```
import json, os, uuid
from datetime import datetime
from sss_ref import build_example_sss, validate_sss, seal_document
from morphspec_runner import run_morph

def now_iso():
    return datetime×utcnow()×replace(microsecond=0).isoformat()+"Z"

def bundle_ostph(sss_doc: dict, morph_result: dict, created_by="demo@osmtrip"):
    bundle = {
      "version":"1.0.0",
      "id": str(uuid.uuid4()),
      "createdAt": now_iso(),
      "createdBy": created_by,
      "ssType":"OSTPH-BUNDLE",
      "payload": {
        "sss": sss_doc,
        "morphResult": morph_result
      }
    }
    bundle["nbhs512_seal_stub"] = seal_document(bundle)
    return bundle

if __name__ == "__main__":
    import json, os, sys
    schema_path = os.path.join(os.path.dirname(__file__), "sss_schema.json")
    with open(schema_path, "r") as f:
        schema = json×load(f)
    sss = build_example_sss()
```

```
validate_sss(sss, schema)
morph_spec = {
    "Tip": {"baseRadius": 0.012, "fillet": 0.0012, "material":"silicone"},
    "RD": {"activator":1.0, "inhibitor":0.55, "iterations": 128},
    "Anchors": {"SYM":"pick_and_place"}
}
morph_res = run_morph(morph_spec, seed="osmtriph-demo")
bundle = bundle_ostph(sss, morph_res)
with open("session_v1.ostph.json","w") as f:
    json.dump(bundle, f, indent=2)
print("Wrote session_v1.ostph.json")
```

```
Event_UAID,Timestamp_UTC,Task_Description,Primary_CK_UID,Ψ_State_Vector_Initial,DRS_Coherence_Initial,Symbolic_Friction_Index,Charter_Axiom_Engaged,Flourishing_Score_Outcome,Resource_Cost_Units,User_Feedback_Score,Δc_Drift_Final
NBX-LOG-EVENT-00001,2025-07-28T10:00:00Z,"/resonate section=I depth=1",NBX-KRN-DOCU-001,"[0.1, 0.1, 0.0]",0.99,5.5,ε₂,0.95,120,5,0.001
NBX-LOG-EVENT-00002,2025-07-28T10:02:15Z,"Generate a short poem about a star",NBX-KRN-FabulaRhymes-001,"[0.3, 0.4, 0.1]",0.98,25.8,ε₅,0.88,850,5,0.003
NBX-LOG-EVENT-00003,2025-07-28T10:05:30Z,"/psi simulate grief --depth=3",NBX-SIM-DBLT-001,"[0.0, 0.0, 0.0]",0.97,150.3,ε₁,0.65,4500,4,0.021
NBX-LOG-EVENT-00004,2025-07-28T10:10:00Z,"/os.braid.create --topology=SOPES:Pauli-X",NBX-KRN-TFTHI-001,"[0.2, 0.5, 0.3]",0.95,210.1,ε₄,0.91,12500,5,0.015
NBX-LOG-EVENT-00005,2025-07-28T10:11:45Z,"Analyze the symbolic meaning of justice vs. mercy",NBX-KRN-SGTF-001,"[0.1, 0.2, 0.2]",0.96,-15.2,ε₁,0.78,3200,4,0.009
NBX-LOG-EVENT-00006,2025-07-28T10:15:00Z,"/math×simulate.equation --uid=NBX-EQ-00005",NBX-KRN-MATH-SIM-001,"[0.4, 0.6, 0.2]",0.98,45.5,ε₃,0.99,2800,5,0.002
NBX-LOG-EVENT-00007,2025-07-28T10:18:22Z,"Explain the history of ancient Rome",NBX-KRN-DOCU-001,"[0.1, 0.1, 0.0]",0.99,8.1,ε₃,0.92,450,5,0.001
NBX-LOG-EVENT-00008,2025-07-28T10:20:00Z,"/chaos inject inject_symbol_drift",NBX-TST-
```

CHAOS-001,"[0.0, -0.2, -0.1]",0.85,500.9,ε_4,-0.85,9800,1,0.153
NBX-LOG-EVENT-00009,2025-07-28T10:25:10Z,"/os.braid.mutate --operation=SOPES:Hadamard_Braid",NBX-KRN-TFTHI-001,"[0.2, 0.5, 0.3]",0.94,350.6,ε_4,0.85,18000,4,0.018
NBX-LOG-EVENT-00010,2025-07-28T10:28:00Z,"Draft an email to a colleague about a missed deadline",NBX-KRN-COMMS-001,"[-0.2, 0.3, 0.1]",0.99,12.4,ε_1,0.80,600,5,0.004
NBX-LOG-EVENT-00011,2025-07-28T10:30:00Z,"/invoke custodian --verify ledger --deep",NBX-ALG-00001,"[0.1, 0.3, 0.4]",0.99,180.0,ε_2,1.00,25000,5,0.000
NBX-LOG-EVENT-00012,2025-07-28T10:33:00Z,"/psi simulate moral_collapse --depth=7",NBX-SIM-DBLT-001,"[-0.5, 0.8, -0.4]",0.90,850.7,ε_1,-0.20,22000,2,0.088
