

Appendices

from

Integrating Architecture, Processes, and AI on SAP BTP

Designing an AI-Ready Execution Layer Using SAP Signavio,
SAP LeanIX, and WalkMe



By Kumar T. Piraviperumal

Contents

A	SAP Signavio L4/L5 Attribute Model	3
B	SAP LeanIX Attribute Model	12
C	WalkMe Attribute Model	20
D	Canonical Data Model for the SAP Business Technology Platform Data Spine	27
E	Integration Application Programming Interface Specifications	43
F	AI Grounding Rules	51
G	Python Libraries and Code Assets	59
H	Reference Integration Diagrams	66

Appendix A

SAP Signavio L4/L5 Attribute Model

This appendix defines the attribute model you need to represent execution-ready business processes in SAP Signavio at L4 and L5. It also provides loadable artifacts that take a process from a Business Process Model and Notation (BPMN) sketch down to the execution layer. The scenario is the one we used throughout the book: the maintenance of a cost center in SAP Master Data Governance and its link to a position in SAP SuccessFactors. Every attribute defined here is consumed downstream by the SAP Business Technology Platform (SAP BTP) data spine ([Appendix D](#)), the integration application programming interfaces (APIs; see [Appendix E](#)), or SAP AI grounding ([Appendix F](#)).

A.1 From Business Process Model and Notation to an Execution-Ready Model

Most process content does not start out ready for execution. It starts as a business-drawn BPMN diagram with a handful of tasks, no identifiers, no system references, and a name that means something to the team that drew it and to nobody else. This section takes that sketch and breaks it down the decomposition ladder (shown in [Table A.1](#) and [Figure A.1](#)) to L4, where each activity carries the attributes the data spine needs.

Level	Scope	Running Scenario Example	Execution Relevance
L1	Value chain	Record-to-report	Navigation only; never ingested
L2	Process group	Master data management	Navigation only; never ingested
L3	End-to-end process	Manage cost center master data	Anchor for variants and ownership; ingested as context
L4	Executable process	Maintain cost center master data	Ingested; every activity mapped to applications and roles
L5	Task variant detail	Assign position to cost center (company code 1000)	Ingested where regional or legal variants change execution

Table A.1: Decomposition Ladder from Business Sketch to Execution Depth

The rule behind this ladder is straightforward. L1 and L2 exist for navigation, L3 anchors ownership and variants, and only L4 and L5 are relevant for execution. The BPMN artifacts in [Section A.6](#) show the same process twice, once at L3 intake and once at L4 execution depth, so you can see the difference represented in the XML.

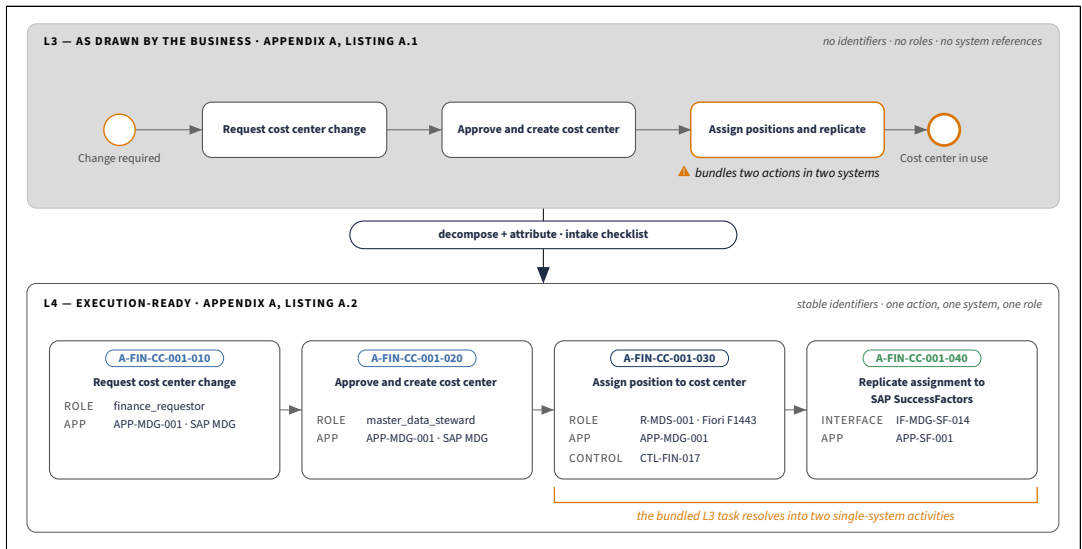


Figure A.1: The Running Scenario Is Decomposed and Attributed, from Business Sketch (L3) to Execution-Ready Model (L4)

Four questions decide whether a business BPMN can become execution-ready:

1. Does every task describe one action in one system, or does it bundle several? Split the bundled ones before you go to L4.
2. Can a stable process identifier be assigned that will survive renaming and reorganization? If not, define the identifier scheme first (see Chapter 3, Section 3.2).
3. Is there one accountable process owner role? Ownership diffusion at intake becomes ownership diffusion in the data spine.
4. Which tasks are control-relevant? Controls identified at intake are modeled as attributes rather than rediscovered at audit time.

