Web Wide Worlds

Worlds as Documents on the Open Web

Dylan Z. Baker
Five Squared Interactive, LLC

Version 2.0, 12 November 2026

Contents

1.0 Abstract

2.0 What people want, and what stood in the way

3.0 Principles

4.0 The spine: files, services, shells

5.0 Being there together

6.0 Persistence without a process

7.0 Doors and the graph

8.0 Worlds with shared state

9.0 Making worlds with the tools you already use

10.0 Hosting, and what happens if Five Squared disappears

11.0 Security

12.0 Applications

13.0 Conclusion

Works cited

What changed since version 1.1

1.0 Abstract

A world is a set of files at a web address, much like a web page. Anyone can open a public world in a browser without installing an application or creating an account. People share it by sending a link. Worlds connect to other worlds through doors, just as pages connect through hyperlinks. A world is independent of any particular host, engine or company—including ours—so its content can outlast the organizations and software that first brought it to life.

Web Wide Worlds is the collection of MIT-licensed components being built to make this possible. VEML is a small document format that describes a world and references ordinary glTF models. WebVerse is the browser-based shell that opens a world from any address on a phone, laptop or VR headset. WorldSync is a synchronization protocol that lets people visiting the same world see one another without requiring a dedicated server for every world. WorldOS is a runtime for worlds whose shared state requires an authoritative process. WorldKit is the family of exporters that turn scenes made in familiar tools into worlds. WorldHub is a commercial service for hosting worlds, signing users in and indexing worlds published anywhere. It is an amenity, not a dependency: no world should require it.

As of November 2026, the working pieces include the Blender exporter, which publishes a scene as a world in one step; a hosting service that lets its maker control who may enter; WebVerse, which opens published worlds on phones, laptops and in VR; WorldSync's serverless mode, including its introduction service and relay, which lets visitors to the same link see one another; doors that open on click or tap; and WorldHub's directory and sign-in service. The rest of this paper describes the intended design. Where the design is ahead of the implementation, that distinction is stated explicitly.

Version 1.1 described a five-layer framework with a message broker at its center. This version replaces that architecture with a simpler model: files, services and shells.

2.0 What people want, and what stood in the way

People who create three-dimensional places want two straightforward things: to show someone a place and let that person experience it, and to be there together. A Blender artist wants a friend to stand inside a room, not just look at a rendering. A designer wants a client to walk through a kitchen on a phone before construction begins. A group of friends wants a place of their own where they can meet, at an address they control.

For two decades, the available ways to share these places have come with friction. A rendering is flat. A video cannot be explored. A model viewer lets someone rotate an object, but not inhabit a space. A game build requires a download, is tied to a platform and rarely connects to anything outside itself. Shared, walkable worlds do exist, but they are usually confined to platforms: visitors must install an app and create an account, while creators depend on a company's policies and servers. If that company changes direction or shuts down, years of work may become inaccessible. These platforms can also incur ongoing costs while nobody is visiting, because their simulations continue running.

The web solved a similar problem for documents. A page is a collection of files at an address. Anyone can publish one without permission, and anyone with a browser can open it. Pages link to other pages without requiring approval. An unvisited page costs almost nothing to host, and its content can outlive the software that first displayed it.

There is another requirement: people should be able to keep using the tools and worlds they already have. Web Wide Worlds is designed so a scene made in a familiar tool can become a world without forcing its creator into a new editor. It is also designed to make existing environments—from games and earlier projects—available as places people can visit and connect to other worlds, rather than requiring every environment to be rebuilt from scratch.

3.0 Principles

The design is guided by nine principles. The first seven adapt ideas Tim Berners-Lee described in his design notes and in Weaving the Web [1]. The final two address requirements that are especially important for shared, persistent environments. These principles are not slogans; they are constraints on what the system should and should not become.

