GSoC 2025: **Web Application to create and check BIM project exchange requirements** (Issue #44)

Name:	Sayan Jyoti Das
Student email:	sayanjyotidasworkmail@gmail.com
Phone:	(+91) 9612853831
IRC / Matrix username:	@sayanjdas:matrix.org
GitHub:	https://github.com/theseyan
Organization:	BRL-CAD (IfcOpenShell)
Potential Mentors:	Dion Moult

Background and motivation

I'm currently a Computer Science undergraduate student at the National Institute of Technology Agartala (NIT-A), India, where I serve as a Technical Lab Assistant at the AI/HCI Lab as well as Head of Tech at The Business Club, NITA. Previously, I've also helped organize the TEDxYouth@RonaldsayRoad event which took place in my home city, and volunteered to develop its website.

My first experience with open-source was back in 2018, when I first discovered GitHub; since then, I've developed a deep passion and respect for open source software and the value it brings to the world. As such, I fully understand the challenges related to proprietary software dominating the AEC (*Architecture, Engineering, Construction*) space and believe that IfcOpenShell is building an important open-source foundation which will drive innovation in the industry for the years to come.

Brief Summary

Information Delivery Specification (IDS) is a standard for defining information requirements in a computer interpretable form, published by buildingSMART International (a non-profit organization in the AEC space). Practically, IDS allows for compliance checking of IFC models and helps define the expectations and guidelines of what information needs to be exchanged.

IDS specifications are defined in XML, according to the standardised IDS XSD. While much easier than writing code, manually writing XML in compliance with the IDS schema can be an extremely difficult task for non-technical BIM users. Hence, a need arose for a web interface to compose IDS specifications in a user-friendly manner and abstract away the need to deal with XML source code directly.

In the previous GSoC years, there has been an effort to implement a web application to this degree. However, the current state of the web app is quite poor as the code is riddled with bugs, while the UI looks dated and lacks many authoring usability/UX features described in the IDS Developer Guide. Moreover, there have been significant strides in the **IfcOpenShell WASM** space, and it is now possible to run IfcOpenShell completely inside the browser, removing the requirement of a backend API for applications such as the IfcTester web UI.

Detailed Proposal

A new foundation on WASM

As discussed with the maintainers @Moult and @aothms, my project will be based on the IfcOpenShell WASM build which enables the following possibilities:

- Removes the requirement of a backend API, reducing server costs.
- Neither IFC nor IDS information ever leaves the browser, greatly enhancing privacy.
- Ability to run IFC compliance audits and generate reports directly in the browser.
- Makes it easy to incorporate context-aware suggestions and autocompletions as described in the IDS Developer Guide.
- Allows reusing the IDS authoring functionality already present in `ifctester`, instead of relying on a redundant implementation in Javascript (seen here).

As a proof of concept, **I've developed a playground** (code) that uses Pyodide for testing IfcOpenShell's WASM capabilities and to verify it's readiness/applicability for web app usage. With a few small patches, the experiment was a resounding success, able to perform IFC audits, author a basic IDS document, as well as run my preliminary autocompletions work.

In my project, the IfcTester web application **will be completely re-written** in a more modular approach to accommodate numerous improvements and enhance readability & maintainability of code:

- IfcTester Web App
  - **IDS module**
    An elegant Javascript wrapper built on top of the IDS authoring and validation functionality in the `ifctester.ids` and `ifctester.facet` modules. Used by the application to compose IDS XML and handle editing of existing IDS specifications, it forms the "backbone" of authoring - handling specifications, facets, cardinality, etc.
    Preliminary work: Link
  - **WASM module***
    Through extensive testing, it has been found that IfcOpenShell in Pyodide can cause UI lag and jitter, making it feel slow and unresponsive. To solve this, I'll be using web workers to run IfcOpenShell in a different thread, where it cannot affect the main (UI) thread.
    This module exposes an idiomatic API to the underlying WASM thread, making it easy and convenient for the Application module to interoperate with IfcOpenShell functionality.
  - **Application module**
    The main frontend UI for IfcTester, written in Svelte 5 and using ShadCN components.

The application directory will be a self-contained `npm` project which can drop-in replace the current web app directory.

*Additionally, It's important to note that IfcOpenShell WASM is still in its infancy; with prior testing, some issues become apparent:

- Providing invalid inputs to IfcOpenShell methods can throw C++ exceptions which crash Pyodide and lead to an unrecoverable, invalid state (also noted here).
  Planned workaround: Sanitize all inputs to IfcOpenShell methods. The WASM module will re-initialize the Pyodide environment in the case of a crash.
- The initial time to load the library is quite long (15+ seconds) which can adversely affect user experience.
  In future development*,* lazy loading should be implemented such that only the bare minimum is initially loaded, with less important parts being loaded on-demand (as suggested by @aothms).

The scope of this project can partially overlap with #99: Create a compelling interface and functionality for the IfcOpenShell WASM / pyodide module. Hence, I am fully open to collaborating with another GSoC student working on that frontier.

Better User Experience

- I'll implement **authoring suggestions & autocompletions** in the editor as described in the IDS Developer Guide for all the supported IFC schemas: IFC2X3, IFC4 and IFC4X3_ADD2.
  Thanks to the `ifcopenshell.util.schema` and `ifcopenshell.util.pset` modules, it is quite trivial to implement the suggested authoring improvements as shown in my preliminary work.