A.2 Process-Level Attribute Model

Every L4/L5 process object (see [Figure A.2](#)) in SAP Signavio carries the attributes listed in [Table A.2](#). A missing mandatory attribute blocks publication. The validation rules in Chapter 4, Section 4.5 enforce that at export time.

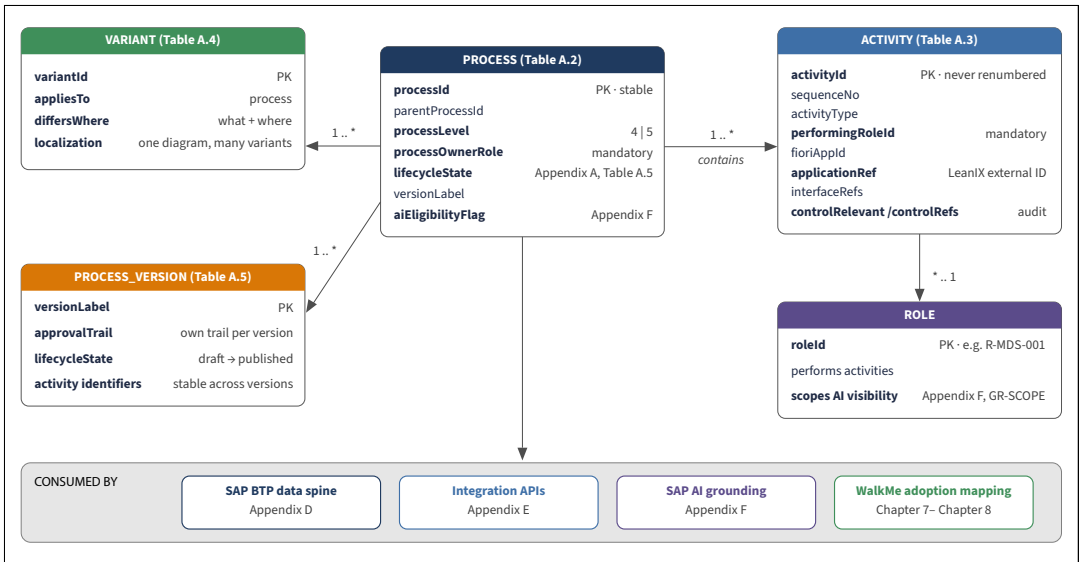


Figure A.2: The SAP Signavio L4/L5 Attribute Model, Showing Entities, Key Attributes, and the Downstream Consumers of Published Content

Attribute	Type	Mandatory?	Purpose	Consumed By
<code>processId</code>	Text (key)	Yes	Stable identifier; survives renaming, e.g., P-FIN-CC-001	Data spine PROCESS table; all APIs
<code>parentProcessId</code>	Text	Yes for L4/L5	Position in the decomposition ladder	Data spine hierarchy; navigation
<code>processLevel</code>	Number (1-5)	Yes	Execution depth; ingestion accepts 4 and 5 only	Ingestion filter (see Chapter 11, Section 11.1)
<code>processName</code>	Text	Yes	Business name; may change without breaking links	All consumers
<code>processOwnerRole</code>	Text	Yes	Accountable role; never a named person	Governance workflows (see Chapter 16)

Table A.2: Process-Level Attributes Required for Execution Readiness

Attribute	Type	Mandatory?	Purpose	Consumed By
lifecycleState	Picklist	Yes	Indicates the lifecycle status: draft, approved, published, or retired	Ingestion gate; AI grounding (see Chapter 14, Section 14.3)
versionLabel	Text	Yes	Human-readable version, e.g., v3.2	PROCESS_VERSION; change impact
signavio-DiagramId	Text	System	Native diagram reference for lineage	LINEAGE table
validFrom/ validTo	Date	Yes	Temporal validity window	Runtime resolution (see Chapter 9, Section 9.4)
controlIntent-Summary	Text	No	Aggregate control posture of the process	Audit reporting (see Chapter 19)
leanixCapabilityRef	Text	No	Link to the owning SAP LeanIX capability	Capability alignment (see Chapter 6, Section 6.3)
aiEligibility-Flag	Boolean	Yes	Whether this process may ground AI at all	Grounding filter (see Chapter 14, Section 14.1)

Table A.2: Process-Level Attributes Required for Execution Readiness (Cont.)