- **Universality.** A world must be openable by anyone with a browser: no install, no account, no plugin. The test is whether a stranger with a link can see it.
- **Permissionless publication.** Anyone can publish a world without asking Five Squared or anyone else. The test is whether someone can publish a fully working world we have never heard of.
- **Links need no consent.** I can put a door to your world in mine. You cannot stop me and do not have to approve it. This was the web's sharpest break from earlier hypertext systems, and the reason it scaled where they did not.
- **Rule of least power.** Prefer the least powerful language that does the job. Declarative markup first; scripting as the escape hatch. This is also the strongest argument for keeping the format thin.
- **Modularity and layer separation.** Files do not know about services; services do not know about the pages that open them. Anyone can replace one tier and keep the rest.
- **Independent invention.** If someone built this separately, the two systems should interoperate. Every reuse of HTTP, addresses, glTF and HTML passes this test; every bespoke element is a bet that someone else converges on it.
- **Tolerance. A world containing elements a shell does not recognize should still open. Unknown elements and attributes are ignored rather than treated as fatal errors. VEML 4.0 makes this an explicit property of the format, helping worlds remain usable as implementations evolve.**
- **Zero cost at rest.** A world nobody visits costs nothing. The web had this by default; the shared 3D worlds of the last decade traded it away for always-on simulation. We restore it rather than invent it.
- **Content outlives its runtime. A world should be able to carry or identify the exact runtime version it needs, including an integrity hash and a fallback copy stored alongside the world. Because we are currently a single implementer, this principle is particularly easy to defer in the name of convenience—and particularly important to preserve.**

4.0 The spine: files, services, shells

The system has three tiers. A world at rest is a collection of files. Services are optional; most worlds need none of the stateful services. A shell is the interface through which a person or program reads, edits or experiences a world. The model draws on Plan 9, the Bell Labs operating system that treated files as a universal interface: files provide the common foundation, services can run wherever appropriate, and different shells can interact with the same underlying world.

Tier	What it is	What is in it
Files	A world at rest is bytes	VEML documents, glTF models, a manifest, optional sidecars
Services	Optional, and mostly not needed	An introduction service, a relay, a persistence endpoint, the WorldOS runtime
Shells	How you interact with a world	WebVerse on the StraightFour engine; ordinary web pages; headless sessions; an editor

4.1 Files

A world is a folder of files. Its central text document is written in VEML [2], the Virtual Environment Markup Language. VEML describes the environment and points to the resources it uses: the sky, models, visitor start positions and doors to other worlds. Those models use glTF [3], the asset format already supported by Blender and modern 3D tools. VEML plays a role for worlds similar to the role HTML plays for web pages: it describes structure and references assets rather than embedding geometry. This keeps the world description small and lets creators use established tools and formats. The demonstration world linked from our site, for example, consists of a 787-byte document, one model and one sky image.

A world can also be a tree of documents rather than a single document. One VEML document can include another from any address. A city, for example, could be assembled from buildings that remain independent documents on separate hosts, with their own owners, edit histories and synchronization. Including is different from linking: a door takes a visitor somewhere else; an include incorporates external content into the current world, much as an image or frame is incorporated into a web page.

Because a world is made of files, it does not need a server process of its own. It is served like a web page, using the same addressing, hosting and caching infrastructure. An unvisited world therefore has no simulation process to keep running. A shell that encounters content it does not understand can leave that content untouched, allowing newer worlds to remain at least partially usable in older shells.

4.2 Services

There are four kinds of service, and a world names the ones it uses in its own document. An introduction service helps two visitors' pages find each other; it is tiny, stateless and generic, and one public instance could serve thousands of worlds. A relay forwards bytes for the minority of networks that cannot connect directly. A persistence endpoint accepts the world's checkpoint, and is usually just a bucket. The WorldOS runtime, described in section 8, is for worlds with shared state that something must own. A world that is only visited needs none of these. A world that is visited together needs the first, and sometimes the second.

4.3 Shells

WebVerse [2] is a web page that loads a 3D engine and opens the world at the address in its link. The browser provides much of the surrounding interface: the Back button, history, bookmarks and address bar. WebVerse runs on phones, laptops and headset browsers, and it is the shell that published links open by default. Under it is StraightFour, the engine responsible for rendering, loading, physics and the sandbox that isolates a world's scripts from the visitor's device and data. A creator encounters the engine directly only when building a shell of their own.

A shell does not have to be three-dimensional. An ordinary web page that lists a world's contents, a form that edits them, a moderation panel or a world embedded in an article can all act as shells. Any shell that writes to a world uses the same session interface as a 3D shell. A shell also does not need a person or a screen: headless sessions can support conformance checks, preview generation, monitoring and programmatic authoring.

