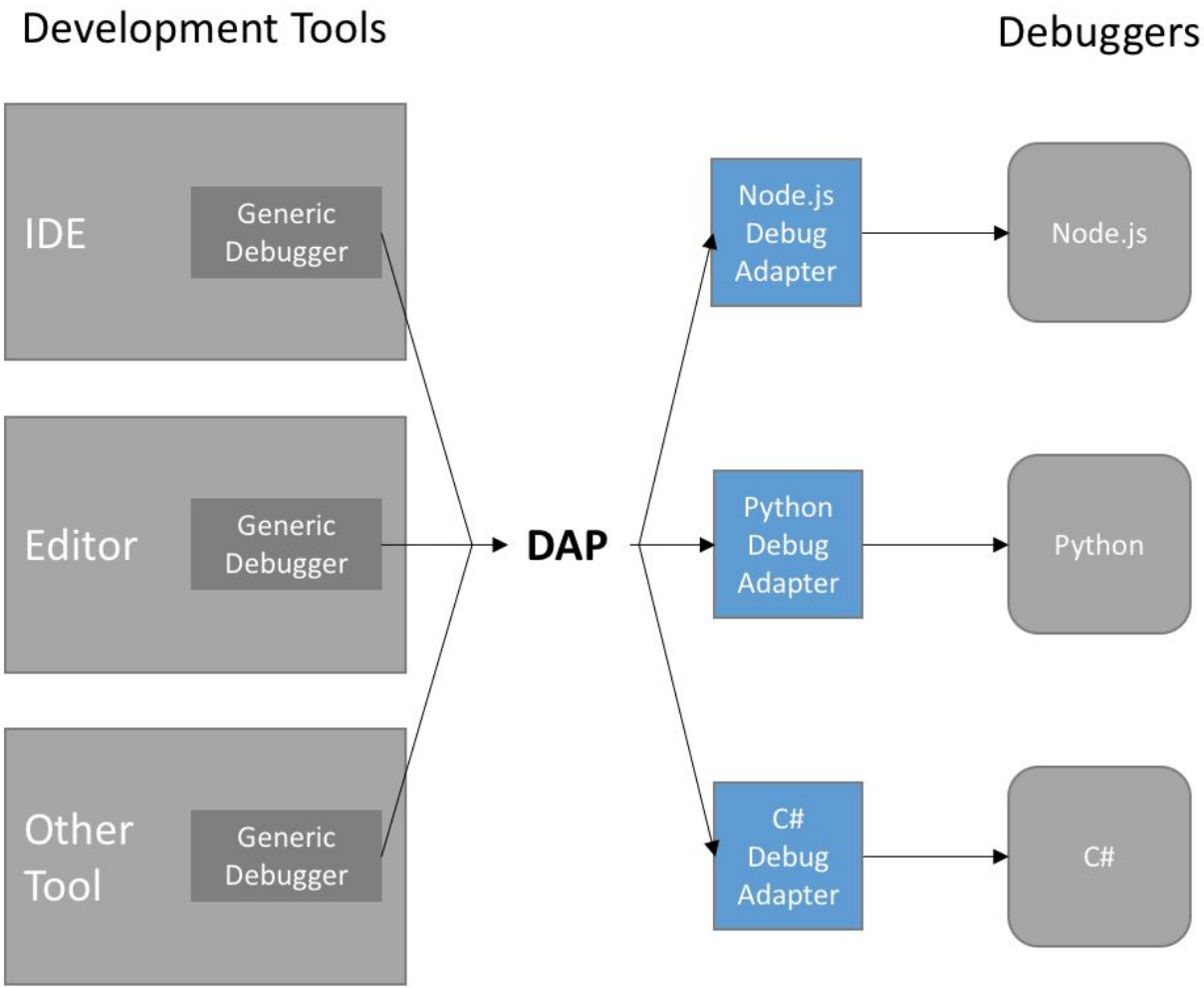
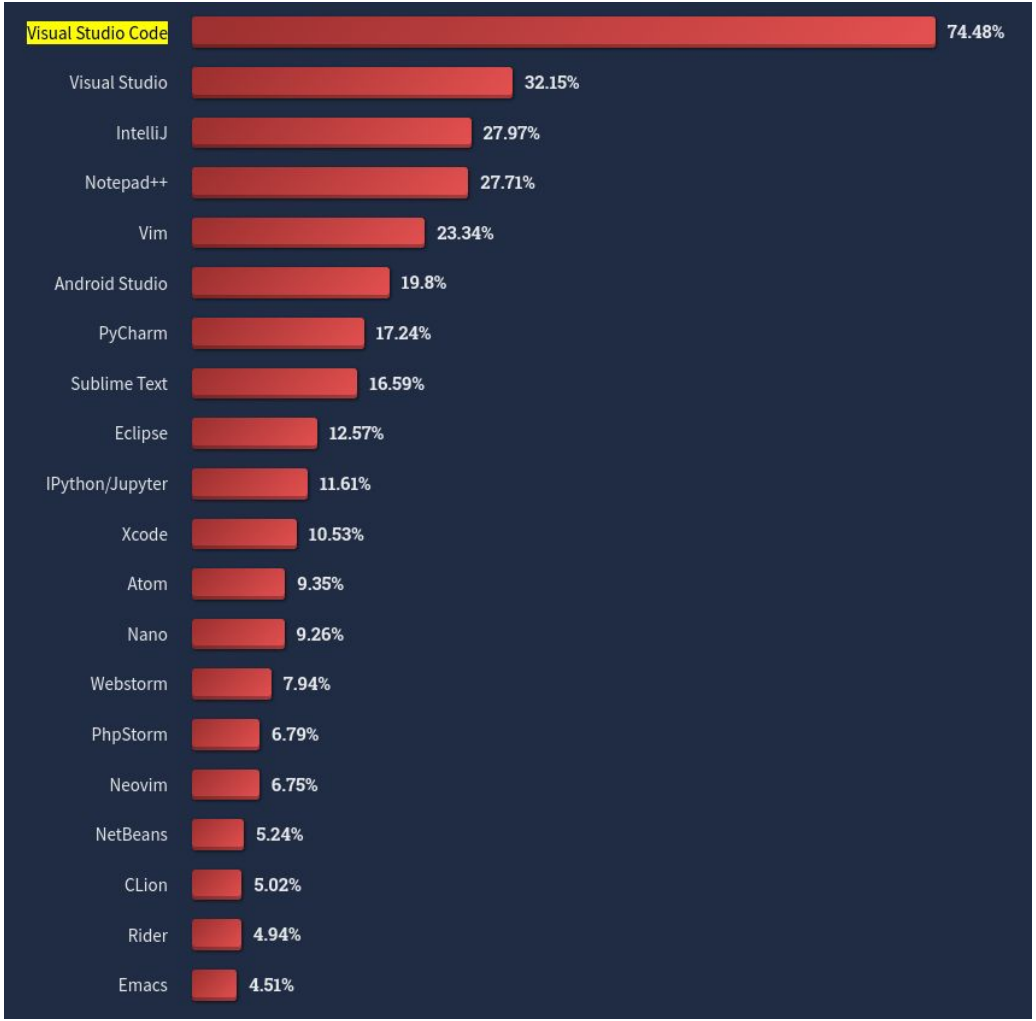
Debug Adapter Protocol

