

Title:

OctoPrint Integration for FreeCAD: Automated Slicer and Print Pipeline

Abstract:

FreeCAD users who design parametric models for 3D printing currently have to follow a repetitive manual workflow: exporting the model as an STL file, opening a slicer tool, importing the STL, configuring slicing parameters, generating the G-code, and finally uploading it to OctoPrint to start the print.

This project proposes a FreeCAD Python add-on that introduces a "Send to Printer" functionality, automating this entire pipeline. The add-on integrates FreeCAD's export process with external slicers such as PrusaSlicer or CuraEngine using subprocess calls, and then communicates directly with OctoPrint through its REST API to upload the generated G-code and initiate printing.

By eliminating these manual steps, the add-on streamlines the workflow from STL export to print execution, making iterative parametric design changes significantly faster and more efficient, while also reducing the need for users to leave the FreeCAD workbench. The add-on will also include a print status monitoring panel and a multi-slicer architecture — designed to support additional slicers and print servers in the future — scoped for the full 350-hour GSoc program.

Proposed Solution:

The add-on, when invoked, exports the active FreeCAD model as an STL file and calls the selected slicer tool as a background subprocess. The slicer takes the STL, applies a saved profile, and outputs a G-code file. This G-code is then sent to OctoPrint via HTTP using its REST API — specifically the file upload, job start, and status check endpoints — implemented entirely through Python's requests library. A task panel inside FreeCAD allows users to configure their OctoPrint connection URL and API key, select a slicer, and manage print profiles. A status panel displays print progress without requiring the user to leave FreeCAD.

The full pipeline connects as follows: button click → UI collects settings → STL export and slicer subprocess → OctoPrint client uploads G-code and starts print → status panel polls for progress.

The add-on will be built following existing patterns in `src/Mod/` rather than as a standalone script, integrating cleanly with FreeCAD's add-on architecture. The slicer bridge is designed as an abstraction layer, allowing additional slicers to be supported in the future without changes to core logic. If the slicer binary is not found or OctoPrint is unreachable, the add-on surfaces a clear error to the user and halts gracefully — no silent failures, no corrupted intermediate files.

Technical Approach:

The addon begins with a call to FreeCAD's built-in `Mesh.export()` function, which programmatically exports the selected object as an STL file to a temporary directory. This eliminates the need for manual export and ensures the most current version of the parametric model is always used.

The STL file is then passed to the selected external slicer via `subprocess.run()`, which executes the slicer's command-line interface in the background — invisible to the user. The slicer loads a saved profile, slices the STL, and writes the G-code output to a specified path. Because each slicer exposes a different CLI interface, this call is wrapped in an abstraction layer — a slicer-specific handler class — so that PrusaSlicer and CuraEngine can be supported without changes to the core pipeline logic.

Before uploading, the addon verifies the OctoPrint connection with a `GET /api/version` call. If successful, the G-code file is uploaded via `POST /api/files/local`, and the file is selected and queued for printing via `POST /api/files/local/{filename}` with a start command. The addon then begins polling `GET /api/job` at a regular interval to retrieve job status — completion percentage, elapsed time, and estimated remaining time — which is displayed in the FreeCAD status panel without requiring the user to open a browser.

The addon is structured following FreeCAD's standard workbench pattern, as seen in existing modules under `src/Mod/`. It defines a workbench class inheriting from `FreeCADGui.Workbench`, implementing `Initialize()` to register toolbars and menus, and `Activated()/Deactivated()` to manage workbench state. The "Send to Printer" action is implemented as a command class with `GetResources()` defining the icon and tooltip, and `Activated()` containing the pipeline logic — registered with `Gui.addCommand("OctoPrint_SendToPrinter", SendToPrinter())`. The task panel follows the pattern used in `gui_shapestrings.py`, launched via `Gui.Control.showDialog()`. User settings — OctoPrint URL, API key, and slicer path — are stored persistently using `FreeCADGui.addPreferencePage()`, consistent with how Draft and other workbenches manage preferences. The slicer subprocess and OctoPrint API calls run in a background `QThread` to prevent blocking the FreeCAD UI, with progress updates passed back to the task panel via Qt signals — ensuring the interface remains fully responsive during slicing and printing.