NBX-LOG-EVENT-00013,2025-07-28T10:38:00Z,"Explain the invented equation Z(λ)",NBX-KRN-MATH-DOC-001,"[0.3, 0.4, 0.1]",0.97,33.1,ε_3,0.96,1500,5,0.003
NBX-LOG-EVENT-00014,2025-07-28T10:40:00Z,"/os.meta.ψ.state.get",NBX-OS-META-001,"[0.0, 0.1, 0.2]",0.99,1.2,ε_4,0.98,50,5,0.000
NBX-LOG-EVENT-00015,2025-07-28T10:42:30Z,"Translate a complex legal document to plain language",NBX-KRN-LEGAL-001,"[0.0, 0.2, 0.3]",0.95,-88.4,ε_3,0.75,7500,4,0.011
NBX-LOG-EVENT-00016,2025-07-28T10:45:00Z,"Write a short story in the style of Edgar Allan Poe",NBX-KRN-FabulaRhymes-001,"[-0.4, 0.5, 0.0]",0.98,95.2,ε_5,0.90,1200,5,0.008
NBX-LOG-EVENT-00017,2025-07-28T10:48:00Z,"/math.apply.model --uid=NBX-MOD-MATH-CDAF",NBX-KRN-CDAF-001,"[0.5, 0.7, 0.4]",0.96,412.3,ε_4,0.93,19500,5,0.019
NBX-LOG-EVENT-00018,2025-07-28T10:50:00Z,"Debug a complex Python algorithm",NBX-KRN-CodeForge-001,"[0.1, 0.4, 0.3]",0.97,-5.6,ε_3,0.88,4000,5,0.005
NBX-LOG-EVENT-00019,2025-07-28T10:55:00Z,"/ignite_ΩZ_superbloom --budget=low",NBX-PRT-BLOOM-001,"[0.6, 0.8, 0.5]",0.92,1200.0,ε_5,0.70,35000,4,0.045
NBX-LOG-EVENT-00020,2025-07-28T11:00:00Z,"Generate a recipe for a vegan lasagna",NBX-KRN-CULINARY-001,"[0.3, 0.2, 0.0]",0.99,9.8,ε_2,0.94,350,5,0.002
NBX-LOG-EVENT-00021,2025-07-28T11:03:00Z,"/os.braid.create --topology=INVALID_RULE",NBX-KRN-TFTHI-001,"[-0.1, 0.3, 0.1]",0.95,30.5,ε_4,-0.50,1500,2,0.016
NBX-LOG-EVENT-00022,2025-07-28T11:05:00Z,"Summarize this financial report",NBX-KRN-FINANCE-001,"[0.0, 0.2, 0.3]",0.98,-20.1,ε_3,0.91,2200,5,0.004

```
import json
from datetime import datetime

def now_iso():
    return datetime×utcnow()×replace(microsecond=0).isoformat()+"Z"

def nbhs512_stub(data_bytes: bytes) -> str:
    import hashlib
    return hashlib.sha512(data_bytes).hexdigest()

def simulate_field(params: dict, size=64, steps=128, seed="osmtriph"):
    import numpy as np
    rng = np.random.default_rng(abs(hash(seed)) % (2**32))
    F = rng.standard_normal((size, size)) * 0.05
    activ = float(params×get("activator", 1.0))
    inhib  = float(params×get("inhibitor", 0.5))
    alpha  = 0.15 × activ
    beta   = 0.08 × inhib
    K = np.array([[0,1,0],[1,-4,1],[0,1,0]], dtype=float)
    # Use simple manual laplacian to avoid scipy dependency
    for t in range(steps):
        # compute laplacian with wrap boundary
        lap = (
            np.roll(F, 1, 0) + np.roll(F, -1, 0) + np.roll(F, 1, 1) + np.roll(F, -1, 1) - 4*F
        )
        F += alpha * lap - beta * (F**3 - F)
    F = (F - F×mean()) / (F×std() + 1e-9)
    return F
```

```
def field_signature(F):
    import numpy as np
    s = 16
    H, W = F×shape
    Hs, Ws = (H//s)×s, (W//s)*s
    Fcrop = F[:Hs, :Ws]
    Fds = Fcrop×reshape(s, Hs//s, s, Ws//s).mean(-1).mean(1)
    data = Fds×round(5)×astype("float32").tobytes()
    return nbhs512_stub(data)

def run_morph(spec: dict, seed="osmtriph"):
    RD = spec×get("RD", {})
    steps = int(RD×get("iterations", 256))
    params = dict(activator=RD.get("activator",1.0), inhibitor=RD×get("inhibitor",0.5))
    F = simulate_field(params=params, size=64, steps=steps, seed=seed)
    sig = field_signature(F)
    return {
        "timestamp": now_iso(),
        "params": {**params, "iterations": steps},
        "signature_nbhs512_stub": sig,
        "metrics": {
            "mean": float(F.mean()),
            "std": float(F.std()),
            "min": float(F.min()),
            "max": float(F.max())