Nick Fitzgerald
Bytecode Alliance SIG-Debugging
2023-03-23

“The idea behind the Debug Adapter Protocol is to standardize an abstract protocol for how a development tool communicates with concrete debuggers.”


Development Tools
Debuggers
IDE
Generic Debugger
Editor
Generic Debugger
Other Tool
Generic Debugger
DAP
Node.js Debug Adapter
Python Debug Adapter
C# Debug Adapter
Node.js
Python
C#


Visual Studio Code 74.48%
Visual Studio 32.15%
IntelliJ 27.97%
Notepad++ 27.71%
Vim 23.34%
Android Studio 19.8%
PyCharm 17.24%
Sublime Text 16.59%
Eclipse 12.57%
IPython/Jupyter 11.61%
Xcode 10.53%
Atom 9.35%
Nano 9.26%
Webstorm 7.94%
PhpStorm 6.79%
Neovim 6.75%
NetBeans 5.24%
CLion 5.02%
Rider 4.94%
Emacs 4.51%

Supported Debugging Features (an incomplete list)

- Breakpoints
- Watchpoints
- Disassembling
- Source-level objects
- Reading/writing directly to memory
- Threads
- Reverse debugging

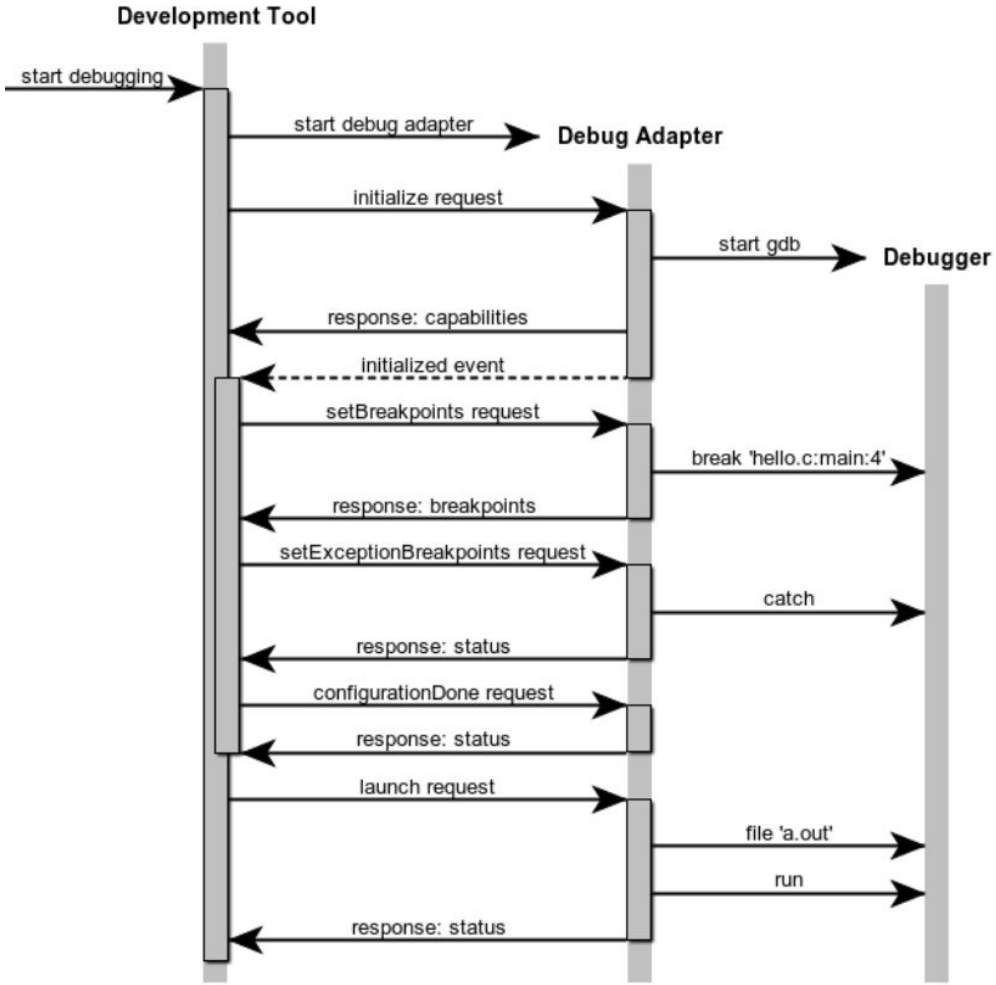
Base Protocol

```
Content-Length: <N> \r\n
<JSON packet of length N>
```

JSON Protocol

```
interface ProtocolMessage {
  /**
   * Sequence number of the message (also known as message ID). The `seq` for
   * the first message sent by a client or debug adapter is 1, and for each
   * subsequent message is 1 greater than the previous message sent by that
   * actor. `seq` can be used to order requests, responses, and events, and to
   * associate requests with their corresponding responses. For protocol
   * messages of type `request` the sequence number can be used to cancel the
   * request.
   */
  seq: number;

  /**
   * Message type.
   * Values: 'request', 'response', 'event', etc.
   */
  type: 'request' | 'response' | 'event' | string;
}
```


Development Tool
start debugging
start debug adapter
Debug Adapter
initialize request
start gdb
Debugger
response: capabilities
initialized event
setBreakpoints request
break 'hello.c:main:4'
response: breakpoints
setExceptionBreakpoints request
catch
response: status
configurationDone request
response: status
launch request
file 'a.out'
run
response: status

Source-Level Protocol

- The protocol speaks in terms of
  - Stacks
  - Scopes
  - Variables
  - Values
  - Etc…
- Not in terms of raw registers and memory

Source-Level Protocol

- This is in contrast to gdb's machine-level protocol, IIUC
