Debugging Tcl in Visual Studio Code

Jonathan Cone
Software Developer
FlightAware
Project Source: https://github.com/conej730/vscode-tcl-debug

For developers without access to ActiveState's Tcl development tools there are few debugging tools available. It is possible to install a Tcl debugger based on ActiveState's ProDebugger in Eclipse, but Eclipse is not a very popular IDE for Tcl development. Visual Studio Code (VSC) is an open source text editor / IDE released in 2015 that is available on Windows, Macs and Linux. It support snippets, syntax highlighting, code folding and many other features for a wide variety of languages through the installation of extensions. At FlightAware, some of the developers are using VSC for their Tcl development as extensions already exist to add syntax highlighting and snippets for the Tcl language. Currently lacking, though, is a Tcl debugging extension.

Visual Studio Code supports a language agnostic Debug Adapter Protocol allowing for any language to be debugged once an appropriate debug adapter is created. To add debug support to VSC for a new language, a debug adapter and debug extension are required (See Figure 1). The debug adapter acts as a bridge between VSC and the language's native debugger. It is usually a stand-alone program and communicates with the VSC UI across stdout/stdin using JSON messages based on the Debug Adapter Protocol (https://microsoft.github.io/debug-adapter-protocol/). Technically the debug adapter will be wrapped as a Debugger Extension, but this is done behind the scenes by VSC. When a debug session is initiated, VSC will launch the debug adapter and send it a "launch" or "attach" request. When launch is requested the adapter will need to launch the program to debug, whereas for an attach request the debugger will attach to an existing process. Subsequent requests to the debug adapter will set desired breakpoints, request a stacktrace or variables, respond to stopped events and manage the debug workflow. Figure 2 shows an example of the calls between VSC, the debug adapter and the native debugger (gdb in this figure). In order for users to access the debug adapter's functionality, VSC provides a debug extension API in the vscode.debug namespace.

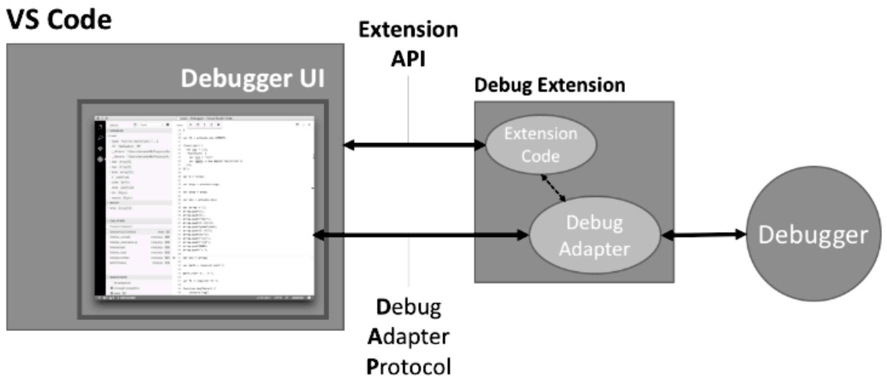

Figure 1 - https://code.visualstudio.com/docs/extensionAPI/api-debugging

The extension API provides some hooks for setting configuration, such as creating an initial launch.json for new projects. There are debug session life-cycle controls available such as starting a debug session, being notified when a debug session has started, a debug session API for returning any debug adapter protocol requests back to the program being debugged and a breakpoints API for getting all user specified breakpoints and receiving notifications when they change. The extension itself will also define what options the user will see when attempting to debug a program and the available configuration options. For example, one might create a debug configuration to launch the current file or another one to define an entry point for the overall program. The specific features that the extension will provide and how it will launch the debug adapter are all included in the package.json file. The package.json file acts as a manifest for any VSC extension listing dependencies, name, publisher, author, etc. The capabilities of the debug extension are also refined here and any specific runtimes required.