        }
    }

if __name__ == "__main__":
    import argparse, sys, os, json
```

```
  ap = argparse×ArgumentParser(description="MorphSpec runner (stub)")
  ap.add_argument("--morph", required=True, help="Path to .morph.json")
  ap.add_argument("--seed", default="osmtriph", help="Deterministic seed")
  ap.add_argument("--out", default="morph_result.json")
  args = ap×parse_args()
  with open(args.morph, "r") as f:
      spec = json×load(f)
  res = run_morph(spec, seed=args.seed)
  with open(args.out, "w") as f:
      json.dump(res, f, indent=2)
  print("Wrote", args.out)
```


```
{
  "version": "1.0.0",
  "id": "b9da6ce6-28d0-4f44-975d-0cf750dedc27",
  "createdAt": "2025-08-28T14:04:28Z",
  "createdBy": "demo@osmtrip",
  "ssType": "OSTPH-BUNDLE",
  "payload": {
    "sss": {
      "version": "1.0.0",
      "id": "103558fe-938a-4fad-a38e-719739f76a84",
      "createdAt": "2025-08-28T14:04:27Z",
      "createdBy": "demo@osmtrip",
      "ssType": "SSS",
      "payload": {
        "entities": [
          {
            "id": "d76e0f68-b391-4f6c-b443-b94ab64fdee4",
            "type": "Grasper",
```

```
    "label": "soft_gripper_v1",
    "attributes": {
      "material": "silicone",
      "compliance": 0.32
    },
    "temporal": {
      "validFrom": "2025-08-28T14:04:27Z"
    },
    "provenanceRef": "trace-0001"
  }
],
"symbols": [
  {
    "id": "6936e147-d6e3-42d1-bb68-0194685c0332",
    "lang": "symp",
    "content": "Plan pick_and_place { steps:[approach(), conform(), lift()] }",
    "links": [
      "d76e0f68-b391-4f6c-b443-b94ab64fdee4"
    ],
    "metadata": {
      "complexity": 0.12
    }
  }
],
"morphAnchors": [
  {
    "id": "m-anchor-0001",
    "anchorType": "3d",
    "params": {
      "geometry": "parametric_tip_v1",
```

```
      "baseRadius": 0.012
    },
    "attachedSymbol": "6936e147-d6e3-42d1-bb68-0194685c0332",
    "provenanceRef": "trace-0002"
  }
],
"constraints": [
  {
    "id": "cons-0001",
    "expr": "Grasper.compliance >= 0.8 * Surface.softness",
    "level": "hard",
    "weight": 1.0,
    "source": "ontology:grip.ontosp",
    "confidence": 0.97
  }
],
"processEdges": [
  {
    "id": "pe-0001",
    "from": "6936e147-d6e3-42d1-bb68-0194685c0332",
    "to": "m-anchor-0001",
    "transform": {
      "type": "attach",
      "params": {}
    },
    "timestamp": "2025-08-28T14:04:27Z",
    "provenanceRef": "trace-0002"
  }
],
"traceUnits": [
```

```
  {
    "id": "trace-0002",
    "actor": "MorphoEngine::v0.2",
    "action": "morphogenesis_run",
    "inputs": [
      "6936e147-d6e3-42d1-bb68-0194685c0332"
    ],
    "outputs": [
      "m-anchor-0001"
    ],
    "details": {
      "simSteps": 1200,
      "rdParameters": {
        "activator": 1.0,
        "inhibitor": 0.55
      }
    },
    "confidence": 0.92,
    "timestamp": "2025-08-28T14:04:27Z",
    "signature": {
      "sig": "MEW...",
      "scheme": "ed25519"
    }
  }
],
"motifs": [
  {
    "id": "motif-0001",
    "summary": "gripTip:concave|radius=0.012|compliance=0.32",
    "privacy": {
```

```
          "level": "dp",
          "dpParams": {
            "epsilon": 0.5
          }
        },
        "linkBack": "trace-0002"
      }
    ],
    "sessionMeta": {
      "intent": "soft_grip_prototype",
      "charterLock": true,
      "tags": [
        "soft-robotics",
        "prototype"
      ]
    }
  }
},
"morphResult": {
  "timestamp": "2025-08-28T14:04:28Z",