Problem Description:

FreeCAD users who work with parametric models for 3D printing currently have to follow a repetitive and manual workflow every time they want to print a design. This typically involves exporting the model as an STL file, opening an external slicer, importing the STL, configuring slicing parameters, generating G-code, and then uploading it to a tool like OctoPrint to start the print.

This process becomes especially inefficient in iterative design scenarios, where even small changes to a model require repeating the entire pipeline from scratch. It also introduces multiple points of friction — switching between tools, managing files manually, and reconfiguring settings — which slows down the overall workflow.

While each individual step is well-supported by existing tools, there is currently no streamlined way to connect them into a single, cohesive pipeline within FreeCAD. This gap makes the

transition from design to physical output unnecessarily time-consuming and disrupts the natural iteration cycle that parametric modeling is meant to support.

This project aims to address that gap by integrating export, slicing, and printing into a unified workflow directly within FreeCAD.

Timeline/Milestones:

Phase 1

A basic FreeCAD add-on skeleton that is properly registered and loadable, with a preferences panel where users can enter an OctoPrint URL and API key. It includes a “Test Connection” button that sends a request to `/api/version` and clearly shows whether the connection succeeds or fails.

Community Bonding (May 1 – May 7): Meet with mentor(s), finalize the technical approach and scope, set up the development environment, and review relevant parts of the FreeCAD codebase and contribution guidelines.

Week 1 (May 8-14): Study FreeCAD add-on architecture in [src/Mod/](#), set up development environment, identify all integration points.

Deliverable: dev environment running, add-on loads without errors.

Week 2 (May 15-21): Build add-on skeleton, register it in FreeCAD, create basic preferences panel UI.

Deliverable: preferences panel opens with URL and API key fields.

Week 3 (May 22-28): Implement [GET /api/version](#) connection test, wire up Test Connection button, handle error states.

Deliverable: Phase 1 complete.

Phase 2

A slicer integration layer that takes an STL file path and a selected profile, invokes PrusaSlicer using `subprocess.run()`, and outputs a valid G-code file path. The design should include a clean abstraction so that additional slicers like CuraEngine can be supported later by implementing a separate handler class.

Week 4 (May 29 – June 4) : Study PrusaSlicer CLI usage, supported flags, and profile handling. Design the slicer interface (input: STL + profile → output: G-code). Set up a basic wrapper to invoke PrusaSlicer via subprocess.

Deliverable: PrusaSlicer can be called from Python with a test STL and produces a G-code file.

Week 5 (June 5 – June 11) : Integrate STL export from FreeCAD into the pipeline. Connect export to slicer execution. Handle file paths, temporary directories, and basic configuration.

Deliverable: A selected FreeCAD object can be exported to STL and passed to PrusaSlicer to generate G-code.

Week 6 (June 12 – June 18) : Add profile selection and configuration support (e.g., layer height, presets). Improve error handling for slicer failures such as invalid profiles, missing executables, or malformed input.

Deliverable: User-selectable slicer profile successfully generates G-code with appropriate error feedback.

Week 7 (June 19 – June 25) : Refactor slicer logic into a clean abstraction layer (base class with a PrusaSlicer implementation). Define an interface that allows additional slicers such as CuraEngine to be added via separate handler classes.

Deliverable: Modular slicer interface with PrusaSlicer implemented and clear extensibility.

Week 8 (June 26 – July 2) — Buffer / Midterm : Use this week to fix bugs from Phase 2, improve stability and edge-case handling, incorporate mentor feedback, and clean up code structure.

Deliverable: Stable slicer pipeline with no major blocking issues.

Phase 3

An OctoPrint client module that uploads the generated G-code file using `/api/files/local`, starts the print job, and periodically checks `/api/job` (e.g., every 5 seconds) to track status. Any failures during upload, start, or monitoring should be surfaced to the user with clear, actionable error messages.

Week 9 (July 3 – July 9) : Study OctoPrint REST API and authentication. Implement basic client setup with API key handling and connection validation.