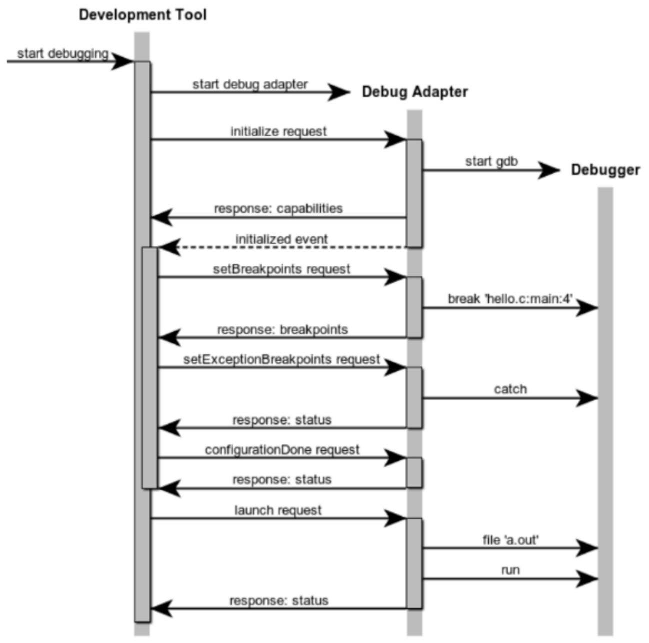

Figure 2 - https://microsoft.github.io/debug-adapter-protocol/overview

Tcl Debug Extension

The Tcl debug extension currently consists of a simple interface for debugging a Tcl application. There is a single configuration supported, “Tcl: Current File”. This configuration is defined in the extension’s package.json under the debuggers initialConfigurations. The “Current File” configuration is a launch type config which passes the path of the currently open file to the debug adapter. The “Current File” configuration is available as an initial configuration for the user when debugging a Tcl file. In the extension.ts file, the activate function (called when the user activates the debugger) creates a new TclConfigurationProvider and pushes that provider onto the extension context subscriptions. Subscriptions for an extension context are just a list of disposables that will be disposed of when the extension is deactivated. By adding this provider, when a user selects to debug a Tcl application, the TclConfigurationProvider's resolveDebugConfiguration method will be called before launching the debug adapter. This method (also defined in extension.ts) will fill in any missing parameters from the launch.json file with some reasonable defaults. The supported configuration attributes are outlined in package.json and include an option to set a specific tclsh path, command line arguments to pass to the program being debugged, and environment variables that can be defined. These options allow for some additional flexibility when starting the debugger if required.

Most of the work for debugging the Tcl application is done by the debug adapter. The debug adapter is implemented in Tcl and is launched by VSC when the user starts a debug session on a Tcl file. The adapter begins by listening on stdin for new messages and emitting responses on stdout. All communication between the debug adapter and VSC follows a JSON schema (https://github.com/Microsoft/vscode-debugadapter-node/blob/master/debugProtocol.json) and this debug adapter uses yajltcl for decoding and serializing JSON messages. Because of this dependency, yajltcl must be installed on the user's host. As JSON messages are received, they are decoded and the type of the message and command are examined. Most of the received messages are of the request type. Different procs exists for handling each of the different request commands. The first request received is the initialize request. This request will specify the options for this debug session and the adapter must respond with its capabilities. Once configuration is done, the adapter will then send an initialized event to VSC. At this point the adapter may receive a request to set breakpoints. In order to set breakpoints the adapter needs an appropriate Tcl debugger. For this extension the debugger has been bundled with the adapter and consists of core portions of the tcldebugger found in TclProDebug (https://github.com/flightaware/TclProDebug). This is the debugger previously released by TclPro and updated to work with Tcl 8.6. The portions of the debugger code related to interaction with the Tk interface in TclProDebug have been removed and instead the debug adapter sources the relevant files (including the tclparser library) and makes direct calls into the debugger. For example, to set breakpoints the debug adapter will ensure that the debugger is initialized and then call `dbg::addLineBreakpoint` after sanitizing the location. Similarly once a breakpoint is hit the `dbg::getStack` and `dbg::getVar` procs are called by the adapter to get information on the requested state and then translate that Tcl response into appropriate JSON message conforming with the VSC schema. Most of the calls to manipulate the debugger are made to the procs in dbg.tcl in the debugger folder.

While the current state of the Tcl debug extension and adapter allows for some simple debugging of a Tcl application, there are some notable limitations. The only implemented configuration is a launch configuration so remote debugging is not available. TclProDebug supports remote debugging so it seems reasonable that the current extension can be expanded to support that. If this could be implemented in a way that was simple for users and required no or minimal code changes in the user's program that would be ideal. Also allowing for the debugger to attach to a running process (local or remote) and provide debug information would likely be useful. This may not be achievable with the current debugger, but might be achievable in combination with tclgdb. Additionally, the debugger is considerably slower than native Tcl.

The debugger code instruments all the procs and parses all the code to keep track of locations for breakpoints, stack information and variable state. This overhead has a significant impact on performance. For small applications this is not an issue, but for a larger and more complex application this could be an obstacle. Any improvements to the underlying debugger code, or a more efficient replacement could provide great benefit. For companies like FlightAware the ability to remotely debug a rivet page would be especially beneficial and is not currently supported. Finally yajltcl has not been tested on Windows, so at this point the debugger does not work on that platform.

This new Tcl debug extension for VS Code is the first step toward providing better support for Tcl in the increasingly popular, cross-platform and lightweight IDE. For new developers who are used to having debug capabilities in their IDE, the addition of this feature will hopefully remove one barrier to their adoption of Tcl. The debugger is still in its infancy and we will continue to add additional features and improve its performance. Community contributions are welcome and the extension will be published in the VSC marketplace by the end of the year.