Debug Adapter Protocol (DAP)

This document is Flutter-specific. For information on the standard Dart DAP implementation, see this document.

Flutter includes support for debugging using the Debug Adapter Protocol as an alternative to using the VM Service directly, simplying the integration for new editors.

The debug adapters are started with the `flutter debug-adapter` command and are intended to be consumed by DAP-compliant tools such as Flutter-specific extensions for editors, or configured by users whose editors include generic configurable DAP clients.

Two adapters are available:

- `flutter debug_adapter`
- `flutter debug_adapter --test`

The standard adapter will run applications using `flutter run` while the `--test` adapter will cause scripts to be run using `flutter test` and will emit custom `dart.testNotification` events (described in the Dart DAP documentation).

Because in the DAP protocol the client speaks first, running this command from the terminal will result in no output (nor will the process terminate). This is expected behaviour.

For details on the standard DAP functionality, see the Debug Adapter Protocol Overview and the Debug Adapter Protocol Specification. Custom extensions are detailed below.

Launch/Attach Arguments

Arguments common to both `launchRequest` and `attachRequest` are:

- `bool? debugExternalPackageLibraries` - whether to enable debugging for packages that are not inside the current workspace (if not supplied, defaults to `true`)
- `bool? debugSdkLibraries` - whether to enable debugging for SDK libraries (if not supplied, defaults to `true`)
- `bool? evaluateGettersInDebugViews` - whether to evaluate getters in expression evaluation requests (inc. hovers/watch windows) (if not supplied, defaults to `false`)
- `bool? evaluateToStringInDebugViews` - whether to invoke `toString()` in expression evaluation requests (inc. hovers/watch windows) (if not supplied, defaults to `false`)
- `bool? sendLogsToClient` - used to proxy all VM Service traffic back to the client in custom `dart.log` events (has performance implications, intended for troubleshooting) (if not supplied, defaults to `false`)
- `List<String>? additionalProjectPaths` - paths of any projects (outside of `cwd`) that are open in the users workspace
- `String? cwd` - the working directory for the Flutter process to be spawned in
- `List<String>? toolArgs` - arguments for the `flutter run`, `flutter attach` or `flutter test` commands
- `String? customTool` - an optional tool to run instead of `flutter` - the custom tool must be completely compatible with the tool/command it is replacing
- `int? customToolReplacesArgs` - the number of arguments to delete from the beginning of the argument list when invoking `customTool` - e.g. setting `customTool` to `flutter_test_wrapper` and `customToolReplacesArgs` to `1` for a test run would invoke `flutter_test_wrapper`

`foo_test.dart` instead of `flutter test foo_test.dart` (if larger than the number of computed arguments all arguments will be removed, if not supplied will default to `0`)

Arguments specific to `launchRequest` are:

- `bool? noDebug` - whether to run in debug or noDebug mode (if not supplied, defaults to debug)
- `String program` - the path of the Flutter application to run
- `List<String>? args` - arguments to be passed to the Flutter program

Arguments specific to `attachRequest` are:

- `String? vmServiceUri` - the VM Service URI to attach to (if not supplied, Flutter will try to discover it from the device)

Custom Requests

Some custom requests are available for clients to call. Below are the Flutter-specific custom requests, but the standard Dart DAP custom requests are also documented here.

`hotReload`

`hotReload` injects updated source code files into the running VM and then rebuilds the widget tree. An optional `reason` can be provided and should usually be `"manual"` or `"save"` to indicate what how the reload was triggered (for example by the user clicking a button, versus a hot-reload-on-save feature).

```
{
    "reason": "manual"
}
```

`hotRestart`

`hotRestart` updates the code on the device and performs a full restart (which does not preserve state). An optional `reason` can be provided and should usually be `"manual"` or `"save"` to indicate what how the reload was triggered (for example by the user clicking a button, versus a hot-reload-on-save feature).

```
{
    "reason": "manual"
}
```

Custom Events

The debug adapter may emit several custom events that are useful to clients. Below are the Flutter-specific custom events, and the standard Dart DAP custom events are documented here.

`flutter.serviceExtensionStateChanged`

When the value of a Flutter service extension changes, this event is emitted and includes the new value. Values are always encoded as strings, even if numeric/boolean.

```
{
    "type": "event",
    "event": "flutter.serviceExtensionStateChanged",
    "body": {
        "extension": "ext.flutter.debugPaint",
        "value": "true",
    }
}
```