- Since an IDS file is a collection of Specifications, it can also be used as **a "library" of reusable Specifications** to import into the authoring document. The IDS module will make it possible to parse as well as mix-and-match specifications from such libraries seamlessly.

- **Integrated XML editor:** I'll build an integrated XML editor for directly editing the underlying XML source of IDS specifications, using the CodeMirror library. Ideally, I will also be adding editor schema support as well using the standardised XSD for IDS.

- Currently, there exists a *usability gap* between the IDS editor and the IFC auditing functionality. In my project, the audit report generation will be tightly knit with the editor interface such that it highlights exactly which specifications and requirements failed, for easy diagnosis or quick write-and-test loops.

- **Overhauled UI and Quality-of-Life:** I'll build a modern, responsive UI modeled similarly to Blender's interface - making Bonsai users feel right at home.
  An "Autosave" feature will keep a local-copy of the authoring IDS file saved in the browser, so users don't accidentally lose work due to unexpected exits. This will be implemented using IndexedDB and Dexie.js.

Bonsai Integration

Bonsai includes a local web UI that communicates with Blender via a Socket.IO server.
This project will utilise the existing websocket-based architecture to provide a deep integration with Bonsai's context, enabling interesting possibilities such as:

- Running audits against the in-memory IFC in Bonsai (relevant: `bonsai.tool.Ifc.get`)
- Import user-defined Property Sets for even smarter autocompletions
- And more, as new ideas emerge and time permits

On the Blender side, most of these changes can be trivially implemented in the relevant web.py and sioserver.py source code.

Documentation and Development

I'll develop extensive documentation on setting up, building and running the web application. Moreover, I'll also be writing a user guide on utilising the various features of the app to their full extent.

During the coding phase, my work will be publicly accessible at a web URL (via PythonAnywhere or similar) for feedback from the mentors and the Open Source Architecture (OSArch) community.

Additionally, I'll maintain a daily/weekly log of development activities in a dedicated page on my blog.

Future scope

Possible future development could focus on deploying and documenting the IDS module API for external usage as an independent Javascript library for IDS authoring and testing. This project should serve as an excellent starting point and example of porting components of IfcOpenShell for use on the web (relevant: Issue #99).

Deliverables

1. Build the basic UI components of the authoring interface and assemble the app layout.
2. Develop the IDS module, covering all aspects of authoring specifications.
3. Connect the IDS module to the main UI, enabling the first piece of functionality - IDS authoring.
4. Implement the WASM module and JS interface for IfcOpenShell functionality required by the app.
5. Integrate the WASM module: implement authoring autocompletions and enable IFC auditing functionality.
6. Implement Audit report generation functionality and editor highlighting.
7. Add the in-built XML code editor and implement the Autosave feature.
8. Extend Bonsai's Web UI and socket server to integrate with the IfcTester application.
9. Write comprehensive documentation and a user guide.

Project Timeline

April	-	Continue learning and exploring the codebase of IfcOpenShell and Bonsai. Discuss the project ideas further with the mentors and community. Begin drafting mock UIs for the web application.
May 8 - Jun 1	*Bonding Period*	
Jun 2 - Jun 15	**Week 1** **Week 2**	Build the basic UI components of the authoring interface and assemble the app layout.
Jun 16 - Jun 22	**Week 3**	Develop the IDS module, covering all aspects of authoring specifications.
Jun 23 - Jul 6	**Week 4** **Week 5**	Connect the IDS module to the main UI, enabling the first piece of functionality - IDS authoring.
Jul 7 - Jul 13	**Week 6**	Implement the WASM module and JS interface for IfcOpenShell functionality required by the app. Prepare for Midterm evaluation.
Jul 14 - Jul 20 (Midterm Eval)	**Week 7**	Integrate the WASM module: implement authoring autocompletions and enable IFC auditing functionality.
*Jul 21 - Jul 27	**Week 8**	Implement Audit report generation functionality and editor highlighting.
*Jul 28 - Aug 3	**Week 9**	Add the in-built XML code editor and implement the Autosave feature.
Aug 4 - Aug 10	**Week 10**	Extend Bonsai's Web UI and socket server to integrate with the IfcTester application.
Aug 11 - Aug 17	**Week 11**	Write comprehensive documentation and a user guide.
Aug 18 - Aug 24	**Week 12**	*Intentionally left empty to account for any unforeseen circumstances or delays.* Work on any remaining issues and prepare for final evaluation.
Aug 25 - Sep 1 (Final Eval)	**Week 13**	Submit final work and mentor evaluation.

** Few days of vacation planned for late July*

Past Contributions

To get familiar with the IfcOpenShell and Bonsai workflow, I made the following contributions and bugfixes:

- [Docs] Fix #4807: invisible title after searching in docs (merged)
- [IfcOpenShell] Fix #5809: Add safety checks for write permissions before writing (merged)
- [Bonsai] Add proper rounding of float attributes by model precision
- [Bonsai] Fix #6350: error when placing temporary section (merged)
- [Bonsai] Fix #6360: errors on removing temporary section cutaways (merged)
- [Bonsai] Fix #6362 (merged)
- [WebUI] Fix jQuery CDN links

Time Availability

During the summer, I have no other commitments and will be available to work full-time on the project - for at least 35-40 hours per week.

From **May 8 to May 16** (part of Community Bonding Period), I'll be having my semester End-term examinations. I will be fully available to work from **May 17** onwards.

Beginning from **July 25**, university classes will commence, however I will allocate sufficient time to continue working 35 hours a week.