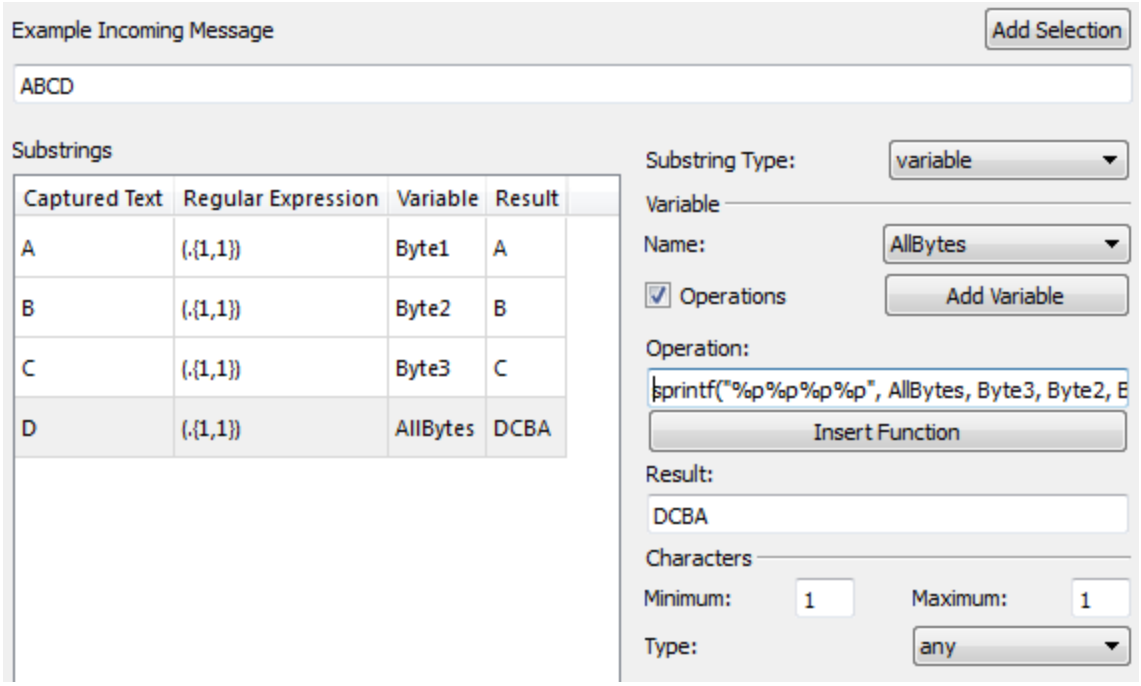
Select the first character, “A”, and click **Add Selection**. Select “local” as the type. Enter **“Byte1”** as the name of the local variable.

Repeat the previous step for the second two characters “B” and “C”, creating local variables named **“Byte2”** and **“Byte3”**.

Finally, select “D”, click Add Selection, and select “variable” as the type. Click **Add Variable** to create a new variable called **“AllBytes”** which will contain all four bytes in the order we want. Make sure **Operations** is checked, and then enter the following in the **Operation** text box:

sprintf("%p%p%p%p", AllBytes, Byte3, Byte2, Byte1)

This will store the final captured character in the AllBytes variable, and then append the remaining characters which were stored in the three local variables we created.



Local variables can be created the same way for outgoing messages as well. In addition, selecting “none” as the output format will store information in a local variable without including it in the actual message. This allows you to use substrings to store complex or commonly-used arithmetic expressions in local variables, and then reference them later as needed.

Appendix B: Regular Expressions and Non-Printable Characters

Non-Printable and Special Characters

Non-printable and other special characters can be represented in incoming and outgoing messages using escape sequences similar to those used by the C programming language:

- To insert a line break (0x0A), type a backslash followed by a lowercase n ("\n").
- To insert a carriage return (0x0D), type a backslash followed by a lowercase r ("\r").
- To insert a horizontal tab (0x09), type a backslash followed by a lowercase t ("\t").
- To insert any 8-bit character value, type a backslash followed by a lowercase x and two hexadecimal digits (case-insensitive). For example, the above three characters can also be entered as "\x0A", "\x0D", and "\x09" respectively.

Regular Expressions

When matching incoming messages, product files use Perl-compatible regular expressions to describe the format an incoming message will take. Regular expressions are a powerful tool for searching and capturing text, but the syntax can be daunting.

Product File Creator is designed to take the work out of creating regular expressions by allowing you to specify the "look" of a message piece-by-piece and generating the appropriate expression automatically. However, there may be instances in which you may prefer to specify a regular expression manually. The following resources are provided as a helpful and comprehensive guide to regular expression syntax:

Regex Cheat Sheet at RexEgg.com – A comprehensive "cheat sheet" for regular expression syntax, including variations used by some common programming languages. The most important parts of this document for creating product files are the sections "Characters" through "Character Classes"; these sections cover all of the syntax which Product File Creator uses when automatically generating expressions.

Regex Tutorials at RexEgg.com – The rest of this site provides a much more in-depth look at regular expressions, including more detailed information about the sections provided by the above cheat sheet.