The Flutter engine adds several extensions to the Dart VM Service Protocol. The Flutter Engine specific extensions are documented here. Applications may also choose to register their own extensions.

List views: `_flutter.listViews`

Tooling requests this very early in the application lifecycle to ask for the details of the root isolate.

No arguments.

Response:

```
{
  "type": "FlutterViewList",
  "views": [
    {
      "type": "FlutterView",
      "id": "_flutterView/0x1066096d8",
      "isolate": {
        "type": "@Isolate",
        "fixedId": true,
        "id": "isolates/453229818",
        "name": "main.dart$main-453229818",
        "number": 453229818
      }
    }
  ]
}
```

Isolate Restart or Cold Reload: `_flutter.runInView`

The IDE has requested (or the user pressed 'R' from the 'flutter run' interactive console) a cold reload. For example, this happens when the user presses the green play button.

Used to "cold reload" a running application where the shell (along with the platform view and its rasterizer bindings) remains the same but the root isolate is torn down and restarted with the new configuration. Only used in the development workflow.

The previous root isolate is killed and its identifier invalidated after this call. Callers can query the new isolate identifier using the `_flutter.listViews` method.

Four arguments:

```
viewId = _flutterView/0x14ba08c68
mainScript = /path/to/application/lib/main.dart
packagesFile = /path/to/application/.packages
assetDirectory = /path/to/application/build/flutter_assets
```

Response:

```
{
  "type": "Success",
```

```
  "view": {
    "type": "FlutterView",
    "id": "_flutterView/0x104e0ab58",
    "isolate": {
      "type": "@Isolate",
      "fixedId": true,
      "id": "isolates/1056589762",
      "name": "main.dart$main-1056589762",
      "number": 1056589762
    }
  }
}
```


The object in the **view** key is constructed in the same way as the views in the **List Views** method.

Flush UI thread tasks: `_flutter.flushUIThreadTasks`

Does nothing but waits for all pending tasks on the UI thread to be completed before returning success to the service protocol caller.

No arguments.

Response:


```
{"type": "Success"}
```


Get screenshot of view as PNG: `_flutter.screenshot`

Get the screenshot as PNG of a random Flutter view on the device. The screenshot data will be base64 encoded in the response body.

No arguments.

Response:


```
{
  "type": "Screenshot",
  "screenshot": "<base64_data>"
}
```


Get screenshot of view as Skia picture: `_flutter.screenshotSkp`

Get the Skia SKP of a random Flutter view on the device. The SKP data will be base64 encoded in the response body.

No arguments.

Response:


```
{
  "type": "ScreenshotSkp",
```

```
  "skp": "<base64_data>"
}
```

Update asset bundle path: `_flutter.setAssetBundlePath`

In case of a hot-reload, the service protocol handles source code updates. However, there may be changes to assets. The DevFS updates assets in an separate directory that needs to be used by the engine.

Two arguments:

```
viewId = _flutterView/0x15bf057f8
assetDirectory = /path/to/flutter_assets
```

Response:

```
{
  "type": "Success",
  "view": {
    "type": "FlutterView",
    "id": "_flutterView/0x104e0ab58",
    "isolate": {
      "type": "@Isolate",
      "fixedId": true,
      "id": "isolates/1056589762",
      "name": "main.dart$main-1056589762",
      "number": 1056589762
    }
  }
}
```

The object in the **view** key is constructed in the same way as the views in the **List Views** method.

Get the display refresh rate: `_flutter.getDisplayRefreshRate`

Get the display refresh rate of the actual device that runs the Flutter view. For example, most devices would return an fps of 60, while iPad Pro would return an fps of 120.

One argument:

```
viewId = _flutterView/0x15bf057f8
```

Response:

```
{
  "type": "DisplayRefreshRate",
  "fps": 60.0
}
```

Get Skia SkSL shader artifacts: `_flutter.getSkSLs`

Get Skia SkSL shader artifacts from an actual device that runs the Flutter view. Such artifacts can be used to warm up shader compilations and avoid jank. One has to first tell Flutter to prepare SkSL shaders by `flutter run --cache-sksl` or `flutter drive --cache-sksl`, and trigger some shader compilations by going through some animations/transitions. Otherwise, this service protocol extension may return an empty set of SkSLs.

The key of the returned `SkSLs` map will be Base32 encoded. It should be used directly as the filename of the shader artifact. The value in that map is the Base64 encoded SkSL shader. Once decoded, it should be the content of the shader artifact file.

One argument:

```
viewId = _flutterView/0x15bf057f8
```

Response:

```
{
  "type": "GetSkSLs",
  "SkSLs": {
    "CAZAAAACAAAAAAAAAAABGAABAAOAAFAADQAAGAAQABSQAAAAAAAAAAAAAABAAAAAEAAGGAA":
"eQ=="
  }
}
```

Estimate raster cache memory usage:

`_flutter.estimateRasterCacheMemory`

Estimate the memory usage of both picture and layer raster cache. For each picture or layer cached, there is a rasterized `SkImage` of that picture or layer to speed up future draws. Only `SkImage`'s memory usage is counted as other objects in the cache system are often much smaller compared to `SkImage`. Function `SkImageInfo::computeMinByteSize` is used to estimate the `SkImage` memory usage.

One argument

```
viewId = _flutterView/0x15bf057f8
```

Response:

```
{
    "type": "EstimateRasterCacheMemory",
    "layerBytes": 40000,
    "pictureBytes": 400
}
```