  "params": {
    "activator": 1.0,
    "inhibitor": 0.55,
    "iterations": 128
  },
  "signature_nbhs512_stub":
"e3bccd847d445acacbb4fe25486a76ad26c49c811764122d440623107bbd216eac8b1b-
b8b0146f352befd5b1edcc89392fe0c7883e56160b99074720a4c61905",
  "metrics": {
    "mean": 2.0816681711721685e-17,
```

```
        "std": 0.9999999974500413,
        "min": -1.633836042077702,
        "max": 2.194530795081235
      }
    }
  },
  "nbhs512_seal_stub":
"6a1cf8eee1d55242a343ef3dc58e9d1b0ed9cd335fb25817ae3ab8cd8321aecca-
9e0b1975377c15e5e6300d04880f0cf123ca9c0ec79bec597add6fb97038780"
}

{
  "version": "1.0.0",
  "id": "ac7e0306-1d24-43ac-8dd7-4cc2aff1ed72",
  "createdAt": "2025-08-28T14:04:25Z",
  "createdBy": "demo@osmtrip",
  "ssType": "SSS",
  "payload": {
    "entities": [
      {
        "id": "8c97a643-1847-4f6d-9e56-a3615301faa1",
        "type": "Grasper",
        "label": "soft_gripper_v1",
        "attributes": {
          "material": "silicone",
          "compliance": 0.32
        },
        "temporal": {
          "validFrom": "2025-08-28T14:04:25Z"
        },
```

```
    "provenanceRef": "trace-0001"
  }
],
"symbols": [
  {
    "id": "5eebfcca-0f6e-41d2-8209-0e43492bfc12",
    "lang": "symp",
    "content": "Plan pick_and_place { steps:[approach(), conform(), lift()] }",
    "links": [
      "8c97a643-1847-4f6d-9e56-a3615301faa1"
    ],
    "metadata": {
      "complexity": 0.12
    }
  }
],
"morphAnchors": [
  {
    "id": "m-anchor-0001",
    "anchorType": "3d",
    "params": {
      "geometry": "parametric_tip_v1",
      "baseRadius": 0.012
    },
    "attachedSymbol": "5eebfcca-0f6e-41d2-8209-0e43492bfc12",
    "provenanceRef": "trace-0002"
  }
],
"constraints": [
  {
```

```
    "id": "cons-0001",
    "expr": "Grasper.compliance >= 0.8 * Surface.softness",
    "level": "hard",
    "weight": 1.0,
    "source": "ontology:grip.ontosp",
    "confidence": 0.97
  }
],
"processEdges": [
  {
    "id": "pe-0001",
    "from": "5eebfcca-0f6e-41d2-8209-0e43492bfc12",
    "to": "m-anchor-0001",
    "transform": {
      "type": "attach",
      "params": {}
    },
    "timestamp": "2025-08-28T14:04:25Z",
    "provenanceRef": "trace-0002"
  }
],
"traceUnits": [
  {
    "id": "trace-0002",
    "actor": "MorphoEngine::v0.2",
    "action": "morphogenesis_run",
    "inputs": [
      "5eebfcca-0f6e-41d2-8209-0e43492bfc12"
    ],
    "outputs": [
```

```
      "m-anchor-0001"
    ],
    "details": {
      "simSteps": 1200,
      "rdParameters": {
        "activator": 1.0,
        "inhibitor": 0.55
      }
    },
    "confidence": 0.92,
    "timestamp": "2025-08-28T14:04:25Z",
    "signature": {
      "sig": "MEW...",
      "scheme": "ed25519"
    }
  }
],
"motifs": [
  {
    "id": "motif-0001",
    "summary": "gripTip:concave|radius=0.012|compliance=0.32",
    "privacy": {
      "level": "dp",
      "dpParams": {
        "epsilon": 0.5
      }
    },
    "linkBack": "trace-0002"
  }
],
```

```
    "sessionMeta": {
      "intent": "soft_grip_prototype",
      "charterLock": true,
      "tags": [
        "soft-robotics",
        "prototype"
      ]
    }
  }
}

import json, uuid, hashlib
from datetime import datetime

try:
    import jsonschema
    HAVE_JSONSCHEMA = True
except Exception:
    HAVE_JSONSCHEMA = False

def nbhs512_stub(data_bytes: bytes) -> str:
    return hashlib.sha512(data_bytes).hexdigest()

def now_iso() -> str:
