

Bug Capsule

©Copyright Digital Software/Johan 'Zarkow' Munkestam

Version: 1.0.0 (Unity 6.3 LTS, 6000.3 and newer; runtime component plus editor tools; any render pipeline; Input System optional)

Introduction

Bug Capsule lets your players and testers report a bug without leaving the game. They press F8 (or tap with three fingers), the game freezes politely, and a panel shows a screenshot of the moment. They draw on it, write a title and a few lines, pick a severity and press Send. The report arrives with the screenshot, the recent log, the last seconds of input, the system and the build it came from, in a folder, a Discord channel, your own web endpoint or a GitHub issue.

Verified for 1.0.0 on Unity 6.3 LTS (6000.3.10) in the Windows editor: reports to a folder, a Discord channel, a webhook and a GitHub issue, the send window, the error-report outbox, and 93 Edit Mode tests. Device builds: to be filled in after the device pass.

Getting started

1. GameObject > Digital Software > Bug Capsule Reporter adds the reporter to the open scene (or drag Prefabs\BugCapsule.prefab in). Put it in your first scene; it survives scene loads, and a second copy removes itself.
2. Press Play, then F8. Write a title, press Send. With no settings asset the report is saved to a folder under Application.persistentDataPath\BugReports; on Windows that is C:\Users\<name>\AppData\LocalLow\<company>\<product>\BugReports.
3. To send reports somewhere else, create a settings asset (Create > Digital Software > Bug Capsule Settings), switch on the sinks you want, fill in their fields and press each sink's Test button. Then drag the asset into the reporter's Settings field.
4. Read the Token modes chapter before you give a build to anyone.

The demo scene (DemoScene\DemoScene.unity) has a spinning cube you can click while a report is sent, to show that the game keeps running.

What a report holds

report.json: id, time, title, description, severity, the optional e-mail, scene, play time, your custom fields, the build stamp and a system summary.

screenshot.png: the frame before the panel opened, with the player's drawings, at most 8K per side.

log.txt: the most recent log messages with stack traces for errors and exceptions, kept in a ring from the start of the game, so messages from before the key press are there.

system.json: operating system, device, processor, memory, graphics device and API, screen and refresh rate, quality level, frame-rate settings, language, Unity version, application version and build GUID.

inputs.json: the Input System controls that changed during the last 60 seconds (Input System projects only).

Your attachments, for example a save file.

The player can untick the screenshot, the log, the system information, the recent inputs and the game data (your fields and attachments) before sending. The panel says: nothing leaves this device until you press Send, and only what is ticked is included.

Every report has a short id (12 characters). It is shown in the Discord message, the GitHub issue, the webhook entry and the folder name, so one report can be found everywhere it went.

The report panel

Left: the screenshot with the drawing tools. Rectangle and arrow: click on the thing to point at and drag. Pen: draw freely. Pin: click to drop a numbered pin; a matching [1], [2] ... line is added to the description for you to complete. Undo and Clear remove marks.

Right: title, description, severity (Low, Normal, High, Crash), an optional e-mail address, the "send with the report" toggles and the privacy line. The e-mail address and the last severity are remembered on the device.

The panel is built from code: no prefab to style, no TextMeshPro, any render pipeline, any scene. While it is open, the cursor is shown and unlocked, game time is stopped (Pause While Open), and an EventSystem is added if the scene has none. When it closes, the cursor lock and visibility, the time scale and the EventSystem are put back exactly as they were.

The reporter component

Settings

The settings asset with the sinks and the token mode. Empty: reports go to the folder sink only.

Opening the panel

Key Binding: an Input System binding path, <Keyboard>/f8 by default (F12 is Steam's screenshot key). Legacy Key: used when the project runs on the old Input Manager. Three Finger Tap: opens the panel on touch screens. Pause While Open: sets Time.timeScale to 0 while the panel is open.

From your own UI, for example a "Report a bug" button in the pause menu:

```
BugCapsuleReporter.Instance.Open();
```

Capture

Screenshot, Screenshot Scale (0.1 to 1), Log, System, Build Stamp: what is captured. Input Trace and Input Trace Seconds (5 to 300): the input recording; Include Pointer Motion also records mouse and pen movement, which is many events per second. Log Max Messages and Log Max Kilobytes: the size of the log ring.

Lifetime

Keep Across Scenes: DontDestroyOnLoad, on by default.

Sinks

A sink is a place reports go. A report goes to every enabled sink: file sinks first, then the text-only GitHub issue, which can then say where the files went. A failing sink never stops the others.

Folder

Saves each report on the player's own device as a folder named <date>-<time>-<id>, holding the files and a zip, under Application.persistentDataPath\<Subfolder> (BugReports by default). Nobody else is told about it; useful when you test yourself or collect reports by hand.

Discord

Posts a message with the title, severity and id, the screenshot shown inline and the zip, to a channel. Webhook URL: in Discord, Edit Channel > Integrations > Webhooks > New Webhook > Copy Webhook URL. Username: the sender name shown on the posts.

Player text is cleaned before it is posted: @ becomes (at), control and invisible characters are dropped, and mentions are switched off, so no report can ping your server. Long text is cropped with a note that the full text is in report.json. Discord accepts 10 MB per message; Bug Capsule always stays within that, and sends the screenshot as a downscaled JPEG when the PNG is too big.