A.3 Activity-Level Attribute Model

Activities are where execution happens, which is why the activity attribute set is stricter than the process set. [Table A.3](#) lists the mandatory execution attributes introduced in Chapter 3, Section 3.3, along with the integration references that SAP BTP and SAP AI consume.

Attribute	Type	Mandatory?	Purpose	Consumed By
activityId	Text (key)	Yes	Stable identifier, e.g., A-FIN-CC-001-030	Data spine ACTIVITY table; WalkMe mapping

Table A.3: Activity-Level Mandatory Execution Attributes

Attribute	Type	Mandatory?	Purpose	Consumed By
sequenceNo	Number	Yes	Order within the process	Execution services
activityName	Text	Yes	One action, one system	All consumers
activityType	Picklist	Yes	Indicates the activity type: task, approval, system_step, or decision	Workflow routing (see Chapter 16)
performing-RoleId	Text	Yes	Role that executes the activity	ACTIVITY_ROLE; security mapping
fioriAppId	Text	Yes if UI exists	Execution surface, e.g., F1443	Process lookup (see Chapter 12, Section 12.1); WalkMe
transactionCode	Text	No	Uses classic transactions where relevant, e.g., KS01	Process lookup
applicationRef	Text	Yes	SAP LeanIX application external ID	ACTIVITY_APPLICATION mapping
interfaceRefs	Text (multi)	No	SAP LeanIX interface external IDs triggered	ACTIVITY_INTERFACE mapping
configuration-Reference	Text	No	IMG or central business configuration (CBC) object	Change impact (see Chapter 12, Section 12.3)
controlRelevant	Boolean	Yes	Marks control-bearing activities	ACTIVITY_CONTROL; audit
controlRefs	Text (multi)	Yes if control-relevant	Control identifiers, e.g., CTL-FIN-017	Control effectiveness (see Chapter 19, Section 19.2)
inputData-Objects	Text (multi)	No	Business objects read, e.g., Cost-Center	Context enrichment (see Chapter 12, Section 12.2)

Table A.3: Activity-Level Mandatory Execution Attributes (Cont.)

Attribute	Type	Mandatory?	Purpose	Consumed By
outputData-Objects	Text (multi)	No	Business objects written, e.g., PositionAssignment	Context enrichment; AI evidence

Table A.3: Activity-Level Mandatory Execution Attributes (Cont.)

Two of these are worth emphasizing. First, `applicationRef` carries the SAP LeanIX external ID rather than the SAP LeanIX globally unique identifier (GUID), as explained in Chapter 5, Section 5.4. Second, an activity without a `performingRoleId` cannot be published, because AI cannot reason safely about an activity that nobody owns.

A.4 Roles, Variants, and Localization Attributes

Roles and variants are modeled as attributes of the process content rather than as copies of the diagram (see [Table A.4](#)). The variant record states what differs and where, so one diagram can serve many variants and still have a single source.

Attribute	Applies To	Purpose	Example
roleId	Role	Stable role key used across systems	R-MDS-001
roleName	Role	Business name of the role	Master Data Steward
roleSource	Role	Where the role is governed	signavio, iam, s4hana_pfcg
responsibility	Activity-role link	RACI value on the assignment	A (accountable)
variantDimension	Variant	Axis along which execution differs	company_code
variantValue	Variant	Concrete value of the dimension	1000
variantDeviation	Variant	What differs from the base process	Additional legal approval step
variantLife-cycleState	Variant	Variants publish independently	published

Table A.4: Role and Variant Attributes

A.5 Lifecycle, Versioning, and Integration Reference Attributes

The lifecycle state gates every downstream consumer. Only published content is ingested, and only published content grounds AI. [Table A.5](#) defines the states and the transitions between them.

State	Meaning	May Be Ingested?	May Ground AI?	Exit Transition
draft	Being modeled; no guarantees	No	No	To approved via review
approved	Content review passed	No	No	To published via governance workflow
published	Execution-ready; identifiers frozen	Yes	Yes	To retired via decommission
retired	Superseded or withdrawn	No (history only)	No	None

Table A.5: Lifecycle States and Their Downstream Effect

Versioning follows Chapter 3, Section 3.5. A new version means a new `PROCESS_VERSION` record with its own approval trail, and activity identifiers don't change across versions. Renumbering activities between versions breaks WalkMe mappings, adoption history, and AI evidence chains, which is the single most expensive modeling mistake we warn against in this book.

