LEA AMPLIFIER MODULES 3.0

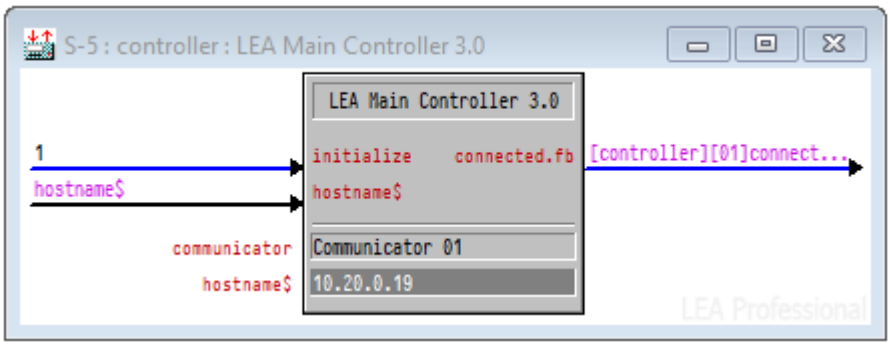
SUMMARY

The LEA Amplifier modules provide a robust and reliable way of controlling the LEA Dante and Network Connect series product line using their latest Open API TCP Protocol. Please email techsupport@leaprofessional.com for assistance.

The module suite abstracts the set of commands made available through the API to seamlessly communicate with API endpoints exposed for your amplifier, including SET/GET/SUBSCRIBE/UNSUBSCRIBE.

The Crestron modules provided are separated out into a combination of diagnostic, controller, and amplifier element modules that you can manipulate at runtime.

Communicating with different amplifier-related controls and sensors is done by an arrangement of main controller(s) to value modules, where the value modules are paired with a main controller by selecting the appropriate communicator to match a parent main controller.



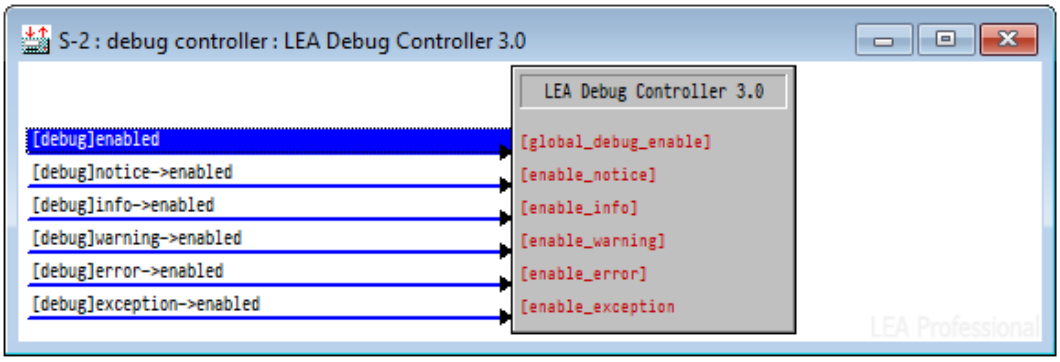
*Note: The **hostname$** serial input (if MSP'd and not dropped as a transient value) will take precedence over the **hostname$** parameter defined at the time the **initialize** signal transitions high on the digital input. (Tip: Try to initialize this signal before all other value modules within a reasonable amount of time to ensure any threads waiting for the communicator can initialize for the value modules to send subscription commands to be executed successfully.)*

The main controller provides up to 16 communicators, which are essentially TCP clients that can connect to the same or different LEA amplifiers for your system needs. A communicator is initialized by the presence of a main controller module defined in your program with the selected communicator number and hostname, defining the address of the device to connect to for the given communicator. Please only use (1) instance of the main controller per communicator number in your program. If for any reason you