Deliverable: Verified connection to OctoPrint server using API key.

Week 10 (July 10 – July 16) : Implement file upload functionality using `/api/files/local`. Handle responses, file naming, and upload status.

Deliverable: G-code file successfully uploaded to OctoPrint.

Week 11 (July 17 – July 23) : Implement print start functionality and job status polling using `/api/job`. Add basic logging of print state.

Deliverable: Uploaded file can be selected, print job started, and status tracked.

Week 12 (July 24 – July 30) : Add robust error handling for network issues, failed uploads, and invalid states. Surface clear, actionable messages to the user.

Deliverable: Complete OctoPrint client with reliable upload, start, and monitoring behavior.

Phase 4

A FreeCAD task panel that guides the user through the full workflow — selecting a model, choosing a slicer profile, and triggering the pipeline in one step. The interface should also display live print progress so the user can monitor the job directly within FreeCAD.

Week 13 (July 31 – Aug 6) : Design and implement the FreeCAD task panel layout. Add controls for object selection and slicer profile selection.

Deliverable: Task panel UI renders correctly and captures required inputs.

Week 14 (Aug 7 – Aug 13) : Connect the UI to the backend pipeline (export → slice → upload → print). Ensure one-click execution works end-to-end.

Deliverable: Full workflow executes from the UI with a single action.

Week 15 (Aug 14 – Aug 20) : Add live print progress display (status updates, basic feedback). Improve user experience with loading states and clearer messaging.

Deliverable: Task panel shows real-time print progress with clear feedback.

Phase 5

End-to-end tests that validate the complete workflow from STL export to print initiation, along with documentation covering installation, configuration, and usage, so that new users can set up and use the add-on without external guidance.

Week 16 (Aug 21 – Aug 25) : Implement end-to-end tests covering the full workflow from STL export to print initiation, including common failure scenarios such as slicer errors and connection issues. Write documentation covering installation, configuration, and usage, followed by final cleanup and polish.

Deliverable: Fully tested, documented, and stable add-on ready for submission.

Risk Assessment:

One potential risk is inconsistency in slicer CLI behavior across different versions of tools like PrusaSlicer or CuraEngine. Flags, profiles, or output formats may vary, which could lead to failures or unexpected results. This will be handled by targeting a specific supported version initially and validating CLI arguments, with clear error messages if incompatibilities are detected.

Another risk is the overall complexity of integrating multiple components into a single pipeline, including FreeCAD export, slicer execution, and OctoPrint communication. Failures in any step could break the workflow or make debugging difficult. To mitigate this, the pipeline will be designed in modular stages with clear boundaries, logging, and error handling at each step to isolate and surface issues effectively.

About Me:

I am currently pursuing a Master's in Information Systems at Northeastern University, with a focus on backend systems, Python development, and building integrated engineering workflows.

During my co-op at Foundation EGI, my primary work involved CAD data pipelines using NX Open and integrating AI Copilot capabilities with CAD and PMI data. I also used FreeCAD for dataset validation and STL generation when NX licenses were unavailable, which gave me practical exposure to iterative design and analysis workflows — not just for modeling, but for downstream tasks as well. I also have experience with C++ through work on Siemens Solid Edge, which has made me comfortable navigating large, engineering-focused codebases.

This project connects directly to that background — it combines CAD workflows, Python automation, and system integration, which are areas I have already worked in professionally and want to deepen through open-source contribution. I am fully available during the GSoC coding period and can dedicate consistent, full-time effort to this project.

Previous Contributions:

I have one merged PR and one open PR in active review.

PR #28595 — Merged: Fixed the FreeCAD Dark theme overriding user tree view font size preferences. Removed a hardcoded font size value from [FreeCAD Dark.cfg](#) that was overwriting user preferences on theme load.

PR #28666 — Open: Fixing [Draft_AddToGroup](#) to prevent circular group hierarchies. Added a recursive [get_ancestors\(\)](#) function using [InList](#) traversal with a visited set to filter both the selected object and all its ancestors from the target group list.