Proposal for Additional Data Fields for Breakpoints in VSCode and DAP

Version 0.2-Draft: 20/10/2023 Asim Gunes

Debug Adapter Protocol (DAP) defines rigid payload schema for breakpoints where there is no suggestion for transferring custom information. Especially in low-level applications, additional information like breakpoint types (hardware/software breakpoints), multicore features and thread features are often considered as general information requirements for a debugging operation. In this proposal, we like to define a way to extend the breakpoint payload in DAP. Firstly, we like to propose a method to exchange custom data. Secondly, we like to propose some well-defined fields for common scenarios. We believe well-defined new payload fields are needed for standardization of common scenarios where additional custom data will cover extending the environment for rare/edge cases. –
Keywords: DAP, Breakpoints

1. Introduction

DAP[1] defines an abstract protocol for development tools to communicate with the debuggers. They called the intermediary component as Debug Adapters which adapts an existing debugger or runtime to the protocol.

In terms of breakpoint usage scenario in DAP Specification[2], we observe TWO important integration point in the use case scenario. First, the **Initialize** request where debug adapter defines its capabilities, second the set breakpoint requests where DAP contains FIVE requests which are **SetBreakpoints**, **SetDataBreakpoints**, **SetExceptionBreakpoints**, **SetFunctionBreakpoints**, **SetInstructionBreakpoints**.

The **Initialize** request, invoked by development tool at the beginning of debug session and sends back the supported features of the debug adapter which is called **Capabilities**. These supported features list contains information like supportsTerminateRequest or supportsSetVariable etc…. It is expected from Developer Tools to act accordingly and do not use any capability if not defined by the debug adapter in Initialize response (For instance if Debug Adapter doesn't support set variable capability, Development Tool should not send this request).

Setting breakpoint requests (**SetBreakpoints**, **SetDataBreakpoints**, **SetExceptionBreakpoints**, **SetFunctionBreakpoints**, **SetInstructionBreakpoints**), are invoked at the start of the debug session, as well as, during the debug session if a new breakpoint is inserted (or an existing breakpoint modified/removed).

[1] https://microsoft.github.io/debug-adapter-protocol/
[2] https://microsoft.github.io/debug-adapter-protocol/specification

2. Previous Works

Extending breakpoint information in DAP comes as a suggestion to the community time to time. It seems that there is a common need not only in low-level applications but also in higher level programming languages. As far as we observer, there are limited tickets created related to this topic with this with a limited number of supporters and closed after a while. We believe that a general approach, with supporting the previous issues and could create a benefit for all.

Previously, G. Perera[3] asked for extra metadata for defining a breakpoint level dependency and implementing a special kind of breakpoint named “trigger breakpoints” for utility like functions which are getting high number of hits but asked to stop in condition if a previous breakpoint hit. Which seems a still a semi-active ticket with including discussion for extending payload in DAP.

In another request[4], it is asked from a user in GitHub that if there is a way to define session specific breakpoints (I presume it is asked for selection of a specific core or thread). It is mentioned in the thread that it is not applicable because of the limitation of the recent DAP.

M. Blout[5], suggesting extra metadata support for breakpoints to provide more flexibility which is recently in community review. Which seems it is the most parallel discussion with this document.

On the other hand, it is valuable to mention that DAP and VSCode is also supporting limited additional data including condition expression and hit count while the defining of the breakpoint[6].

The main idea of this paper comes from the concept discussion in an e-mail thread “CDT Cloud: Multi-core Debugging” between S. Dirix, M. Goodchild, J. Graham, A. Gunes, J. Helming, K. Kirenko, P. Langer, R. Moran, J. Reinecke, W. Riley (*in surname order*). We try to summarize the general concept and here in this document.

3. Suggestions

This section contains change suggestion for DAP. Following suggestions are consolidated information from previous works and discussions.

3.1. Additional Custom Data Fields for DAP

Basically, we like to propose adding a new field named “**data**” into the payload of the set breakpoint requests. The “**data**” field will be type of object without a defined schema, and support transferring any information. We will call the fields defined in data as **Custom Field**.

