Incorporating GMEXT-Elements into a GameMaker Project

Gamemaker · https://namazustudios.com/docs/game-engine-client-support/gamemaker/incorporating-gmext-elements-into-a-gamemaker-project/

A walkthrough for adding YoYoGames' GMEXT-Elements extension to any GameMaker Studio project by vendoring the source directly from GitHub.

> A Marketplace listing for GMEXT-Elements isn't available yet — vendoring from GitHub is the only option today. This doc should get a Marketplace-install alternative once that listing exists.

Why vendor the source

Reasons to pull the source in directly rather than depending on a prebuilt drop-in:

- You need a fix or feature that's landed on `main` but hasn't been tagged in a stable point yet.
- You want to read/patch the actual source (a compiled-in version isn't easily diffable).
- You're scripting project setup and want reproducible, version-pinned source rather than a manual install step.

Trade-off: you're now responsible for tracking upstream changes yourself.

---

1. Understand what you're actually importing

Don't just copy the whole GMEXT-Elements repo wholesale — most of it is the *demo project* GameMaker uses to showcase the extension, not the extension itself. Clone it first and look:

```
git clone --depth 1 https://github.com/YoYoGames/GMEXT-Elements.git
```

The relevant subtree is `source/Elements_gml/`, which is itself a full `.yyp` demo project. Inside it:

Path	What it is	Do you need it?
`extensions/Elements/Elements.yy`	The extension resource — just holds config **options** (server URLs, ports, debug flag). No code lives here.	Yes
`objects/obj_elements_core/`	Singleton that owns the HTTP async event, auth token storage, request bookkeeping.	Yes
`objects/obj_elements_crossfire/`	Singleton that owns the WebSocket async event and dispatches Crossfire messages.	Yes, if you want realtime/matchmaking
`scripts/elements_rest_api/`, `elements_rest_helpers/`, `elements_rest_schemas/`	The generated REST client — every `elements_*` HTTP wrapper function and request/response schema constructor.	Yes
`scripts/elements_crossfire_api/`, `elements_crossfire_client/`, `elements_crossfire_helpers/`	The Crossfire WebSocket client — connect/matchmake/send/receive.	Yes, if you want realtime/matchmaking
`extensions/Elements/docs/`	A static HTML help viewer (fonts, CSS, JS) for GameMaker's in-IDE "view docs" button.	No — skip it, it's ~4 MB of dead weight for a game build. The real docs are the GitHub wiki.
`objects/obj_game`, `objects/obj_gm_button`, `objects/obj_gm_text`, `objects/obj_gm_textbox`, `objects/obj_elements_crossfire_*` (the `Mouse_7`-only ones), `objects/obj_elements_rest_*`, `rooms/rm_main`, `fonts/fnt_gm_*`	Demo-project UI: buttons and labels used to click through the sample.	No — these only exist to make the demo clickable.

The key realization: the "extension" resource itself (`Elements.yy`) is nearly empty — it's just a place for `server_rest_url` , `server_crossfire_url` , `server_crossfire_port` , and `debug_logging` to live as extension options. All the actual functionality is in plain GML scripts and two small controller objects, which you import as ordinary project resources.

---

2. Copy the resources into your project

GameMaker's `.yy` / `.yyp` files are JSON-with-trailing-commas. You can hand-edit them; the IDE re-normalizes on next save. Copy in this shape:

```
your_project/
  extensions/Elements/Elements.yy
  objects/obj_elements_core/{obj_elements_core.yy, Create_0.gml, CleanUp_0.gml, Other_62.gml}
  objects/obj_elements_crossfire/{obj_elements_crossfire.yy, Create_0.gml, Other_68.gml}
  scripts/elements_rest_api/{elements_rest_api.yy, elements_rest_api.gml}
  scripts/elements_rest_helpers/{elements_rest_helpers.yy, elements_rest_helpers.gml}
  scripts/elements_rest_schemas/{elements_rest_schemas.yy, elements_rest_schemas.gml}
  scripts/elements_crossfire_api/{elements_crossfire_api.yy, elements_crossfire_api.gml}
  scripts/elements_crossfire_client/{elements_crossfire_client.yy, elements_crossfire_client.gml}
  scripts/elements_crossfire_helpers/{elements_crossfire_helpers.yy,
elements_crossfire_helpers.gml}
```