Reading a world requires no session; anyone can fetch its files and open it. Writing to a world takes place through a session, but a session is not the same thing as an avatar. It may represent a person, a build tool, a monitor or another program. The design gives the world owner a way to grant capabilities to individual sessions, with authorization recorded in the world itself so it can survive the disappearance of any service provider, including us. The live implementation is simpler: the host of a published world currently allows either anyone with the link or only people named by its maker.

5.0 Being there together

When two people open the same world, their pages discover one another and exchange changes through WorldSync [2]. The world files themselves are not involved in this exchange. Our services provide introduction and, for networks that cannot establish a direct connection, a relay that forwards data without interpreting it. This shared-visit capability is live today, with our introduction service as the default.

WorldSync is designed around a protocol contract rather than a particular transport or central authority. The protocol document defines how participants join, leave and rejoin; how ordered operations are applied consistently; how object leases are acquired, renewed and allowed to expire; how checkpoints are created and who may write them; what presence information may be exchanged; and how

participants behave when a connection or service is degraded. Each transport is specified separately. WebRTC data channels [4] and WebSocket are the first planned bindings; others can be added when a deployment has a reason to use them.

The trust model—not the complexity of a world—determines which synchronization tier it needs. In the local tier, an interaction affects only the visitor who performs it; opening a door, for example, requires no protocol. In the shared tier, everyone needs to see the same event, but a server does not need to calculate the outcome. Participants agree on event ordering, and one visitor's page temporarily holds authority, handing it to another participant when they leave. In the authoritative tier, a trusted process is necessary because the world contains hidden information, scarce resources or adversarial participants. That level of trust is appropriate for competitive games, but it is often unnecessary in galleries, showrooms and other shared spaces. The shared and authoritative tiers use the same protocol and engine. Where authority runs is a deployment choice: on a visitor's device, in a small service dedicated to a world, or in WorldSync's long-running server mode. The latter is a standalone process, but it is still WorldSync—not the WorldOS runtime described in section 8.

WorldSync is the default, not a requirement. A world's document names its synchronization service and the protocol it speaks, so a world may name another protocol, and a shell uses the first one it understands.

6.0 Persistence without a process

A persistent world does not necessarily need a process running around the clock. It needs somewhere to store its state. Much of the cost of hosting persistent worlds comes from treating those two requirements as if they were the same.

World state is represented as a serializable document. The design separates state exchange into two paths operating at different timescales. The live path carries edit operations through WorldSync as they happen. These are small structured changes—often a few hundred bytes each—and can be far smaller than the stream of position updates from a single moving avatar. The durable path periodically writes a checkpoint of the complete state to the world's host, typically after a pause or after a configured number of operations. A twenty-minute building session might therefore produce only a handful of durable writes, rather than thousands, and no process needs to remain active between visits. The live path is implemented in the protocol engine, and the page currently carries presence over it. Checkpointing to the host has been designed but is not yet connected to the page.

A world that appears to change while nobody is visiting often does not need continuous execution. If its state can be derived deterministically from elapsed time, the world can store a timestamp and calculate the current state when someone next opens it. Crops can grow and daylight can advance without a process running in the background. A world that requires continuous authority can use WorldSync's server mode. Only a world that needs its own shared state and application logic requires the separate runtime described in section 8.

The shared tier's integrity model has three planned parts, none of which is implemented yet. First, participants sign operations. Second, an access list stored in the world and signed by its owner specifies

which keys may perform which actions. Third, each participant validates incoming operations and rejects those that are not authorized. Checkpoint writes to the host are protected by the host's normal HTTP credentials, so an unauthorized participant may broadcast data but cannot make it durable. A participant can still misrepresent state to itself—for example, by rendering a deletion locally—but this is no more consequential than a browser displaying a misleading version of any web page. The model does not guarantee availability: the shared tier has no peer-voting mechanism. If an invited co-author makes a bad change, the intended response is to restore an earlier checkpoint, not to prevent the change through consensus. Today, the host's access control is the only implemented protection: it determines who may enter.