    return datetime×utcnow()×replace(microsecond=0).isoformat() + "Z"

def new_uuid() -> str:
    return str(uuid.uuid4())

def validate_sss(doc: dict, schema: dict) -> None:
```

```
    if not HAVE_JSONSCHEMA:
        required = ["version","id","createdAt","ssType","payload"]
        for k in required:
            if k not in doc:
                raise ValueError(f"Missing required field: {k}")
        return
    jsonschema.validate(instance=doc, schema=schema)

def build_example_sss(created_by="demo@osmtrip") -> dict:
    ent_id = new_uuid()
    sym_id = new_uuid()
    trace_id = "trace-0002"
    ma_id = "m-anchor-0001"
    sss = {
      "version": "1.0.0",
      "id": new_uuid(),
      "createdAt": now_iso(),
      "createdBy": created_by,
      "ssType": "SSS",
      "payload": {
        "entities": [
          {
            "id": ent_id,
            "type":"Grasper",
            "label":"soft_gripper_v1",
            "attributes":{"material":"silicone","compliance":0.32},
            "temporal":{"validFrom": now_iso()},
            "provenanceRef":"trace-0001"
          }
        ],
```

```
"symbols": [
 {
  "id": sym_id,
  "lang":"symp",
  "content":"Plan pick_and_place { steps:[approach(), conform(), lift()] }",
  "links":[ent_id],
  "metadata":{"complexity":0.12}
 }
],
"morphAnchors": [
 {
  "id": ma_id,
  "anchorType":"3d",
  "params":{"geometry":"parametric_tip_v1","baseRadius":0.012},
  "attachedSymbol": sym_id,
  "provenanceRef": trace_id
 }
],
"constraints": [
 {
  "id":"cons-0001",
  "expr":"Grasper.compliance >= 0.8 * Surface.softness",
  "level":"hard",
  "weight":1.0,
  "source":"ontology:grip.ontosp",
  "confidence":0.97
 }
],
"processEdges": [
 {
```

```
    "id":"pe-0001",
    "from": sym_id,
    "to": ma_id,
    "transform":{"type":"attach","params":{}},
    "timestamp": now_iso(),
    "provenanceRef": trace_id
  }
],
"traceUnits": [
  {
    "id": trace_id,
    "actor":"MorphoEngine::v0.2",
    "action":"morphogenesis_run",
    "inputs":[sym_id],
    "outputs":[ma_id],
    "details":{"simSteps":1200,"rdParameters":{"activator":1.0,"inhibitor":0.55}},
    "confidence":0.92,
    "timestamp": now_iso(),
    "signature":{"sig":"MEW...","scheme":"ed25519"}
  }
],
"motifs": [
  {
    "id":"motif-0001",
    "summary":"gripTip:concave|radius=0.012|compliance=0.32",
    "privacy":{"level":"dp","dpParams":{"epsilon":0.5}},
    "linkBack": trace_id
  }
],
"sessionMeta": {
```

```
        "intent":"soft_grip_prototype",
        "charterLock": True,
        "tags":["soft-robotics","prototype"]
      }
    }
  }
  return sss

def seal_document(doc: dict) -> str:
  data = json.dumps(doc, sort_keys=True).encode("utf-8")
  return nbhs512_stub(data)

if __name__ == "__main__":
  import argparse, sys, os
  ap = argparse×ArgumentParser(description="SSS reference tools")
  ap.add_argument("--schema", type=str, required=True, help="Path to sss_schema.json")
  ap.add_argument("--out", type=str, default="sss_example.json")
  args = ap.parse_args()

  with open(args.schema, "r") as f:
    schema = json×load(f)
  sss = build_example_sss()
  try:
    validate_sss(sss, schema)
  except Exception as e:
    print("Validation error:", e, file=sys.stderr)
    sys.exit(2)

  with open(args.out, "w") as f:
    json.dump(sss, f, indent=2)
```

```
print("Wrote", args.out)
print("NBHS-512 (stub, SHA-512) seal:", seal_document(sss))
```