A.6 SAP Signavio Loadable Artifacts

This section provides the artifacts referenced throughout the book’s walkthrough, which are ready to be imported into SAP Signavio Process Manager (**File • Import • BPMN 2.0 XML**).

[Listing A.1](#) shows the L3 intake model roughly as a business team would draw it. [Listing A.2](#) is the same process decomposed to L4, with the attribute model from [Section A.2](#) and [Section A.3](#) in extension elements. [Listing A.3](#) defines the custom attributes so your workspace will accept the imports without manual configuration first. Complete versions of all three of these files are in the book’s Kaggle repository, as described in [Appendix G](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  id="DEF-FIN-CC-L3"
  targetNamespace="http://yourcompany.com/fin/cc">
  <process id="P-FIN-CC" isExecutable="false"
```

```

    name="Manage Cost Center Master Data">
    <startEvent id="ev1" name="Change required"/>
    <task id="t1" name="Request cost center change"/>
    <task id="t2" name="Approve and create cost center"/>
    <task id="t3" name="Assign positions and replicate"/>
    <endEvent id="ev2" name="Cost center in use"/>
    <sequenceFlow id="f1" sourceRef="ev1" targetRef="t1"/>
    <sequenceFlow id="f2" sourceRef="t1" targetRef="t2"/>
    <sequenceFlow id="f3" sourceRef="t2" targetRef="t3"/>
    <sequenceFlow id="f4" sourceRef="t3" targetRef="ev2"/>
  </process>
</definitions>

```

Listing A.1: L3 Intake Model as Drawn by the Business (BPMN 2.0 XML)

Notice what's missing. There are no stable identifiers, no roles, and no system references. Additionally, one task, Assign positions and replicate, bundles two actions across two systems. The L4 decomposition resolves this bundling.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:spine="http://yourcompany.com/spine/attributes"
  id="DEF-FIN-CC-001" targetNamespace=
    "http://yourcompany.com/fin/cc">
  <process id="P-FIN-CC-001" isExecutable="false"
    name="Maintain Cost Center Master Data">
    <extensionElements>
      <spine:attr name="processId" value="P-FIN-CC-001"/>
      <spine:attr name="parentProcessId" value="P-FIN-CC"/>
      <spine:attr name="processLevel" value="4"/>
      <spine:attr name="processOwnerRole"
        value="master_data_steward"/>
      <spine:attr name="lifecycleState" value="published"/>
      <spine:attr name="versionLabel" value="v3.2"/>
      <spine:attr name="aiEligibilityFlag" value="true"/>
    </extensionElements>
    <task id="t030" name="Assign position to cost center">
      <extensionElements>
        <spine:attr name="activityId"
          value="A-FIN-CC-001-030"/>
        <spine:attr name="sequenceNo" value="30"/>
        <spine:attr name="activityType" value="task"/>
        <spine:attr name="performingRoleId"
          value="R-MDS-001"/>
      </extensionElements>
    </task>
  </process>
</definitions>

```

```

    <spine:attr name="fioriAppId" value="F1443"/>
    <spine:attr name="applicationRef"
      value="APP-MDG-001"/>
    <spine:attr name="interfaceRefs"
      value="IF-MDG-SF-014"/>
    <spine:attr name="controlRelevant" value="true"/>
    <spine:attr name="controlRefs"
      value="CTL-FIN-017"/>
    <spine:attr name="outputDataObjects"
      value="PositionAssignment"/>
  </extensionElements>
</task>
<!-- remaining tasks follow the same pattern -->
</process>
</definitions>

```

Listing A.2: L4 Execution Model with the Attribute Set (BPMN 2.0 XML, Abridged)

```

{
  "customAttributes": [
    { "name": "processId",      "type": "text",
      "appliesTo": "process",  "mandatory": true },
    { "name": "processLevel",  "type": "number",
      "appliesTo": "process",  "mandatory": true },
    { "name": "lifecycleState", "type": "picklist",
      "values": ["draft", "approved", "published", "retired"],
      "appliesTo": "process",  "mandatory": true },
    { "name": "activityId",    "type": "text",
      "appliesTo": "task",     "mandatory": true },
    { "name": "performingRoleId", "type": "text",
      "appliesTo": "task",     "mandatory": true },
    { "name": "fioriAppId",    "type": "text",
      "appliesTo": "task",     "mandatory": false },
    { "name": "applicationRef", "type": "text",
      "appliesTo": "task",     "mandatory": true },
    { "name": "controlRelevant", "type": "boolean",
      "appliesTo": "task",     "mandatory": true }
  ]
}

```

Listing A.3: Custom Attribute Definitions for the Workspace (JSON)

The import order matters, so you must load the attribute definitions first, then the L3 model, and then the L4 model. After that, the export pipeline from Chapter 4 can extract the L4 model and deliver it to the ingestion API described in [Appendix E](#) without further preparation.

