# Canonical types to support (recommended)

Treat these as your public, cross-platform types:

- **text/plain; charset=utf-8** — plain text (always include)
- **text/html** — HTML fragment (also expose a plain-text fallback)
- **text/uri-list** — list of URIs (primarily `file:///…`; may also include `http(s)://`)
- **image/png** — lossless image; ideal interchange for transparency
- **image/jpeg** — common lossy fallback
- **image/bmp** — if you decide to surface raw DIB/BMP drops as-is
- **image/svg+xml** — vector images (nice-to-have)
- **application/json** — lightweight structured data (nice-to-have)
- **application/x-virtual-files** — *your* abstraction for "promised" files (deferred rendering)
- **application/x-color** — simple color payload (e.g., `#RRGGBBAA` or CSS color) (nice-to-have)

Also support the **operation** (copy/move/link/none) in a platform-neutral way.

# Map native formats ⇄ canonical types

## Windows (OLE DnD)

- Plain text → `CF_UNICODETEXT` ⇄ **text/plain; charset=utf-8**
- HTML → registered `"HTML Format"` ⇄ **text/html** (parse Start/EndFragment; also offer plain text)
- URLs → `"UniformResourceLocatorW"` ⇄ **text/uri-list** (single URL)
- Files on disk → `CF_HDROP` ⇄ **text/uri-list** (`file:///…` for each item)
- Shell items → `IShellItemArray` ⇄ **text/uri-list** (resolve to file URLs when possible)
- PNG → registered `"PNG"` ⇄ **image/png`
- Bitmaps → `CF_DIBV5`/`CF_DIB` ⇄ **image/bmp** (you may convert to **image/png** internally)
- Virtual files (not on disk yet) → `FileGroupDescriptorW` + `FileContents` ⇄ **application/x-virtual-files** (list of descriptors + lazy `IStream` suppliers)
- Preferred/performed effects → `"Preferred DropEffect"` / `"Performed DropEffect"` (map to your copy/move/link result)

## macOS (NSPasteboard / UTType)

- Plain text → `public.utf8-plain-text` ⇄ **text/plain; charset=utf-8**
- HTML → `public.html` ⇄ **text/html** (WebArchive exists, but HTML covers most)
- URLs → `public.url` (NSURL) ⇄ **text/uri-list** (include title separately if you like)
- Files → `public.file-url` (one or many) ⇄ **text/uri-list**
- PNG/JPEG/SVG/BMP → `public.png`/`public.jpeg`/`public.svg-image`/`public.bmp` ⇄ **image/***

- Virtual files → **File Promises** (`NSFilePromiseProvider` / `NSFilePromiseReceiver`) ⇄ **application/x-virtual-files**
- Operations → `NSDragOperationCopy/Move/Link` ⇄ your normalized op

## Linux (X11 XDND / Wayland + GTK/Qt/FDO)

- Plain text → `text/plain;charset=utf-8` ⇄ **text/plain**
- HTML → `text/html` ⇄ **text/html**
- Files/URIs → `text/uri-list` (newline-separated URIs) ⇄ **text/uri-list**
- Images → `image/png`, `image/jpeg`, sometimes `image/bmp`, `image/svg+xml` ⇄ **image/***
- Virtual files → toolkit-specific streams (GTK GIO streams, Qt MIME providers) ⇄ **application/x-virtual-files**
- Operations → copy/move/link per XDND/Wayland action mask ⇄ your op

# Extraction & production rules (portable behavior)

**text/plain**

- Always UTF-8 internally. On Windows, convert `CF_UNICODETEXT` (UTF-16LE) ⇄ UTF-8.

**text/html**

- Prefer the HTML fragment (Windows "HTML Format" has Start/EndFragment markers; macOS/Linux give raw HTML).
- Also provide a plain-text fallback by stripping tags.

**text/uri-list**

- Parse per RFC 2483 style: `\r\n` or `\n` separated, ignore `#` comment lines, entries are absolute URIs.
- For `file:///` URIs, decode percent-escapes and normalize to platform paths.
- Be prepared for **http(s)://** URLs too (browsers do this).

**image/png / image/jpeg / image/bmp / image/svg+xml**

- If multiple are offered, **prefer PNG**, then JPEG, then SVG (if your consumers can render it), then BMP.
- On Windows, `CF_DIBV5`/`CF_DIB` can be wrapped into a BMP *file* or converted to PNG internally.

**application/x-virtual-files** (deferred rendering)

- Normalize to: a list of `{ name, size?, suggested_mime?, openStream(index)->Reader }`.
- Map from:
  - Windows: `FileGroupDescriptorW` (names/size/etc) + `FileContents` (per-file `IStream`)
  - macOS: `NSFilePromiseReceiver` (receive to a folder, or vend a stream per file)
  - Linux/GTK/Qt: per-file streams promised by the toolkit
- Expose a **pull** API so you only materialize bytes if/when the drop is accepted.

**Operation (copy/move/link)**

- Normalize from: Windows `DROPEFFECT_*`, macOS `NSDragOperation*`, XDG/Wayland actions.
- Let callers request a preferred op; intersect with source's allowed mask.

# API shape that works well

Design a tiny, capability-oriented interface:

```
// Discovery
offer.types(): string[]                    // e.g., ["text/plain; charset=utf-8","text/uri-list","image/png"]
offer.has(type: string): boolean
offer.pickImage(): Promise<{ type: "image/png"|"image/jpeg"|"image/svg+xml"|"image/bmp", data: Uint8Array }>
offer.getText(): Promise<string>
offer.getHtml(): Promise<{ html: string, plain?: string }>
offer.getUris(): Promise<string[]>       // URIs, not paths
offer.getVirtualFiles(): Promise<Array<{ name: string, size?: number, openStream(): Promise<Readable> }>>

// Operation
offer.allowedOps(): ("copy"|"move"|"link")[]
offer.chooseOp(desired: "copy"|"move"|"link"): "copy"|"move"|"link"|"none"
```

On the **source** side, mirror it with `DataPackage`-like setters that accept your canonical types and let the platform bridge handle the rest.

# "Baseline" vs "Nice-to-have"

- **Baseline (do these first):**
  - `text/plain; charset=utf-8`
  - `text/uri-list`
  - `image/png` (and accept `image/jpeg`)
  - `text/html`
  - **application/x-virtual-files** (if you ever expect Outlook/email/archives/browser drags)

    - op mapping (copy/move/link)
- **Nice-to-have:**
    - `image/svg+xml`, `image/bmp`
    - `application/json`
    - `application/x-color` (e.g., `#RRGGBB` or `rgb(a)`)

# Edge cases & tips

- **Security:** treat URIs carefully. Don't auto-fetch network URLs; validate paths; resolve symlinks if your app writes.
- **Huge drops:** cap total bytes you'll buffer (especially for images/virtual files). Stream when possible.
- **HTML:** Windows' "HTML Format" includes headers; extract the fragment by markers (or byte offsets) for consistent results.
- **File paths vs URIs:** Always normalize to **URIs** at your boundary; convert to paths only at the platform edge that needs them.
- **Testing:**
    - Windows: ClipSpy / your own diagnostic target (dump all formats).
    - macOS: Pasteboard inspector, Finder/Preview (for file URLs and images), small Cocoa tester app.
    - Linux: `gtk3-demo` DnD examples / `xclip` / small GTK test.

If you'd like, I can sketch the exact **mapping table code** for your Zig layer (Windows OLE ⇄ canonical) and outline the parallel macOS/Linux adapters so you've got a clear to-do list per platform.