Every `.yy` resource file has a `"parent"` field pointing at a folder, e.g.:

```
"parent": { "name": "Rest", "path": "folders/Elements/Rest.yy" }
```

Folders in the `.yyp` format are virtual — `folders/Elements/Rest.yy` is not a real file on disk anywhere, it's just an identifier string. You don't need to create anything at that path; you only need a matching entry in the project's `.yyp` `Folders` array (see next step). Leave the copied `.yy` files' `parent` fields untouched — they already point at the right virtual paths.

---

3. Register everything in the `.yyp`

Two arrays in your project's `.yyp` need new entries: `Folders` (so the IDE shows a folder in the asset tree) and `resources` (so the resources actually load).

`Folders` — add the folder chain the copied resources expect:

```
{"$GMFolder":"","%Name":"Elements","folderPath":"folders/Elements.yy","name":"Elements","resourceT
ype":"GMFolder","resourceVersion":"2.0",},
{"$GMFolder":"","%Name":"Crossfire","folderPath":"folders/Elements/Crossfire.yy","name":"Crossfire
","resourceType":"GMFolder","resourceVersion":"2.0",},
{"$GMFolder":"","%Name":"Rest","folderPath":"folders/Elements/Rest.yy","name":"Rest","resourceType
":"GMFolder","resourceVersion":"2.0",},
```

`resources` — one entry per resource, `name` matching the resource's internal name and `path` matching where you put its `.yy` file:

```
{"id":{"name":"Elements","path":"extensions/Elements/Elements.yy",},},
{"id":{"name":"obj_elements_core","path":"objects/obj_elements_core/obj_elements_core.yy",},},
{"id":{"name":"obj_elements_crossfire","path":"objects/obj_elements_crossfire/obj_elements_crossfi
re.yy",},},
{"id":{"name":"elements_rest_api","path":"scripts/elements_rest_api/elements_rest_api.yy",},},
{"id":{"name":"elements_rest_helpers","path":"scripts/elements_rest_helpers/elements_rest_helpers.
yy",},},
{"id":{"name":"elements_rest_schemas","path":"scripts/elements_rest_schemas/elements_rest_schemas.
yy",},},
{"id":{"name":"elements_crossfire_api","path":"scripts/elements_crossfire_api/elements_crossfire_a
pi.yy",},},
{"id":{"name":"elements_crossfire_client","path":"scripts/elements_crossfire_client/elements_cross
fire_client.yy",},},
{"id":{"name":"elements_crossfire_helpers","path":"scripts/elements_crossfire_helpers/elements_cro
ssfire_helpers.yy",},},
```

Placement inside the `resources` array doesn't matter functionally — GameMaker doesn't care about ordering — but grouping by type (near other extensions, near other objects, near other scripts) keeps diffs readable.

Validate before opening the IDE. The `.yyp` is JSON except for trailing commas, so you can sanity-check your edit without launching GameMaker:

```
import re, json
text = open("YourProject.yyp").read()
cleaned = re.sub(r',s*([}]])', r'1', text)
data = json.loads(cleaned)  # raises if malformed
```

```
names = [r["id"]["name"] for r in data["resources"]]
assert len(names) == len(set(names)), "duplicate resource name"
```

A malformed `.yyp` or a duplicate resource name will cause GameMaker to fail to open the project or silently drop a resource — catching it before opening the IDE saves a round trip.

---

4. One likely fix: make the core singleton persistent

Both `obj_elements_core` and `obj_elements_crossfire` are **lazily instantiated singletons** — you never place them in a room. The pattern (from `elements_rest_helpers.gml`):

```
function _elements_get_singleton(_where) {
    static instance = instance_create_depth(0, 0, 0, obj_elements_core);
    with (instance) return self;
}
```

The `static` keyword means this instance is created exactly once, the first time any `elements_*` function runs. `obj_elements_crossfire` ships `persistent: true` in the stock extension, but **`obj_elements_core` ships `persistent: false`.** If your project has more than one room and the room changes after the singleton is created, the non-persistent instance is destroyed — but the `static` variable still holds a reference to it, so every subsequent call silently operates on a dead instance (auth tokens, in-flight request bookkeeping, all gone).