Additionally, we like to define the following requirements for this new implementation:

1. Debug Adapter could send the information of the accepted Custom Fields at the response of the **Initialize** request.
2. Development Tool should not send a Custom Field if not defined as accepted.

---

[3] G. Perera, “Add support for breakpoint dependencies (trigger points)”, 17/07/2022, https://github.com/microsoft/vscode/issues/152428

[4] A general user, “Vscode Sets breakpoint in both sessions in multicore debugging with single Debug Configuration”, 08/05/2023, https://github.com/microsoft/vscode/issues/181794

[5] M. Blout, “Support extra metadata for Breakpoints”, 13/07/2023, https://github.com/microsoft/vscode/issues/187854

[6] https://code.visualstudio.com/Docs/editor/debugging#_advanced-breakpoint-topics

3. Information exchange between Debug Adapter to Development Tool should contain enough information for Development Tool to populate User Interface for additional data. (Another alternative, this could be defined in the VSCode extension configuration not in DAP)
4. Custom Fields could be in types of string, boolean or enum.
5. Enum type Custom Fields data could be populated dynamically in debug session time. This could be a requirement for core selection or thread selection.

propose a control mechanism for additional data.

Initialization:
We are planning to add **supportedCustomFields** to the Capabilities which returns from the Initialize request of DAP. supportedCustomFields will be an array of ISupportedCustomField.

```
ISupportedCustomField {
    name: string; // Name of the field
    type: "string" | "boolean" | "enum"; // Type of the field
    values?: string[]; // Values of the enum
}
```

The following code represents a sample response of supportedCustomFields:

```
supportedCustomFields = [{
    name: "aStringField",
    type: "string"
},{
    name: "aBooleanField",
    type: "boolean"
}, {
    name: "enumField",
    type: "enum",
    values: ["Item1", "Item2", "Item3"]
}];
```

Open Issues:

1. Should we need to cover integer data type, currently planning to ignore numeric values and use string instead, still keep boolean for UI/UX purposes just to support simple checkboxes for UI/UX interaction.
2. Should we need to cover complex/object data type, then how the UI part of the complex data type will be populated.
3. **Initialize** is invoked at the start of the debug session, how developer tool will get the custom fields information. (Please check CodeTime/DebugTime Dilemma section below)

3.2. Additional Well-defined Fields

Even the following fields could be implemented with the Custom Data Fields scenario, we like to propose to add these fields into the standard payload of the set breakpoint operations.

breakpointType:

This field will be an optional field with valid values of "hardware" or "software". Debug adapter will insert breakpoint according to "breakpointType" value if it is defined. Otherwise, the adapter should choose which type to insert.

Additionally, an additional Capabilities definition called **supportedBreakpointTypes** should be added to transfer supported breakpoint types.

4. Notes

4.1. CodeTime/DebugTime Dilemma:

In DAP, the mechanism learns the capabilities of the debug adapter at the beginning of the debug session by invoking the "**Initialize**" request. At this request, the debug adapter will return back its capabilities like supportsTerminateRequest or supportsSetVariable etc…. With these capabilities, the IDE is choosing what messages to send and what messages to ignore.

We can use "**Initialize**" request to communicate with Development Tool (IDE) and provide the capabilities of the debug adapter. However, this request is not performed when developer opens and starts writing their code. Even there is a possibility that user did not configure the debug launch configuration yet and IDE is unaware of the debug adapters at all (especially at the beginning of the project).

- How could IDE know which custom fields/ breakpoint types are supported?

It could be tricky, and in VSCode while defining conditional breakpoint expressions, the option to define the condition is always visible, but in "Initialize" request the debugger choose to support conditional breakpoints or not. We could choose to always the editing interface of the custom fields, then leave it to IDE and debugger to choose the communication standards. It could be easy for breakpoint type definition but could be hard to get the possible options of custom fields dynamically from the debug adapter itself and populate the user interface.

Thus, we might come to a conclusion to inject custom fields information of the breakpoints over the "contribution" definition of the VSCode extension.