Webhook

A multipart POST to your own endpoint, or to a relay in front of Discord, Jira, Trello or anything else. Parts: report (report.json), screenshot (the PNG, as its own part so a relay can show it without unzipping) and package (the zip). Key: optional text sent as the X-BugCapsule-Key header, so your endpoint can ignore posts without it. It ships inside the build, so it keeps out casual noise, not a determined player. For a first test, a free <https://webhook.site> address shows what arrives.

GitHub

Opens an issue with the report text, the system summary and the last lines of the log (Log Excerpt Lines). GitHub's API cannot attach files, so the issue says where the files went and links the Discord message when there is one. Owner and Repository: from the address github.com/<owner>/<repository>. Token: a fine-grained personal access token (github_pat_...) for that one repository with Issues: Read and write, nothing else. Labels: comma-separated, created by GitHub when missing. Tester mode only.

Package size

Max Package MB (default 9): when a package is bigger, the input trace, then the log (first cut to its last 64 KB, then left out), then attachments, largest first, are trimmed until it fits, and the report says what was trimmed.

Testing the sinks

Each sink in the settings inspector has a Test button that sends a sample report from the editor. Under the fields, the inspector lists entries that look wrong, such as an e-mail address where the GitHub owner belongs or a Discord URL that is not a webhook.

Token modes and secrets

Everything in the settings asset ships inside your build, and players can extract it. Bug Capsule has two modes for that reason.

Public (the default): for anything released. Folder, Discord and webhook only. A Discord webhook URL in a build is tolerable: the worst case is spam in one channel, and you revoke it by deleting the webhook in Discord. GitHub is never used.

Tester: every sink, GitHub included, for closed playtests with people you trust.

A build guard checks every Bug Capsule settings asset in the project before each build, whether a scene uses it or not, because a forgotten asset could still be pulled in. A Public-mode asset that holds a GitHub token stops the build. A Tester-mode asset with a token in a non-development build gives a warning.

Keep the settings asset that holds your real URLs and tokens in a folder your version control ignores, and do not commit it.

Sending in the background

When the player presses Send, the panel closes at once and the game gets its cursor and time back. A small window in the lower right corner shows each sink: waiting, preparing, connecting, sending with percent, waiting for reply, sent or failed. It hides 3 seconds after success and 10 seconds after a failure. Reports sent while another is on its way wait in a queue. Bug Capsule sets no timeout of its own, so a large upload on a slow line is not cut off.

The window's Abort and Close buttons need a visible cursor and an EventSystem in the scene; the EventSystem Bug Capsule adds for the panel is removed with the panel. In a game with a locked cursor, call AbortSend from your own input if you want an abort:

```
BugCapsuleReporter.Instance.AbortSend();
```

Pausing after Send

Some games would rather not drop the player straight back into the action. ReportSubmitted is raised as the panel closes; open your pause menu there:

```
BugCapsuleReporter.Instance.ReportSubmitted += report => PauseMenu.Show();
```

Quitting while a report is sent

On desktop, quitting waits until the reports in progress and in the queue are sent; the send window says the game closes when this is done. Abort skips the rest and quits. On mobile and WebGL the operating system or the browser may close the game without waiting.

Error reports

When your own ReportSubmitted or ReportSent handler throws, Bug Capsule logs the exception and keeps an error report about it in

Application.persistentDataPath\BugCapsuleOutbox. It is sent when the game quits on desktop, otherwise at the next start, and removed once a sink has it.

From game code

Fields go into every report from the moment they are set; null removes one:

```
BugCapsule.SetField("level", "3-2");
```

Attachments are read when the report is sent, so a save file is current. Return null to skip the file. The same name again replaces the provider; RemoveAttachment removes it:

```
BugCapsule.AddAttachment("save.json", () => File.ReadAllBytes(savePath));
```

Events on BugCapsuleReporter.Instance: Opened, Closed, ReportSubmitted(ReportData), ReportSent(ReportData, List<SinkResult>) with each sink's result. Properties: IsOpen, IsSending. Methods: Open, Close, AbortSend.

Static state is reset at SubsystemRegistration, so Enter Play Mode without a domain reload starts clean. Add fields and attachments from code that runs after that, for example in Awake or Start.

Build stamp

When the build carries a build-info.json in StreamingAssets, as written by Ship It from the same publisher, the report holds its version, git hash, branch and build time, so you know exactly which build a report came from.

Platforms

Windows, Linux and macOS: every sink. Android and iOS: every sink; the report key is the three-finger tap. WebGL: browsers only allow requests to servers that permit them (CORS),

and files saved there stay in the browser's storage, so use the webhook sink with your own endpoint or relay that answers with an Access-Control-Allow-Origin header, and forward to Discord or GitHub from there.

Input

With the Input System package (1.0 or newer) installed and active, the key is an Input System binding and the input trace is recorded. On the old Input Manager, Legacy Key opens the panel and there is no input trace. With Active Input Handling set to Both, either key opens the panel and the input trace is recorded.

Further support

Head over to <https://community.digitalsoftware.se/> or the Discord at <https://discord.gg/fyqtxZt>