

## Abstract

This project addresses the verbosity and constraints of *mcfunction*, Minecraft's domain-specific language (DSL) for creating content bundles known as "packs". This work presents a Java library that reduces environmental boilerplate and supplies utility functions, enabling developers to write more concise and maintainable code.

The first section outlines the structure and key components of the DSL, establishing the foundational concepts upon which the language and its file ecosystem are built. Next, I will analyze the language's limitations, including its lack of variable support, mandatory file-per-function architecture, and verbose command syntax, before presenting approaches to mitigate these issues. The following chapter focuses on the Java library developed during the internship, which I conceived to mitigate these issues. The library aims to simplify repetitive and verbose tasks, while also focusing on ease of access and user experience.

The library's architecture evolved through iterative refactoring, incorporating design patterns and abstraction layers that balance developer experience with code modularity and maintainability. By introducing features of a high-level language like Java, the library not only provides the means to eliminate the more tedious aspects of working with *mcfunction*, but also enables the declaration and use of multiple resources within a single file: a behavior not natively supported by Minecraft's compiler.

A working example will present how the library is used through the development of a project tailored to showcase its advantages compared to the traditional approach.

Finally, a comparative analysis between the Java-based implementation and the conventional method will demonstrate significant improvements: the example project required 40% fewer lines of code and consolidated 31 files into 3 manageable source files.

## Abstract 2

This project addresses the verbosity and architectural constraints of *mcfunction*, Min

ecraft's domain-specific language (DSL) for creating custom game mechanics known as "*packs*". This document presents a Java library that models *mcfunction* files and resources as Java objects, enabling developers to programmatically assemble, edit, and generate pack content with significantly improved conciseness and maintainability.

The first section outlines the structure and key components of *mcfunction*, establishing the foundational concepts of the language and its file ecosystem. The following section analyzes the language's core limitations: lack of variable support, mandatory file-per-function architecture, and verbose command syntax that hinders productivity in large-scale projects.

The central chapter presents the Java library developed to mitigate these issues. The library employs builder patterns and typed resource representations, allowing developers to construct *mcfunction* commands through fluent Java interfaces while maintaining full compatibility with Minecraft's native format. Through iterative refactoring, the architecture evolved to balance developer experience with modularity and maintainability. By leveraging Java's high-level features, the library eliminates tedious boilerplate while enabling the declaration of multiple resources within single source files, bypassing Minecraft's native requirement for separate files per function.

A practical implementation demonstrates the library's application through a complete *pack* project designed to showcase its advantages. Finally, a comparative analysis validates the approach: the example project achieved a 40% reduction in lines of code while consolidating 31 files into 3 manageable source files, demonstrating substantial improvements in both code density and project organization.