VS Code Macro Wrapper v0.2.6 — Consolidated (Graph + Ensures + Scan + Git)

> Single-source, ship-ready wrapper for the **VS Code** IDE. Parity with Theia wrapper. Includes: Auto-Graph toggle, Ajv input validation (via macro package), Ensures eval (runner preflight + wrapper authoritative), pre-apply secret/license scan, hash-guarded selective apply, single-commit rollback, JSONL telemetry, AST-accurate preview, and HRM adapters via `macro-graph`.

/README.md

~~~
# VS Code Macro Wrapper (v0.2.6)

## Highlights
- **Auto-Graph** (suggest | always | never) with quick-pick prompt when inputs are HRM-shaped or macro has multi-step plan.
- **Safety**: Ensures policy = block; MCP off unless allowlisted; no auto-apply; workspace confinement; pre-apply secret/license scan; hash guards; single-commit rollback.
- **Preview**: AST-accurate full-file computed after-texts; exact diffs.
- **Audit**: runId + JSONL trajectory events; node-level graph events in graph mode.

## Requirements
- Node 18+
- VS Code 1.85+
- `theia-deterministic-macros` ≥ **v0.3.9** (runner Ensures + Lint CI)
- `macro-graph` ≥ **v0.1.1**

## Install (dev)
```bash
npm i
npm run build
# package with vsce if you want a .vsix
~~~

Settings

```
{
  "macro.run.autoGraph": "suggest" // suggest | always | never
}
```

Commands

- **Macro: Run** — run macro (direct or graph, depending on toggle/prompt)

Flow

1) Run → (optional) suggest **Graph** 2) Execute → compute AST after-map 3) Ensures (wrapper) + secret/license scan 4) Diff preview → select files → **Apply** 5) Hash-guarded writes → optional Git single-commit rollback

~~~
---

## /CHANGELOG.md
```md
# Changelog

## [0.2.6] - 2025-09-03
### Added
- Auto-graph toggle (`macro.run.autoGraph`): suggest | always | never
- The same Ensures/scan/Git safeguards retained from 0.2.5

## [0.2.5] - 2025-09-01
### Added
- Ensures evaluator (wrapper authoritative) against AST afterMap
- Pre-apply secret/license scan (block/warn)
- Optional Git auto-commit for rollback
- Multi-select apply, hash guards, runId telemetry

## [0.2.4] - 2025-09-01
- Apply Selected Files and runId correlation across events

## [0.2.3] - 2025-09-01
- Ajv validation; JSONL telemetry; policy hard-stop; AST/hash-guarded apply
~~~

/package.json

```
{
  "name": "vscode-macro-wrapper",
  "displayName": "VS Code Macro Wrapper",
  "version": "0.2.6",
  "engines": { "vscode": "^1.85.0" },
  "activationEvents": ["onCommand:macro.run"],
  "main": "./dist/extension.js",
```

```
  "contributes": {
    "commands": [ { "command": "macro.run", "title": "Macro: Run" } ],
    "configuration": {
      "type": "object",
      "title": "Macro Wrapper",
      "properties": {
        "macro.run.autoGraph": {
          "type": "string",
          "enum": ["suggest","always","never"],
          "default": "suggest",
          "description": "When to use graph execution: suggest on HRM/multi-
step, always, or never."
        }
      }
    }
  },
  "scripts": { "build": "tsc -p ." },
  "dependencies": {
    "ts-morph": "^22.0.0",
    "ajv": "^8.17.1",
    "macro-graph": "^0.1.1"
  },
  "devDependencies": { "typescript": "^5.6.3", "@types/node": "^20.11.30" }
}
```