- This should work out fine for Wasm-level debugging, since Wasm is a relatively high-level language
  - Conversely, a machine-level protocol could be a problem if the protocol bakes in (for example) the assumption that the stack can be recovered from just memory and registers, since that is not the case in Wasm where it is actually built into the architecture

The request returns the variable scopes for a given stack frame ID.

```
interface ScopesRequest extends Request {
  command: 'scopes';

  arguments: ScopesArguments;
}
```

Arguments for `scopes` request.

```
interface ScopesArguments {
  /**
   * Retrieve the scopes for the stack frame identified by `frameId`. The
   * `frameId` must have been obtained in the current suspended state. See
   * 'Lifetime of Object References' in the Overview section for details.
   */
  frameId: number;
}
```

Response to `scopes` request.

```
interface ScopesResponse extends Response {
  body: {
    /**
     * The scopes of the stack frame. If the array has length zero, there are n
     * scopes available.
     */
    scopes: Scope[];
  };
}
```

Inspecting Compound Values

- Client sends, for example, an `evaluate` request
- Adapter responds with a result and **“value reference number”**
  - Value reference number is a handle that is valid until the debuggee resumes execution
- Client sends a `variables` request
  - The request references the value reference number
- Adapter responds with a list of fields on the value
  - Each field’s value will also have a value reference number

Displaying Values

- In addition to a value reference number, values also have a **“variable presentation hint”**
- Clients can use these hints when displaying values in their UI

```
kind?: 'property' | 'method' | 'class' | 'data' | 'event' | 'baseClass'
    | 'innerClass' | 'interface' | 'mostDerivedClass' | 'virtual'
    | 'dataBreakpoint' | string;
```

```
attributes?: ('static' | 'constant' | 'readOnly' | 'rawString' | 'hasObjectId'
    | 'canHaveObjectId' | 'hasSideEffects' | 'hasDataBreakpoint' | string)[];
```

```
visibility?: 'public' | 'private' | 'protected' | 'internal' | 'final' | string;
```

```
lazy?: boolean;
```

Learn More

- Overview
- Specification
- List of already-implemented adapters
- Emacs `dap-mode`