7.0 Doors and the graph

A door is a link. It contains the address of another world and may also name an entry point within that world. Target worlds can declare named entry points, much as web pages declare anchors; a door that does not specify one uses the target's default. The link is one-way, and the target world does not need to know about it or approve it. In Blender, a door is a property of an object. In a web page, it can be an ordinary anchor. Each shell decides how to present it.

A link is one-way, but a visitor usually expects to be able to return. The way back does not need to be another link; it can be browser history. Because worlds are documents at addresses, the browser's history already records the sequence of doors a visitor has followed. The shell can display the door through which the visitor arrived without writing that information into the destination world, leaving the target unchanged and unaware. A list of backlinks—worlds that link here—would be opt-in and belongs in an index. This is a legitimate role for WorldHub: it can answer who links to a world without becoming a gatekeeper or part of the process that resolves a link. The in-world return door and backlink index are not yet implemented; today, the browser's Back button provides the return path.

A web link is opened by an intentional action, while a physical doorway can be crossed simply by walking. To preserve the web's clarity and avoid accidental navigation, a door opens only after an explicit action: a click on a laptop, a tap on a phone or a controller trigger in VR—not collision alone. This is how doors work today. The design adds two protections familiar from the web. Before opening a door, the shell should show where it leads, including at least the destination host, to reduce phishing risk. Today, the page displays the door's label; showing the destination host is a further step. A shell may also begin loading a destination as a visitor approaches, making the transition feel as immediate as following a link. On phones and laptops, the browser supplies the address bar and Back button. In VR, where browser controls are not visible, the shell must provide equivalent navigation controls; that work remains outstanding.

8.0 Worlds with shared state

Some worlds need shared state that must live somewhere: objects that remain where visitors place them, environments that people edit together, or worlds connected to external systems such as game servers and sensors. WorldOS [2] is the runtime for these cases. It acts as the authority for a world's live

state, while plugins provide its application-specific behavior. Every tool that interacts with the world—the browser page, an editor or a terminal—uses the same session interface. WorldOS is deliberately not a catch-all for server-side functions. Introduction, relay and persistence are separate services that can be shared across worlds and are often unnecessary.

The plugin interface is the shared contract; the runtime is simply one way to host it. The design defines one interface with two hosts: a process host, providing strong isolation and support for plugins written in different languages, and a browser host, using workers and JavaScript. The aim is for a TypeScript plugin to behave consistently in either environment. Today, the process host runs plugins, while the browser host provides storage and palettes. The differences between the hosts are documented design choices, not accidental gaps. WorldOS is loaded only by worlds that need it, keeping it off the ordinary world-loading path.

The design calls for WorldOS to serialize the state it manages as VEML plus asset references. This would allow the live world and its durable representation to come from the same pipeline, preserving the principle that even a WorldOS-backed world is ultimately files at an address. That writer has not been built. Descriptions are intended to flow through it; assets are not. Serving meshes and textures through the runtime would recreate the expensive model this architecture is intended to avoid, charging bandwidth for every visitor. Instead, models are uploaded once to object storage, addressed by content hash and referenced by the world description.

9.0 Making worlds with the tools you already use

The system is designed to work with the tools people already use and to accommodate worlds they have already made. A world is exported from an existing editor; the editor remains the place where the creator works. Export is a one-way publishing step, and the resulting files are the world. WorldKit is the name for the family of exporters.

The first exporter is the Blender add-on [5]. After installing it and signing in once, a creator can choose Save and Publish. The add-on exports the scene, publishes it at an address and opens that address in a browser. Publishing again updates the world at the same address. The creator chooses who can enter: anyone with the link, in which case the world can also appear in WorldHub's directory, or a named group of people, who sign in once and are admitted by the host. Models created in SketchUp, Revit, Rhino or a game engine can be imported into Blender using its existing importers and published through the same workflow.

Existing environments should be brought in where practical, not rebuilt. Earlier work brought worlds from Minecraft and Cities: Skylines 2 into Web Wide Worlds. That work points to four levels of compatibility: a world can be exportable, so its scene becomes files; linkable, so it can be a destination for doors; linking, so it can contain doors to other worlds; or live, so it can stream state to visitors through WorldOS. The middle two levels are especially powerful and comparatively inexpensive: they make an existing game or environment a node in a larger graph of worlds, rather than merely a source of assets. There is an important limit. A game's logic does not automatically export. Compatibility means

that an environment can become a place people visit and connect to—not that the original game itself runs in a browser.