/src/extension.ts

```
/** VS Code Macro Wrapper v0.2.6 — Auto-graph selector + preview/apply pipeline.
*/
import * as vscode from 'vscode';
import { runMacro } from './macroBridge';
import { previewChangeSet } from './preview';
import { tracer } from './telemetry/tracer';
import { shouldSuggestGraph } from './autoGraph';
import { uuidv4 } from './util/uuid';
// eslint-disable-next-line @typescript-eslint/no-var-requires
const { runWithGraph } = require('macro-graph/dist/integrations/vscode');

export function activate(context: vscode.ExtensionContext) {
  context.subscriptions.push(vscode.commands.registerCommand('macro.run', async
() => {
    const runId = uuidv4();
    const name = await vscode.window.showInputBox({ prompt: 'Macro name (e.g.,
add_import or secure_endpoint)' });
```

```
    if (!name) return;
    const file = vscode.window.activeTextEditor?.document.uri.fsPath
      || (await vscode.window.showOpenDialog({ canSelectFiles: true,
canSelectFolders: false }))?.[0]?.fsPath;
    if (!file) return;
    const inputsStr = await vscode.window.showInputBox({ prompt: 'Macro inputs
(JSON)', value: JSON.stringify({ file }, null, 2) });
    if (!inputsStr) return;
    const inputs = JSON.parse(inputsStr);

    const workspaceRoot = vscode.workspace.workspaceFolders?.[0]?.uri.fsPath ||
'';
    const mode =
vscode.workspace.getConfiguration().get<'suggest'|'always'|'never'>('macro.run.autoGraph',
'suggest');

    let useGraph = false;
    if (mode === 'always') useGraph = true;
    else if (mode === 'never') useGraph = false;
    else if (mode === 'suggest' && shouldSuggestGraph(name, inputs)) {
      const choice = await vscode.window.showQuickPick(['Use Graph
(recommended)', 'Direct (simple)'], { placeHolder: 'Inputs look HRM or macro is
multi-step. Use graph?' });
      useGraph = choice?.startsWith('Use Graph') ?? false;
    }

    tracer.emit({ kind: 'macro.run.requested', when: Date.now(), data: { runId,
macro: name, file, mode: useGraph ? 'graph' : 'direct' } });
    try {
      const t0 = Date.now();
      const cs = useGraph
        ? await runWithGraph({ macroName: name, inputs, runId, emit:
(e:any)=>tracer.emit(e), workspaceRoot })
        : await runMacro({ name, inputs, workspaceRoot });
      tracer.emit({ kind: 'macro.run.succeeded', when: Date.now(), data: {
runId, macro: name, files: cs.changes.map((c:any) => c.path), durationMs:
Date.now() - t0 } });
      await previewChangeSet(cs, runId, name, inputs);
    } catch (e:any) {
      tracer.emit({ kind: 'macro.run.failed', when: Date.now(), data: { runId,
macro: name, error: String(e?.message || e) } });
      vscode.window.showErrorMessage(`Macro failed: ${e?.message || e}`);
    }
  }));
}
export function deactivate() {}
```

/src/autoGraph.ts

```
/** Auto-graph heuristics + prompt driver for VS Code wrapper. */
// eslint-disable-next-line @typescript-eslint/no-var-requires
const { findMacro } = require('theia-deterministic-macros/dist/macros');

export function looksLikeHRM(inputs: any): boolean {
  if (!inputs || typeof inputs !== 'object') return false;
  if (inputs?.meta?.source === 'HRM') return true;
  const hasFilePath = typeof inputs.filePath === 'string';
  const hasModuleAndSym = typeof inputs.module === 'string' &&
(Array.isArray(inputs.symbols) || typeof inputs.symbol === 'string');
  const hasGuards = Array.isArray(inputs.guards) || typeof inputs.guardsMap ===
'object';
  return hasFilePath && (hasModuleAndSym || hasGuards);
}

export function macroHasSteps(macroName: string): boolean {
  try { const m = findMacro(macroName); return !!m?.plan?.steps?.length; }
catch { return false; }
}

export function shouldSuggestGraph(macroName: string, inputs: any): boolean {
  return looksLikeHRM(inputs) || macroHasSteps(macroName);
}
```

/src/preview.ts

```
/**
 * v0.2.6: AST afterMap + wrapper Ensures + secret/license scan + selective
apply + Git rollback.
 */
import * as vscode from 'vscode';
import * as path from 'node:path';
import * as crypto from 'node:crypto';
import { enforceWorkspacePath } from './safety';
import { buildAfterTextMap } from './ast/applyChangeSetAst';
import { tracer } from './telemetry/tracer';
import { getMacroEnsures } from './util/validation';
import { evaluateEnsures } from './policy/ensures';
import { scanAfterMap } from './compliance/scanner';
import { commitFiles } from './git/service';
```

```
export type Change = { path: string; newText: string };
export type ChangeSet = { changes: Change[]; summary?: string; rationale?:
string; blocked?: boolean; astHints?: any };

function sha(text: string) { return
crypto.createHash('sha256').update(text).digest('hex'); }

export async function previewChangeSet(cs: ChangeSet, runId: string,
macroName?: string, macroInputs?: any) {
  tracer.emit({ kind: 'macro.preview.opened', when: Date.now(), data: { runId,
files: cs.changes.map(c => c.path) } });
  if (cs.blocked) { vscode.window.showWarningMessage('ChangeSet blocked by
policy/compliance.'); return; }
  if (cs.rationale) vscode.window.showInformationMessage(`Rationale: $
{cs.rationale}`);

  const workspaceRoot = vscode.workspace.workspaceFolders?.[0]?.uri.fsPath ||
'';
  const { afterMap } = await buildAfterTextMap(cs, async (abs) => (await
vscode.workspace.openTextDocument(abs)).getText());

  // BEFORE hashes for concurrency
  const beforeHashes: Record<string,string> = {};
  await Promise.all(Object.keys(afterMap).map(async f => {
    const doc = await vscode.workspace.openTextDocument(f);
    beforeHashes[f] = sha(doc.getText());
  }));

  // Evaluate Ensures (policy: block on failure)
  const ensuresExpr = macroName ? await getMacroEnsures(macroName) : undefined;
  if (ensuresExpr && macroInputs?.file && afterMap[macroInputs.file]) {
    const res = evaluateEnsures(ensuresExpr, afterMap[macroInputs.file],
macroInputs);
    if (!res.ok) { vscode.window.showErrorMessage(`Ensures failed: ${res.reason}
`); return; }
  }

  // Pre-apply scan (block on high severity)
  const findings = scanAfterMap(afterMap);
  const blocks = findings.filter(f => f.severity === 'block');
  if (blocks.length) { vscode.window.showErrorMessage(`Pre-apply scan blocked: $
{blocks[0].rule} in ${blocks[0].file}`); return; }
  const warns = findings.filter(f => f.severity === 'warn');
  if (warns.length) vscode.window.showWarningMessage(`$
{warns.length} warning(s) found in scan.`);

  // Multi-select files to apply
  const items = Object.keys(afterMap).map(p => ({ label: path.basename(p),
description: p }));
  const picks = await vscode.window.showQuickPick(items, { placeHolder: 'Select
files to APPLY', canPickMany: true });
  if (!picks || picks.length === 0) return; // user canceled

  // Show diff for first selection
  const first = picks[0].description!;
  const leftUri = vscode.Uri.file(first);
  const rightUri = vscode.Uri.parse(`macro-after:${first}`);
  await AfterContentProvider.registerOnce();
  await AfterContentProvider.setContent(rightUri, afterMap[first] ?? '// (no
computed after-text)');
  await vscode.commands.executeCommand('vscode.diff', leftUri, rightUri, `After
→ ${path.basename(first)}`);

  const approve = await vscode.window.showInformationMessage(`Apply $
{picks.length} selected file(s)?`, 'Apply', 'Cancel');
  if (approve !== 'Apply') { tracer.emit({ kind: 'macro.apply.canceled', when:
Date.now(), data: { runId } }); return; }
  tracer.emit({ kind: 'macro.apply.approved', when: Date.now(), data: { runId,
files: picks.map(p => p.description) } });

  // Hash-guarded full-file replace for selected files only
  const edit = new vscode.WorkspaceEdit();
  const appliedFiles: string[] = [];
  for (const pick of picks) {
    const file = pick.description!;
    enforceWorkspacePath(file, workspaceRoot);
    const uri = vscode.Uri.file(file);
    const doc = await vscode.workspace.openTextDocument(uri);
    const current = doc.getText();
    const currentHash = sha(current);
    const expected = beforeHashes[file];
    if (expected && expected !== currentHash) {
      vscode.window.showWarningMessage(`File changed since preview: ${file}. Re-
run preview.`);
      continue;
    }
    const fullRange = new vscode.Range(doc.positionAt(0),
doc.positionAt(current.length));
    edit.replace(uri, fullRange, afterMap[file]);
    appliedFiles.push(file);
  }
  await vscode.workspace.applyEdit(edit);
  await vscode.workspace.saveAll();

  // Optional Git commit for rollback
  const committed = await commitFiles(workspaceRoot, appliedFiles, `Macro apply
(runId=${runId}) — ${appliedFiles.length} file(s)`);
  if (committed) vscode.window.showInformationMessage(`Applied & committed $
{appliedFiles.length} file(s).`);
  else vscode.window.showInformationMessage(`Applied ${appliedFiles.length}
file(s). (No Git commit)`);
}

class AfterContentProvider implements vscode.TextDocumentContentProvider {
  static scheme = 'macro-after';
  private static _instance: AfterContentProvider | undefined;
  private _onDidChange = new vscode.EventEmitter<vscode.Uri>();
  private contents = new Map<string, string>();

  static async registerOnce() {
    if (!this._instance) {
      this._instance = new AfterContentProvider();
      vscode.workspace.registerTextDocumentContentProvider(this.scheme,
this._instance);
    }
  }
  static async setContent(uri: vscode.Uri, text: string) {
    this._instance!.contents.set(uri.toString(), text);
    this._instance!._onDidChange.fire(uri);
  }
  onDidChange?: vscode.Event<vscode.Uri> = this._onDidChange.event;
  provideTextDocumentContent(uri: vscode.Uri): string { return
this.contents.get(uri.toString()) ?? '// (no content)'; }
}
```

/src/policy/ensures.ts

```
/** Minimal Ensures evaluator. Currently supports: fileContainsImport(named,
from). */
export type EnsuresResult = { ok: boolean; reason?: string };

export function evaluateEnsures(expr: string | undefined, afterText: string,
inputs: any): EnsuresResult {
  if (!expr) return { ok: true };
  const m = /^(\w+)\s*\(([^)]*)\)\s*$/.exec(expr.trim());
  if (!m) return { ok: true };
  const fn = m[1];
  const argNames = m[2].split(',').map(s => s.trim()).filter(Boolean);
  const args = argNames.map(n => inputs?.[n]);
  if (fn === 'fileContainsImport') {
    const [named, from] = args as [string[]|string, string];
    const names = Array.isArray(named) ? named : (typeof named === 'string' ?
[named] : []);
    const mod = String(from || '');
    const reg = new RegExp(`import\\s+.*?from\\s+['\"]${escapeReg(mod)}['\"]`,
'm');
    if (!reg.test(afterText)) return { ok: false, reason: `Missing import from
'${mod}'` };
    for (const n of names) {
      const rn = new RegExp(`(import\\s+[^;]*\\{[^}]*\\b${escapeReg(n)}\\b[^}]*\
\}[^;]*from\\s+['\"]${escapeReg(mod)}['\"])`, 'm');
      if (!rn.test(afterText)) return { ok: false, reason:
`Missing named import '${n}' from '${mod}'` };
    }
    return { ok: true };
  }
  return { ok: true };
}
function escapeReg(s: string) { return s.replace(/[.*+?^${}()|[\]\\]/g, r => `\\
${r}`); }
```

/src/compliance/scanner.ts

```
/** Pre-apply scanners for secrets and license headers. */
export type Finding = { file: string; rule: string; severity: 'block'|'warn';
match: string };

const rules: { name: string; re: RegExp; severity: 'block'|'warn' }[] = [
  { name: 'AWS Access Key', re: /AKIA[0-9A-Z]{16}/, severity: 'block' },
  { name: 'AWS Secret Key', re: /(?<![A-Za-z0-9])[A-Za-z0-9\/+=]{40}(?![A-Za-
z0-9])/, severity: 'block' },
  { name: 'Private Key', re: /-----BEGIN (?:RSA|EC|DSA|OPENSSH|PGP) PRIVATE
KEY-----/, severity: 'block' },
  { name: 'Google API Key', re: /AIza[0-9A-Za-z\-_]{35}/, severity: 'warn' },
  { name: 'Slack Token', re: /xox[aboprs]-[0-9A-Za-z-]{10,}/, severity:
'block' }
];

const licenseRe = /\b(MIT|Apache License|GPL|BSD|MPL)\b/i;

export function scanAfterMap(afterMap: Record<string,string>): Finding[] {
  const findings: Finding[] = [];
  for (const [file, text] of Object.entries(afterMap)) {
    for (const r of rules) { const m = text.match(r.re); if (m) findings.push({
file, rule: r.name, severity: r.severity, match: m[0] }); }
    const head = text.slice(0, 500);
    if (!licenseRe.test(head)) findings.push({ file, rule: 'License header
missing/unknown', severity: 'warn', match: '' });
  }
  return findings;
}
```

/src/git/service.ts

```
/** Minimal Git helper. Fails soft if git is absent or repo not initialized. */
import { execFile } from 'node:child_process';
import { promisify } from 'node:util';
import * as path from 'node:path';
const pexec = promisify(execFile);

export async function commitFiles(workspaceRoot: string, files: string[],
message: string): Promise<boolean> {
  try { await pexec('git', ['rev-parse', '--is-inside-work-tree'], { cwd:
workspaceRoot }); } catch { return false; }
  try {
    const rel = files.map(f => path.relative(workspaceRoot, f));
    if (rel.length) await pexec('git', ['add', ...rel], { cwd: workspaceRoot });
    await pexec('git', ['commit', '-m', message], { cwd: workspaceRoot });
    return true;
  } catch { return false; }
}
```

/src/ast/applyChangeSetAst.ts

```
/** AST applier (ts-morph) for accurate preview of final file content. */
import { Project, SyntaxKind } from 'ts-morph';
export type Change = { path: string; newText: string };
export type ChangeSet = { changes: Change[]; summary?: string; rationale?:
string; blocked?: boolean };

export async function buildAfterTextMap(cs: ChangeSet, readFileText: (absPath:
string) => Promise<string>) {
  const out: Record<string, string> = {};
  for (const ch of cs.changes) {
    const original = await readFileText(ch.path).catch(() => '');
    const project = new Project({ useInMemoryFileSystem: true });
    const file = project.createSourceFile(ch.path, original, { overwrite:
true });

    if (/^\/\/\s*ensure import/i.test(ch.newText)) {
      ensureImport(file, ch.newText);
    } else if (/\/\/\s*add console\.(debug|info|warn|error)/i.test(ch.newText))
{
      addFunctionEntryLogging(file, ch.newText);
    } else {
      file.insertText(0, ch.newText);
    }
    out[ch.path] = file.getFullText();
  }
  return { afterMap: out };
}

function ensureImport(file: import('ts-morph').SourceFile, hint: string) {
  const m = /ensure import\s+([^\s]+)\s+from\s+'([^']+)'/i.exec(hint) || /\
{\s*([^}]+)\s*\}\s*from\s+'([^']+)'/i.exec(hint);
  if (!m) return;
  const names = m[1].split(',').map(s => s.trim()).filter(Boolean);
  const mod = m[2];
  const existing = file.getImportDeclarations().find(d =>
d.getModuleSpecifierValue() === mod);
  if (existing) {
    const set = new Set(existing.getNamedImports().map(i => i.getName()));
    names.forEach(n => { if (!set.has(n)) existing.addNamedImport(n); });
  } else {
    file.addImportDeclaration({ moduleSpecifier: mod, namedImports: names });
  }
}

function addFunctionEntryLogging(file: import('ts-morph').SourceFile, hint:
string) {
  const m = /console\.(debug|info|warn|error)/i.exec(hint);
  const level = (m?.[1] || 'info') as 'debug'|'info'|'warn'|'error';
  file.forEachDescendant(node => {
    if (node.getKind() === SyntaxKind.FunctionDeclaration || node.getKind() ===
SyntaxKind.MethodDeclaration || node.getKind() === SyntaxKind.ArrowFunction) {
      const body = (node as any).getBody?.();
      if (!body) return;
      const first = body.getStatements?.()[0];
      if (first && first.getText().includes('console.')) return;
      body.insertStatements(0, `console.${level}("enter:${(node as
any).getName?.() || '<fn>'}");`);
    }
  });
}
```

/src/macroBridge.ts

```
import { enforceWorkspacePath } from './safety';
// eslint-disable-next-line @typescript-eslint/no-var-requires
const { runMacroTool } = require('theia-deterministic-macros/dist/tooling/
run_macro_tool');

export async function runMacro({ name, inputs, workspaceRoot }: { name: string;
inputs: any; workspaceRoot: string }) {
  if (inputs?.file) enforceWorkspacePath(inputs.file, workspaceRoot);
  const cs = await runMacroTool({ name, inputs, workspaceRoot });
  for (const ch of cs.changes) enforceWorkspacePath(ch.path, workspaceRoot);
  return cs;
}
```

/src/util/validation.ts

```
/** Ajv accessors into macro registry for schemas + ensures text. */
import Ajv from 'ajv';
const ajv = new Ajv({ allErrors: true, strict: false });
// eslint-disable-next-line @typescript-eslint/no-var-requires
const { buildRegistry } = require('theia-deterministic-macros/dist/registry/
export');

export async function getMacroSchema(name: string): Promise<any | undefined> {
  try { const reg = buildRegistry(); return reg.macros.find((x: any) => x.name
=== name)?.inputs; } catch { return undefined; }
}
export async function getMacroEnsures(name: string): Promise<string |
undefined> {
  try { const reg = buildRegistry(); return reg.macros.find((x: any) => x.name
=== name)?.ensures; } catch { return undefined; }
}
export function validateInputsOrThrow(schema: any, inputs: any) {
  const validate = ajv.compile(schema);
  if (!validate(inputs)) {
    const msg = (validate.errors || []).map(e => `${e.instancePath || '/'} $
{e.message}`).join('; ');
```

```
    throw new Error(`Input validation failed: ${msg}`);
  }
}
```

/src/telemetry/tracer.ts

```
import * as vscode from 'vscode';
import type { TrajectoryEvent } from './types';
let channel: vscode.OutputChannel | undefined;
function chan() { return channel ?? (channel =
vscode.window.createOutputChannel('Macro Trajectory')); }
export const tracer = { emit(e: TrajectoryEvent) { try {
chan().appendLine(`[trajectory] ${JSON.stringify(e)}`); } catch {} } };
```

/src/telemetry/types.ts

```
export type TrajectoryEvent = {
  kind:
'macro.run.requested'|'macro.run.succeeded'|'macro.run.failed'|'macro.preview.opened'|'macro.appl
  when: number;
  data: Record<string, any>;
};
```

/src/safety.ts

```
import * as fs from 'node:fs';
import * as path from 'node:path';
export function enforceWorkspacePath(absPath: string, workspaceRoot: string) {
  const root = path.resolve(workspaceRoot) + path.sep;
  const real = fs.realpathSync.native(path.resolve(absPath)) + '';
  if (!real.startsWith(root)) throw new Error('Path outside workspace');
}
```

/src/util/uuid.ts

```
export function uuidv4() {
  // Simple UUID v4 generator (not crypto-strong; VS Code context is fine)
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, c => {
    const r = Math.random() * 16 | 0, v = c === 'x' ? r : (r & 0x3 | 0x8);
    return v.toString(16);
  });
}
```