If your project is single-room, this doesn't matter. If it's multi-room (a login screen, a menu, gameplay rooms, etc.), set it persistent:

```
// objects/obj_elements_core/obj_elements_core.yy
"persistent": true,
```

---

5. Configure the extension options

Open the project in GameMaker, find the **Elements** extension under Extensions in the asset tree, and fill in:

- `server_rest_url` — your Elements REST API base URL

- `server_crossfire_url` — your Elements Crossfire WebSocket host
- `server_crossfire_port` — usually `443` for WSS
- `debug_logging` — `True` while integrating, so failed requests and Crossfire phase transitions print to the debug console

These are read via `extension_get_option_value("Elements", "...")` inside the vendored scripts — don't hardcode URLs elsewhere in your game code.

---

6. Verify it works

1. Open the project in GameMaker. It should load with no missing-resource errors and no red squiggles in the copied scripts.
2. Drop a test call somewhere reachable at startup, e.g. in a controller object's Create event:
3. Run the game. `debug_logging = True` should print the outgoing request and response in the console. A `_code` of `200` (or a clear connection error if your server isn't running yet) confirms the plumbing — extension options, singleton creation, async event handling — is wired correctly.

```
elements_get_application("your-app-id", function(_code, _data, _request) {
    show_debug_message($"Elements reachable: {_code}");
});
```

---

7. Using it: REST and Crossfire basics

REST — every endpoint is a thin async wrapper, callback-based:

```
elements_create_username_password_session(_session_request, function(_code, _data, _request) {
    if (_code == 200) {
        _elements_request_auth_set_token("auth_bearer", _data.sessionSecret);
        // now every endpoint that declares "auth_bearer" as a security scheme is authenticated
    }
});
```

Crossfire — connect, then either find/create/join a match:

```
elements_crossfire_set_identity(profile_id, session_secret);
elements_crossfire_events_on_connected_callback(function() {
    elements_crossfire_create_match("my_config");   // host path — server responds with a join
code
});
elements_crossfire_events_on_created_callback(function(_msg) {
    show_debug_message("Room code: " + _msg.joinCode);
});
elements_crossfire_connect();
```

Guests call `elements_crossfire_join_match_by_code(_join_code)` instead of `elements_crossfire_create_match`, and listen for `elements_crossfire_events_on_matched_callback` instead of `_on_created_callback`.

Once connected, `elements_crossfire_send_string_broadcast(_text)` pushes a message to every participant in the match — this is the "push an event to all connected clients" mechanism for anything server-driven (e.g., "your generated content is ready").

Gotchas learned the hard way

- **Don't hand-patch a feature before checking upstream.** GMEXT-Elements is under active development. A capability that looks missing from the version you last read the source of — whether that was an older commit you vendored from, or a plan doc written a few weeks ago — may already exist on `main` — always re-diff against the current GitHub source before writing a patch, or you'll duplicate work upstream already did.
- **The extension resource itself has an empty `files` list.** If you're used to native extensions with per-platform binaries, this one is disorienting — there's nothing to compile, it's pure GML. Don't go looking for a `.dll` / `.so` / `.dylib` to place; there isn't one.
- **Don't import the demo project's UI objects.** It's tempting to copy everything under `source/Elements_gml/objects/` for completeness, but half of them (`obj_gm_button`, `obj_elements_rest_create_profile`, etc.) exist only to make the sample clickable and add dead weight and demo-specific coupling to your project.
- **Folder paths in `.yy` files don't need real files.** If you see `"path": "folders/Elements/Rest.yy"` in a copied resource, that's a virtual identifier consumed only by the `.yyp`'s `Folders` array — don't

go looking for (or creating) an actual file there.