10.0 Hosting, and what happens if Five Squared disappears

Because a world consists of files, it can live anywhere a web page can: on a creator's domain, a static hosting service or an object-storage bucket. The shell that opens it and the host that serves it do not need to be the same. A world's address is its identity. A world hosted on its creator's domain depends on none of our infrastructure except the introduction service named in its document, and the creator can change that service. A host must allow the world to be read from another origin; static hosts generally support this through a default or configurable setting. Worlds published from Blender today are hosted by WorldHub [6] at addresses on its domain, just as pages may be hosted by any web-hosting provider. Moving one of these worlds to a creator's own domain requires downloading and uploading the files, not converting them. WorldHub also provides sign-in for private worlds and indexes worlds that creators want people to discover, regardless of where those worlds are hosted. A world using WorldHub for identity is structurally no different from one using another identity provider—or no identity provider at all. WorldHub may host a world's files or list a world hosted elsewhere, but visitors fetch the world from its own address; WorldHub is not in the serving path.

The design includes an explicit audit of what would stop working if our servers disappeared: discovery, introduction, relay, default addresses embedded in the page and any other function that resolves through a domain we own. The most dangerous dependency is one that silently places our infrastructure in a world's resolution path. The rule is therefore that, beyond hosting files as any other provider can, no world should have to route through us. Introduction and relay services should be plural and replaceable: a world can name third-party or self-hosted alternatives. Those alternatives still need to be documented and tested; today, the world's document names these services, and our services are the defaults. The design also calls for a world to carry or identify its own pinned version of WebVerse, distributed through a content network with an integrity hash and a fallback copy stored beside the world. In that arrangement, the world and its shell could be served together from one static folder, without a live dependency on our infrastructure or continued operation. Neither the full dependency audit nor the self-contained runtime distribution is implemented yet. They are included here as requirements, not presented as completed features.

The limits matter. Independence means that the content can survive and the code can be forked. It does not mean that software will maintain itself indefinitely. Without a maintainer, a web application can fall behind changes in browsers and devices. Acknowledging that limitation is essential to making a credible promise about longevity.

11.0 Security

The central promise is that people can open worlds from an address, including worlds published by people they do not know. That creates the same security responsibility that browsers have when opening web pages. Isolation must be an explicit property, supported by a documented capability model

—not an accidental consequence of how content happens to load. The model described here is the design; the engine's sandbox, which isolates world scripts from the visitor's device, is implemented. By default, a world has very limited capabilities. Access to the microphone, camera, persistent storage, other network origins and location information each requires the visitor's permission. A world cannot read another world's private state, access the surrounding shell or enumerate the visitor's browsing history.

Composition makes these boundaries more important, not less. A child document does not inherit capabilities from its parent, and a parent cannot read a child's state across the boundary. Scripts run in the engine's sandbox, separated from the visitor's device and data. Where possible, declarative behaviors, animations and triggers should be preferred because conformant shells can implement them without executing arbitrary code. Showing a door's actual destination before opening it is also part of the security model, not merely a user-interface detail: it helps protect visitors from phishing.

12.0 Applications

A space before it is built

A designer sends a client a private link. The client can walk through the kitchen on a phone before construction begins, with the designer there alongside them. There is nothing for the client to install. Buyers can explore every room in a listing from the train, so those who book a viewing already have a sense of the layout.

A scene, shared

A Blender artist sends a friend the room they modelled instead of a render of it. The friend stands in it, on a phone or a laptop, and the artist is there too.

A world for friends

A group creates a world of its own at an address it controls. Friends open it from a link, see one another and spend time together without anyone operating a dedicated world server. A door leads to a friend's world at a different address, and the two places become part of a wider network of worlds.

Existing games, extended

A world from Minecraft or Cities: Skylines 2 can become a place a visitor walks into from a link, and a node in the graph with doors in and out. Earlier work brought such worlds in; the current pieces are where opening them from a link will be built.

Education

A teacher builds an explorable environment without writing code, from models made in the tools already on the department's computers, and students open it from a link on whatever device they have. The declarative format lets non-programmers scaffold a guided walkthrough; a script is the escape hatch, not the entry fee.