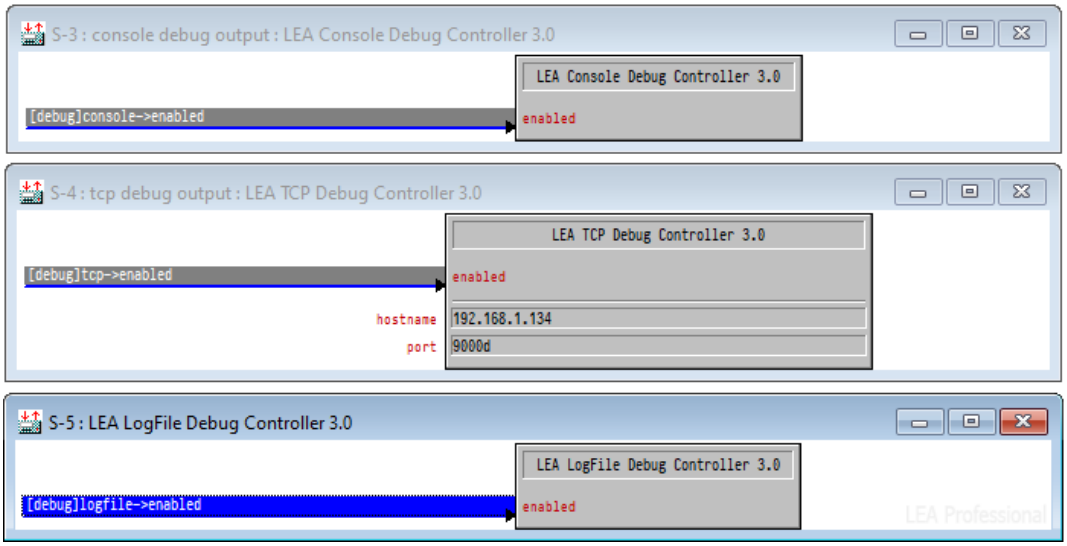
need more than 16 communicators in a single program, please reach out to LEA professional for assistance.

DIAGNOSTICS

The diagnostic/debug modules can be arranged in your program with any combination of (1) general debug controller to enable/disable debugging globally, or to set any of the available debug levels that will automatically be filtered through any of the corresponding console, TCP, and/or logfile debug controllers. Please note, that you should only have (1) instance of each type of debug module in your program.



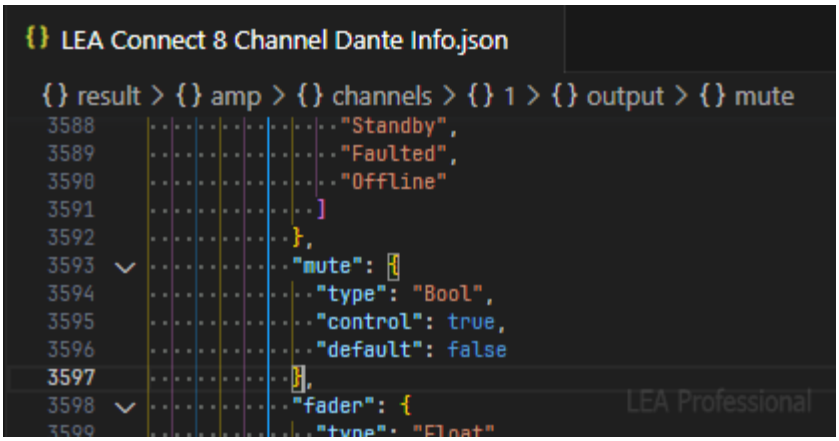
The (3) debug output controller modules are shown below:



Note: Logs are saved to \User\Logs\ with a timestamped value by default. There may be future versions of the modules that will allow you to specify a directory if you need to save to alternative storage, however that is currently not included in version 3.0.

