Questo progetto implementa un semplice linguaggio funzionale dotato di un sistema di permessi, controllati a runtime tramite stack inspection, ispirato a quello di Java. Per semplicità è presente un set fisso di permessi globali di accesso ai file: Read, Write, Create, Execute e Delete; rappresentati come token unici.
Sono disponibili costrutti del linguaggio per abilitare, disabilitare e controllare i permessi disponibili a run-time.
Una funzione “loader” analizza il codice e gli assegna un protection domain che comprende i permessi disponibili e informazioni sull’origine del codice. Il loader assegna i permessi in base a una policy fornita dall’utente.
A runtime ad ogni chiusura viene assegnato un protection domain in base all’origine. Quando viene eseguita un operazione su file l’algoritmo di stack inspection controlla che il permesso sia disponibile in tutti i domain sullo stack (strategia lazy), oppure termina immediatamente con risultato positivo se il domain in cima possiede un privilegio abilitato per quel permesso.
Il costrutto “Enable” abilita un permesso (privilegio) in una espressione, il privilegio viene concesso a runtime in base a una policy fornita dall’utente che tiene conto dello stato dell’esecuzione (nell’esempio ambiente e stack dei permessi). Se il permesso viene concesso, il privilegio viene ereditato dai frame successivi sullo stack.
Il costrutto “Disable” elimina un permesso dal frame corrente.

Per testare il sistema dei permessi con codice di origini diverse, vengono implementate delle funzioni per consentire la valutazione di espressioni di origine diversa.
L’interprete prende una lista di espressioni, ognuna con un nome e un origine, ed esegue la prima.
Il costrutto “Use” permette di legare ad una variabile il risultato di un’espressione ricercata per nome, la valutazione dell’espressione esterna avviene con un nuovo ambiente e un nuovo permission stack.