Collaborative prototyping

A distributed team iterates on a layout or an interaction in one world, together, and shares each draft as a link, with no project build and no installation for each collaborator.

Digital twins and industrial systems

A world with shared state fed by sensors, robots or simulation engines runs on WorldOS, whose plugins connect it to those systems. Visitors open it from a link like any other world; the runtime holds what they see, and in the design writes it out as files like any other world.

13.0 Conclusion

Web Wide Worlds proposes a way for the web to grow beyond documents and media into places people can enter. A three-dimensional environment should be as straightforward to create, share and visit as a web page—and it should belong to the person who made it. A world is files at an address. It opens in a browser without a dedicated installation. People can experience it together without operating a permanent server for every world. It can link to other worlds, and other worlds can link to it, without asking permission. When nobody visits, it need not run. And because it is built on open formats, ordinary web infrastructure and replaceable components, it can outlast any one company, including ours.

The components are MIT-licensed. The Blender exporter is available on GitHub, and the other open components are being published as they are released. The exporter, browser shell, serverless synchronization mode, doors, and hosting and sign-in service are live. We invite creators to publish worlds and developers to replace any component while leaving the rest intact. That is not just an invitation to use the system; it is the test by which the architecture should be judged.

Works cited

[1] T. Berners-Lee, Weaving the Web, HarperCollins, 1999; and "Principles of Design," W3C Design Issues, [Online]. Available: https://www.w3.org/DesignIssues/Principles.html.

[2] Five Squared Interactive, LLC, GitHub organization, where the open pieces are published as they are released; the repositories there that already carry these names hold the previous generation, [Online]. Available: https://github.com/Five-Squared-Interactive.

[3] The Khronos Group, "glTF: Runtime 3D Asset Delivery," [Online]. Available: https://www.khronos.org/gltf/.

[4] W3C, "WebRTC: Real-Time Communication in Browsers," [Online]. Available: https://www.w3.org/TR/webrtc/.

[5] Five Squared Interactive, LLC, "WorldKit VEML Blender add-on," [Online]. Available: https://github.com/Five-Squared-Interactive/WorldKit-VEML-Blender-Plugin.

[6] Five Squared Interactive, LLC, "WorldHub," [Online]. Available: https://www.worldhub.me/.

[7] D. Z. Baker, "Web Wide Worlds: A Modular Framework for Persistent, Interoperable Digital Environments," version 1.1, June 2026, [Online]. Available: https://fivesqd.com/docs/www_whitepaper_v1.1.pdf.

What changed since version 1.1

Version 1.1 [7], of June 2026, described Web Wide Worlds as a five-layer framework: a scene viewer, a world definition layer, a networking layer with a message broker at its center, a tooling layer and a publishing layer. Readers of that version should know what this one replaces.

- **The architecture.** Five layers became three tiers: files, services and shells. A world is files at an address; services are optional; the browser supplies the shell's chrome.
- **The network. The message broker is gone from the center. WorldSync is defined without a transport, with WebRTC and WebSocket as its first bindings; a broker binding is deferred until a deployment wants one. Server-authoritative engines from the games industry are no longer planned integrations; the authoritative tier is WorldSync's own server mode.**
- **Publishing and tooling.** Version 1.1 described publishing a world as a Git repository through GitWorlds, a World Builder application, and tooling around USD and the OMI glTF extensions. None of these are part of the design now; publishing is one press from the tool you already use, and the files it produces are the world.
- **WorldOS.** Version 1.1 presented WorldOS as the server framework for deploying worlds. It is now the runtime for worlds with shared state only, and every other server-side concern is a separate, optional, often shared service.
- **Formats.** VEML 4.0 is the one world format, and it refers to glTF for every model. X3D and USD are no longer described as alternative world definitions.
- **Doors, composition and sessions** are new: links between worlds that need no consent, documents that include documents, and the session as the unit of presence and permission.
- **The perimeter.** WorldHub is described as an amenity that no world depends on, with the rule that it is never in a world's resolution path, and the paper states what happens if Five Squared disappears.
- **The promise.** The paper now says plainly that Web Wide Worlds is built so you can share the worlds you already have and make new ones with the tools you already use.