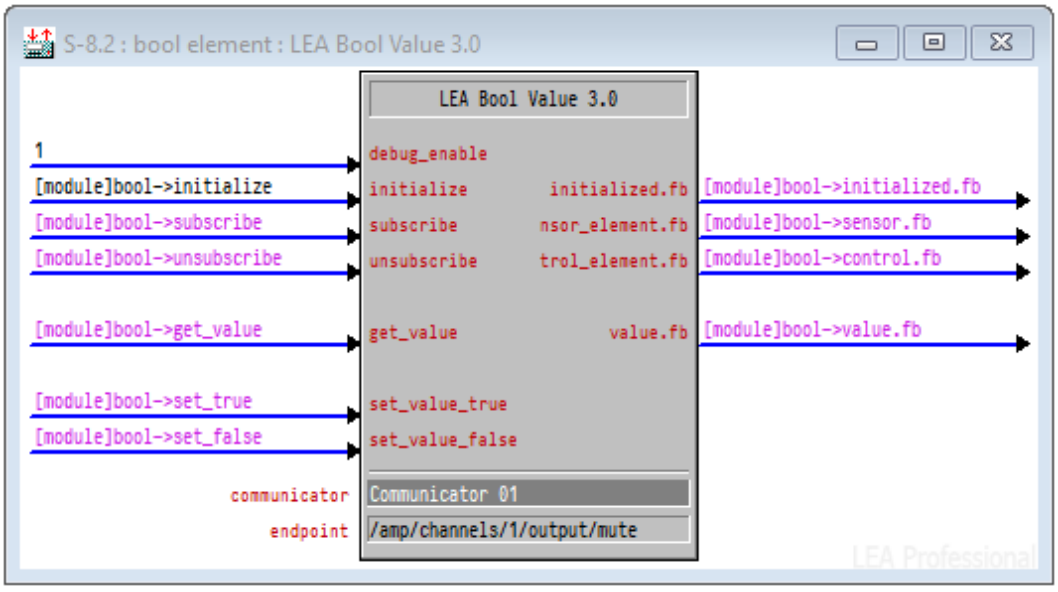
OTHER MODULES

There are (3) generic modules for Boolean, Float, and String value endpoints made available for you to control specific endpoints throughout the system. Please use the API JSON definition provided with the modules to find the appropriate case-sensitive JSON path for the corresponding **endpoint$** module parameter. This will tell you what module you'll need to use for a given endpoint. As an example, to control Channel #1 Output Mute, you would need the Bool Value module.



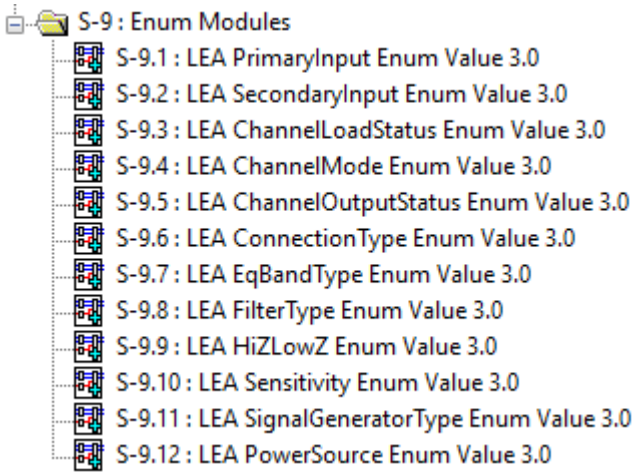
You can see for the "mute" definition that its type is "Bool" above, and that it's a control element. The "breadcrumbs" at the top of the Visual Studio Code editor shows the JSON path too; you need to include after (and including) "amp" in the path. **If you do not type this in with the proper casing of the path, it will not work since JSON paths are case-sensitive!** If using these modules on many projects, creating SIMPL Windows module wrappers around these generic modules might be a good solution. We did not provide that in the module suite to keep the modules extensible, in case of any future API additions or changes.

The proper module has been added to the program as follows:



*Note: You shouldn't have to trigger the **get_value** digital input or the **subscribe** or **unsubscribe** digital inputs, as these are provided for convenience only. The modules will automatically detect whenever a connection to the amplifier has been made (even after a lost connection) and will automatically subscribe and retrieve the current value. The module only has to be initialized once after the initialization for the relevant main controller with the corresponding communicator number has been initialized. The initialization of the modules only pairs the module with the corresponding communicator and does not have anything to do with the TCP connection of the communicator. (Tip: The modules should work if you wanted to quickly put a **1** on any initialize signals, since the SIMPL# library handles automatically synchronizing the proper initialization order.)*

The collection of Enum value modules are easier to use since they do not require the programmer to define any JSON endpoint.



For example: