WARD/EN 6000

HALDEN INDUSTRIAL GROUP / DIRECTIVE LANGUAGE

LANGUAGE REFERENCE

DSL 2 / PRINT EDITION

A directive is the whole program a drone carries into a wreck. This document is the language as the controller runs it: every keyword, every hardware call, the record `observe()` returns, the library that ships with the terminal and the limits the controller enforces. The same facts drive the Workbench's hover text and the VS Code extension.

COMPILER	`warden-dsl-2.7`
OPERATING SYSTEM	WARD/EN OS 2.4.1
APPLIES TO	The terminal's Workbench and the VS Code extension
ISSUED	15 November 2026

CONTENTS

01	A directive	03
02	Values and operators	04
03	Statements	05
04	Functions and callbacks	06
05	Enums and types	07
06	Hardware and builtins	08
07	The observe() record	11
08	The shipped library	15
09	The controller's limits	18
10	In the Workbench	19

Compiler **warden-dsl-2.7**. The course in the training berth teaches this in fifteen lessons; this document is for looking things up.

ABOUT THIS DOCUMENT

Generated from the WARD/EN language reference as published with the terminal, at compiler **warden-dsl-2.7**. The same facts drive the Workbench's hover text, the VALIDATE step before dispatch, and the VS Code extension's language server. When the language changes, this document is rebuilt from the reference; a copy that names an older compiler is superseded.

COMPILER	**warden-dsl-2.7**
OPERATING SYSTEM	WARD/EN OS 2.4.1
ISSUED	15 November 2026
CLASSIFICATION	Property of Halden Industrial Group

A DIRECTIVE

01 / FILES

MAIN.WARDEN

```
language 2;

include "lib/recovery.warden";

// Only the entry file declares the
language and on start.
on start {
    go_home(Style.quick,
ignore_contact);
}
```

A directive is one entry file plus the modules it includes. The entry file starts with **language 2;** and holds exactly one **on start** block, which runs when the drone leaves the airlock. Everything else, in any file, is a function or an enum declaration.

include	**include "path.warden";** links another file at that point. Paths are relative to the file that includes them, without **..** or a leading slash. A file is linked once however many times it is included, and a circular include is refused.
Modules	A module contains functions, enums and includes. It may not declare **language** or **on start**. The shipped modules live under **lib/**; yours can too.
Comments	**// to the end of the line** or **/* across lines */**. A docblock above a function is what the editor shows on hover.
Names	Letters, digits and underscores, not starting with a digit. One namespace for functions, enums, parameters and locals; a directive's own enum names are reserved.

VALUES AND OPERATORS

02 / DATA

TYPE	LITERAL	NOTES
number	`0`, `18`, `2.5`	Always finite. Arithmetic that produces infinity or NaN is an error.
string	`"NAV_CORE"`	Double quotes, JSON escapes, at most 4,096 characters. Item and site names are strings.
boolean	`true`, `false`	What comparisons return and what `assert` expects.
null	`null`	Nothing there: an unobserved item, an unfitted weapon, a cell without a door.
array	`[a, b]`, `array(n, v)`	Indexed from 0. Reading past the end is an error, not null. `push` and `pop` change it in place.
record	`{ x: 3, y: 4 }`	Named fields. Read with `r.x` or `r["x"]`; a missing field reads null.
site	`observe().sites.AIRLOCK`	A record with numeric `x` and `y`. The type name for annotations.
function	`ignore_contact`	Any declared function, passed by name. Two references are equal when they name the same function.

OPERATORS	HIGHEST FIRST
`!x -x`	Not, negate.
`* / %`	Multiply, divide, remainder.
`+ -`	Add, subtract. `+` joins two strings; anything else must be numbers.
`< > <= >=`	Numbers only.
`== !=`	Structural: two records or arrays with the same contents are equal, so `seen.position == target` works.
`&&`	Short-circuit and.
`||`	Short-circuit or.

> Parentheses group. Conditions treat `false`, `null`, `0` and `""` as false and everything else as true. Expressions may nest 64 deep.

STATEMENTS

03 / CONTROL

STATEMENTS

```
let i = 0;
i = i + 1;
seen.energy = 0;            // a field
route[0] = { x: 1, y: 1 }; // an index

if (result == MoveResult.ARRIVED) {
    scan();
} else {
    return false;
}

for (site in sites) {
    if (site == null) { continue; }
    if (search(site)) { break; }
}

while (observe().tick < until) {
    wait_tick();
}

return;        // returns null
```

let	Declares a local with an initial value. A local lives to the end of its block and may be reassigned.
Assignment	The target is a name, a field (**r.x**) or an index (**a[i]**). Records and arrays are shared by reference, so the change is seen everywhere the value is held.
if / else	Braces are always required. **else if** is an **else** whose block holds an **if**.
for / in	Walks an array: the body runs once per element, with the name bound to the current one. The name is a local of the function; another **for** may reuse it, a **let** may not, and the body cannot assign it. Anything but an array is a fault. Elements pushed during the loop are visited.
while	Repeats while the condition holds: the loop for counting and for waiting on the world. **break** leaves either loop, **continue** goes to the next pass.
return	Leaves the function with a value, or null when none is given. Inside **on start** it ends the directive. A directive that ends without **extract()** strands the drone.
Expression	Any call or expression followed by **;**. Hardware calls are usually written this way and their **Result** ignored; the library checks them.

> Blocks nest 64 deep and a directive holds at most 4,000 statements. Semicolons end every simple statement.

FUNCTIONS AND CALLBACKS

04 / REUSE

FUNCTIONS

```
fn search(target: site, item: string): boolean {
    let result = move_to(target, Style.quick, respond);
    if (result != MoveResult.ARRIVED) { return false; }
    scan();
    return observed_item(item) != null;
}

// A movement callback: what a contact means, step by step.
fn respond(seen: record, style: Style): Response {
    if (seen.contact == null) { return Response.CONTINUE; }
    if (seen.ammo == 0) { return Response.ABORT; }
    fire(seen.contact);
    return Response.ACTED;
}

// Planners are pure: the controller refuses hardware inside.
pure fn manhattan(a: site, b: site): number {
    return abs(a.x - b.x) + abs(a.y - b.y);
}
```

fn — `fn name(params) { ... }` at the top level of any file. Parameters are positional; a call must supply every one.

Callbacks — A function is a value. `move_to` takes a response callback, calls it `response(seen, style)` before every step and acts on its `Response`. The library ships `ignore_contact` and nothing else: the policy is yours to write.

Recursion — Allowed, 32 calls deep including the callbacks the library makes into your code.

Annotations — Optional on parameters and the return: `x: number`, `: MoveResult`. Types are `number`, `string`, `boolean`, `site`, `array`, `record`, `function`, `any` or an enum name. The editor checks literals against them before dispatch; the controller checks the value when it arrives.

pure fn — May read `observe()` and `cell()` and compute, but any hardware action inside it, or inside anything it calls, halts the directive. Route planning is pure; the shipped planner is one.

ENUMS AND TYPES

05 / VOCABULARY

```
ENUMS
enum Order { core_first, recorder_first }

fn plan(order: Order) {
    if (order == Order.core_first) { ... }
}

// A member's value is its own name as a string.
log(Style.quick == "quick"); // true
```

`enum Name { member, ... }` declares a set of values at the top level of any file. Use them as `Name.member`; a bare name or an undeclared member is a compile error. Passing a string literal where an enum is expected is refused unless it is a member, and the Workbench then shows the member to write instead.

The hardware's own enums are always in scope:

ENUM	MEMBERS	WHERE
Direction	north, east, south, west	**step**, **turn**, **open**, **cut**
Result	OK, LOW_POWER, BLOCKED, LOCKED, NOT_CLOSED, INVALID_DIRECTION, NOT_CUTTABLE, NO_RELAYS, NETWORK_FULL, RELAY_PRESENT, NO_PROBE, NO_CURRENT_CONTACT, NO_AMMO, OUT_OF_RANGE, NO_OBSERVED_ITEM, NOT_AT_AIRLOCK, NO_LINK, NO_RECIPIENT, TOO_LARGE	What every hardware action returns. Only OK means it completed.
Door	open, closed, locked	**cell(x, y).door**; null without a door
Confidence	unknown, schematic, confirmed, questionable	**cell(x, y).confidence**
Task	recover, survey	An objective's **kind**
Priority	primary, optional	An objective's **priority**
Progress	pending, satisfied, complete	An objective's **status**
Method	observe, scan	How a survey objective is satisfied

> The library adds **Style**, **MoveResult**, **Response** and **Search**, listed with it below.

HARDWARE AND BUILTINS

06 / CALLS

Reading calls take no world time. Every hardware action waits at least one tick, a quarter of a second, during which sensors, threats and the battery keep going; the drone's fit decides which actions exist at all. An action the fit lacks is refused before dispatch. Each action returns a **Result**.

READING

CALL	DOES
observe()	The drone's public state, evidence, fit and briefing as one record. See below.
cell(x, y)	Known map evidence at integer coordinates: **known**, **passable**, **door**, **confidence**, **cuttable**. Unknown cells read unknown and impassable.
type_of(value)	One of number, string, boolean, null, array, record, function.
len(value)	Length of an array or string.
receive()	The oldest waiting message as **{ from, tick, value }**, removed from the inbox, or null. Needs no hardware to read; only a transceiver ever fills it.

ACTING

CALL	DOES	NEEDS
step(direction: Direction)	Move one cell. Waits for the installed drive.	drive
turn(direction: Direction)	Face a direction without moving. One tick; a directional detector keeps sweeping.	drive
open(direction: Direction)	Open an adjacent ordinary door. Locked doors report **Result.LOCKED**.	drive
cut(direction: Direction)	Cut an adjacent maintenance barrier: eight ticks and eight energy with the barrier cutter, twice as long with the hand cutter, one loud tick with a breaching charge (two charges, **observe().cut_charges**). A bypass kit instead unlocks any locked door, security doors included, in twelve quiet ticks and leaves it closed. Structural hull cannot be cut.	cutter, charge or bypass kit

CALL	DOES	NEEDS
scan()	Confirm nearby terrain and item readings. Items are only collectable once seen. Reach by scanner: spot 2 cells for 1 energy, terrain 3 for 3, survey 5 for 6, forward lidar 7 cells in a 90° cone ahead for 2. The acoustic mapper charts 4 cells through walls and is heard; the markings reader reads and charts nothing; the budget scanner charts 4 cells as questionable; the mapping strip widens what every step charts to 3 cells. Aboard a hull that is still settling, a charting scan also reads the condition of the structure it charts into **observe().structure**; the budget scanner and the markings reader do not.	scanner
probe()	Occupancy pulse within three cells, through walls, into **observe().proximity**. Four energy and audible. Hull, machinery, salvage and bodies share the same bit. Empty does not mean safe terrain. Cells are not weapon targets. A mapping probe also charts the terrain of every cell it samples; a mass probe adds each occupied cell's **kind** (body, object or structure); a targeting probe's occupied cells may be passed to **fire** until the drone's next action.	proximity module
decoy(target: site)	Throw a noise source to a cell within five steps; it sounds for six ticks and creatures hear it like any other noise. Three charges (**observe().charges**); **NO_CHARGES**, **OUT_OF_RANGE**, **INVALID_TARGET**.	decoy launcher
smoke()	Smoke the cells round the drone: creatures cannot see through it for twenty ticks, though they can still hear. Two charges.	smoke canister
jam(direction: Direction)	Jam the adjacent door: creatures read it as locked for thirty ticks, though a breacher can force it. The drone passes as before. Three charges.	door jammer
route(target: site)	The cells to step through to **target** over the map the drone knows (closed doors allowed, locked doors and unknown cells not), or null. A fixed 2,000 instructions.	navigation co-processor
fire(contact)	Aim at recorded contact evidence with the fitted weapon. No hit confirmation; out of range wastes the shot. The shot follows the recorded bearing: another operator's drone is not a target, and a round aimed at one is spent. An area stunner needs no contact (**fire(null)** stuns everything in sight within two cells); after a targeting probe, a probed cell may be passed instead.	weapon, detector

CALL	DOES	NEEDS
`collect(item: string)`	Pick up an observed item at the occupied cell.	collector
`deploy_relay()`	Drop one of two relay charges here. Ten-cell hops must reach the airlock network. Relays persist between expeditions.	relay module
`extract()`	Leave through an authorised airlock with what is carried. Ends the expedition.	drive
`send(value, to)`	Transmit any value to crew member `to`, or to every companion when null. The drone stands and transmits one tick per `rate` bytes; a companion in range hears it at once, a linked ship holds it for the rest. Refused for free with `NO_RECIPIENT`, `NO_LINK`, `TOO_LARGE` or `LOW_POWER`.	transceiver
`wait_tick()`	Wait one tick. Sensors, threats and energy continue; the next interpreter turn has a fresh instruction allowance. Yield between expensive planning batches, outside pure functions.	nothing

CONTROL AND DATA

CALL	DOES
`log(value, ...)`	A line in the execution log: up to eight values of any kind, separated by spaces and written as `str` writes them, cut at 200 characters. The ship reads it when the uplink carries it. 400 lines per expedition.
`str(value)`	The value as text: `5` is `"5"`, `{x: 3, y: 4}` is `"{x: 3, y: 4}"`. `+` joins text only with text: `"at " + str(position)`.
`assert(condition, message)`	Halt with the message when the condition is false.
`push(array, value)` `pop(array)`	Append, or remove and return the last item, in place.
`array(length, value)`	A new array of that length filled with the value.
`abs(n)` `min(a, b)` `max(a, b)` `floor(n)`	Arithmetic.

THE OBSERVE() RECORD

07 / EVIDENCE

One call, one record, no world time. Everything in it is what the drone knows, not what is true: contacts are recorded evidence that ages, sites are the ship's chart, items are last readings. Fields for hardware the drone does not carry read null.

FIELD	HOLDS
tick	Ticks since the drone left the airlock.
position	The drone's cell, **{ x, y }**.
facing	The direction faced, with a detector fitted.
energy	Battery remaining.
controller	**{ instruction_limit, instructions_left }**: this turn's allowance and remaining instructions at the read, including the cost of calling **observe()**. A saved record does not update. Available on new equipped dispatches; historical replays keep their original record.
ammo	Rounds remaining.
cargo	Array of carried item names.
payload	**{ used, capacity }** mass of the load and what the handling gear can carry.
width, **height**	The map's size in cells.
sites	The chart's named positions by name: **sites.AIRLOCK**, **sites.POWER**, plus any recovery site the drone has found an item at. On an investigation claim the briefing withholds compartments: a withheld site is **null** here until the drone has sensed it or read of it in a record.
items	Every item reading: **{ site, position, item, observedAtTick, ... }** with its handling mass. On an investigation claim each reading also carries **marking**: the identity stamped on the object, **null** until its site has been scanned locally.
records	Investigation claims only. What the vessel's own records say, once the drone has scanned the site that holds them: **{ source, item, site, readAtTick }** per entry, a class of item and the compartment it was held in. An entry never says which object is the one the client wants, and the site it names joins **sites**.
signal	Uplink quality at this cell.
id	This drone's id.

FIELD	HOLDS
crew	Companions on the same job: { id, position, status, senior }, known only while both are on the network.
relay_charges	Relay charges left to deploy.
charges	Decoys or smoke canisters left; present only with a countermeasure fitted.
cut_charges	Breaching charges left; present only with one fitted.
seismic	[{ range, bearing }]: every body within the seismic sensor's radius, through walls, moving or still. No position, and not a weapon target.
structure	[{ position, condition, tick, inherited }]: what this drone has read of a hull that is still settling. A charting scan reads condition (load_bearing, strained, failing or collapsed) where it charts; a closed frame the drone sees is recorded however it was seen. inherited marks a reading from an earlier expedition, with its own tick. Nothing says which frame a housing carries. Absent aboard other hulls.
mail	{ pending, bytes }: what waits in the transceiver's inbox.
transceiver	{ rate, range, energy_per_tick, inbox }, or null without one: bytes per tick, direct range in cells, the cost of a transmitting tick, the inbox in bytes.
drive	{ step_ticks, energy_per_step, noise_radius }.
weapon	{ range, energy_per_shot, stun_ticks, noise_radius }, or null.
detector	{ range, arc, sweep_ticks, sweep_ms, energy_per_sweep, targeting_age_ticks } for a sweeping detector, or null.
contact	The nearest recorded contact: { position, range, bearing, range_delta, tick, sampled_at_ms, age_ms, target_valid }, or null.
contacts	Every recorded contact, same shape, with a detector fitted. A return is a body that moved, creature or machine; nothing in the evidence says which.

FIELD	HOLDS
proximity	The last **probe()**: **{ kind: "occupancy", tick, sampled_at_ms, origin, radius, cells: [{ position, occupied }] }**, or null before the first successful pulse. The footprint excludes the drone's own cell and is frozen at the sampling time, before the pulse's action tick advances the world. It includes empty cells; an old pulse is not refreshed by observing again. Historical dispatches through version 11 retain their body-contact shape.
mission	The briefing the ship sent with the drone, below.

OBSERVE().MISSION

A BRIEFING

```
{
  objectives: [
    { id: "core", label: "Recover the navigation core",
      priority: "primary", kind: "recover",
      item: "NAV_CORE", site: null,
      candidates: ["POWER", "COMMS"],
      method: null, status: "pending" },
    { id: "recorder", label: "Recover the recorder",
      priority: "optional", kind: "recover",
      item: "RECORDER", site: null,
      candidates: ["COMMS"], method: null,
      status: "pending" }
  ],
  extraction: ["AIRLOCK"],
  sites: ["AIRLOCK", "POWER", "COMMS"]
}
```

objectives In the ship's order. **kind** is a **Task**, **priority** a **Priority**, **status** a **Progress** evaluated as the drone goes.

candidates The site names that may house a recovery's item, in the ship's order. Names, not positions: walk to **observe().sites[name]**, or let **housings(item)** turn the list into positions. A suggestion, never the item's true position.

marking Investigation claims only. The marking of the wanted object. Such a recovery has an empty **candidates** list and a **null** site: find the item from **observe().records**, and lift the one whose **marking** matches. An object of the right type and the wrong marking does not complete the objective.

site, method For a survey objective: the site to confirm and whether **Method.observe** (be there) or **Method.scan** (scan there) satisfies it.

extraction	The airlocks the contract accepts. **home_site()** is the first. C/09 and C/10 can offer two: compare known routes yourself. An authorized exit does not guarantee a reachable route or radio coverage. Insertion and the wired uplink remain at AIRLOCK; AFT_AIRLOCK supplies extraction only. Aboard HIG-2570 HARROW a passage can close while the drone works, so an exit chosen earlier can become the long way round: compare again after acting.

> A plan that reads the briefing instead of naming sites flies on any wreck the client can describe, and finds nothing to do when **candidates** is empty: that search is yours to write.

THE SHIPPED LIBRARY

08 / LIB/

Four modules ship with every operation, as the last crew left them. They hold mechanism, not policy: what to do about a contact, and how to work a briefing, are yours to write. They are ordinary source, so open one in the Workbench, read it, improve it, and every directive that includes it gets the improvement. An operation's library can gain further modules as it goes; anything it gains is ordinary source too. Where a routine is deliberately naive its docblock says what it does not do.

NAVIGATION.WARDEN

DECLARES	MEANING
`enum Style { quick, cautious }`	Route preference. Cautious pays to avoid doors and doubtful cells and waits for detector sweeps.
`enum MoveResult { ARRIVED, UNRESOLVED_TARGET, ABORTED, NO_KNOWN_ROUTE, LOW_POWER }`	How a walk ended. Only ARRIVED means the drone stands on the target.
`enum Response { CONTINUE, ACTED, ABORT }`	A callback's verdict: keep going, reobserve and replan after acting, or stop.
`move_to(target, style: Style, response: function): MoveResult`	Walk to a recorded position, opening doors and replanning around obstructions, asking the response callback before each step.
`pure plan_route(start, target, style, blocked, cost_fn): array`	Dijkstra over the known map. Returns a reverse stack of cells; `pop` gives the next step. Uses the caller's instruction allowance. Cost varies with the explored map, target and callbacks; an unreachable target may require a full search. Repeated queries accumulate until a hardware action yields. Measure in the test bench and batch in the caller.
`pure route_cost(terrain: record, style: Style): number`	The planner's weight for one cell reading. Edit this to change what the drone avoids.
`pure route_priority(cost, x, y, target): number`	Visit order for candidates. Returns cost; add a heuristic for A*.
`heap_push(queue, entry)`, `heap_pop(queue): record`	The planner's priority queue.
`pure direction_to(start: site, target: site): Direction`	The cardinal direction from one cell toward another.
`pure companion_at(seen, position)`	A linked companion standing on a cell, or null.
`step_aside(position: site): number`	Up to two steps away from a point, over known passable cells. Returns the steps taken. `move_to` uses it to get clear of an obstruction and to give way to a senior companion.

DEFENCE.WARDEN

DECLARES	MEANING
`ignore_contact(seen: record, style: Style): Response`	Always CONTINUE. The only response the library ships: what to do about a contact is a policy, and policy is yours. `seen.contacts` is the evidence, `fire`, `step` and `turn` are the actions, and `Response` says what `move_to` should do next.

SENSING.WARDEN

DECLARES	MEANING
`observed_item(id: string)`	The last recorded position of an item, or null.
`has_cargo(id: string): boolean`	Whether the drone carries the item.

RECOVERY.WARDEN

DECLARES	MEANING
`survey_site(target: site, method: Method, style: Style, response: function): boolean`	Walk to a site and confirm it the way the objective asks.
`housings(item: string): array`	The briefing's candidate sites for an item, as positions.
`objectives(): array`	The briefing's objectives, pending ones first.
`primary_pending(): boolean`	Whether a primary objective is still unmet.
`reserve_for_home(): number`	Energy to reach the first extraction airlock by the shortest known route, plus a reserve. This as-found helper does not compare exits. An estimate; combat and detours cost more.
`home_site(): site`	The briefing's first extraction airlock. See the editable `public/examples/exit-choice.warden` and the Exits test-bench scenarios for comparing alternatives, and `public/examples/structure-exit.warden` with the Structure scenarios for comparing them again after the hull changes.
`go_home(style: Style, response: function): MoveResult`	Walk to the airlock and extract, with your callback.

COMMS.WARDEN

A crew's first protocol, over `send` and `receive`. A message is any value and costs its JSON bytes: one tick per 32 on the link transceiver, per 128 on the wideband. A companion within the transceiver's range with a line of sight hears a transmission the moment it ends; a linked ship holds it for everyone else and hands it over when they are linked with room in their inbox.

DECLARES	MEANING
`report(kind: string, at: site): Result`	Broadcast `{ kind, at }` to every companion.
`hear_within(ticks: number)`	Wait up to `ticks` for the next message, one `wait_tick` at a time; the message or null.
`hear_from(id: string, ticks: number)`	The same, for one sender; mail from others is read and dropped by the call.
`reply(message: record, value): Result`	Send `value` back to a message's sender.

THE CONTROLLER'S LIMITS

09 / HARDWARE

The onboard controller is a small computer. A directive that exceeds a limit halts with the reason as the last line of its log, and a halted drone does not come home. World time, battery energy and instructions are separate budgets. Pure computation consumes instructions without advancing the world. Returning from a function, reading **observe()** or **cell()**, and logging do not renew the allowance. Library calls share their caller's turn. The standard controller every chassis ships with:

The test bench shows the current or last turn's used/limit, the run's peak (including a fault) and total instructions. To spread work, call **wait_tick()** between batches in the caller. Waiting advances threats and sensors and spends energy. A single query must still fit one turn: reduce its search if it cannot. The compact controller cannot run every stock route query on larger hulls. There is no fixed instruction price for **plan_route**, and remaining allowance alone cannot predict the next query's cost.

LIMIT	VALUE	WHEN IT BITES
Instructions per turn	**500,000**	Between hardware actions, shared by all library calls and loops. The compact controller allows 150,000. A batch of route plans can exhaust it without advancing one tick.
Statements	**4,000**	Across the entry file and every included module.
Program size	**800,000 bytes**	The linked source.
Call depth	**32**	Recursion, including the library's callbacks into your functions.
Heap	**32,768 items, 4,096 containers**	Every element of every live array and record counts. Garbage is collected.
Container size	**4,096 entries**	One array or record. **array(width * height, 0)** fits any wreck.
String length	**4,096 characters**	Literals and joins.
Nesting	**64**	Blocks and expressions.
Log	**400 lines, 200 characters each**	Per expedition; later lines are dropped.

IN THE WORKBENCH

10 / EDITOR

Ctrl+Space	Completion: builtins, library, your functions, and the enum members a parameter expects.
F12	Go to definition. Right-click a symbol for its declaration and usages.
F9	Toggle a breakpoint on the current line; click the gutter for the same.
Hover	The docblock, the annotated signature, or a hardware enum's meaning. While the bench is paused, a local's value.
Cmd+Shift+F	Find across the directive and its modules.
Tab, Shift+Tab	Indent or outdent by four spaces, a line or a selection. Escape then Tab leaves the editor.

VALIDATE compiles the directive against the drone's actual fit and the wreck's chart before dispatch. TEST BENCH runs it on a prebuilt scenario with breakpoints, stepping and a live schematic; the debugger is never available on a real salvage. The same checks run in VS Code through the WARD/EN extension and its language server.