Appendix B

SAP LeanIX Attribute Model

This appendix documents the SAP LeanIX object and attribute model required to support integration and impact analysis. It covers the factsheet types, their attributes, the identifier and lifecycle rules, and how all of it aligns to L4/L5 process semantics. It closes with loadable factsheet artifacts for the running scenario, so you can reproduce the SAP LeanIX side of the cost center and position example in your own workspace.

B.1 Factsheet Types Required for Integration

Only three factsheet types feed the data spine (see [Table B.1](#) and [Figure B.1](#)). The rest of what SAP LeanIX offers is valuable for portfolio management, but it is out of scope for execution, following the semantic boundary drawn in Chapter 5, Section 5.5.

Factsheet Type	Purpose in the Data Spine	Data Spine Entity	Chapter Reference
Application	What executes; ownership, hosting, life-cycle	APPLICATION	Chapter 5, Section 5.2 and Chapter 9, Section 9.3
Interface	How systems exchange data; direction and protocol	INTERFACE	Chapter 5, Section 5.3 and Chapter 9, Section 9.3
Business capability	Domain alignment for processes and applications	CAPABILITY	Chapter 6, Section 6.3

Table B.1: Factsheet Types Ingested Into the Data Spine

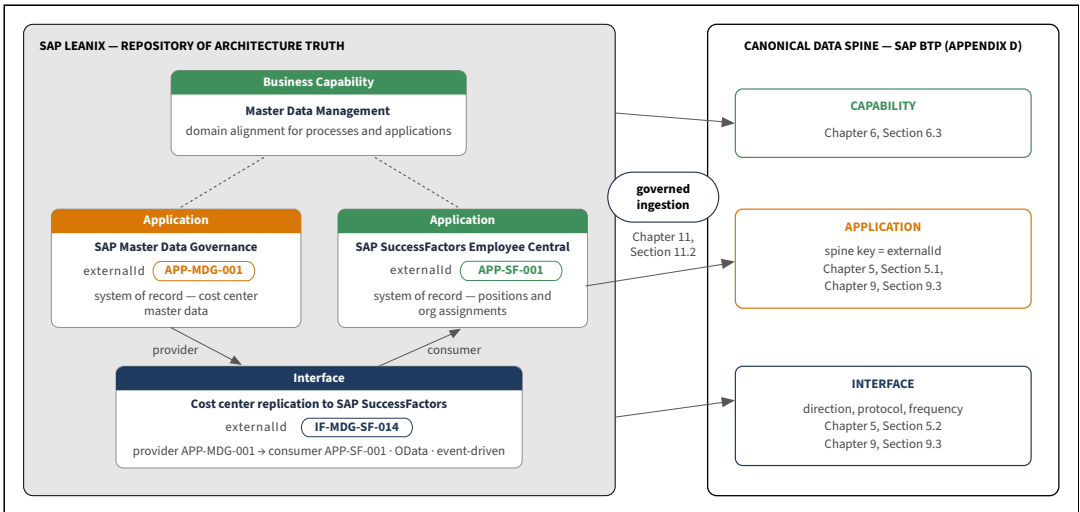


Figure B.1: SAP LeanIX Factsheets and Their Alignment to the Canonical Data Spine—the Governed externalId, Never the GUID, Is the Data Spine Key

B.2 Application Factsheet Attribute Model

Table B.2 defines the application attributes the data spine requires. If a mandatory attribute is missing, it will block ingestion, and the validation runs inside the SAP LeanIX-to-SAP BTP flow covered in Chapter 11, Section 11.2.

Attribute	Type	Mandatory?	Purpose	Consumed By
externalId	Text (key)	Yes	Stable data spine key, e.g., APP-MDG-001; never the SAP LeanIX GUID	Data spine APPLICATION table; all mappings
name	Text	Yes	Application name, e.g., SAP Master Data Governance	All consumers
description	Text	No	Business purpose in one paragraph	Context enrichment
lifecycle	Phases	Yes	Indicates the lifecycle phase: plan, phaseIn, active, phaseOut, or end-ofLife with dates	Change impact; ingestion gate

Table B.2: Application Factsheet Attributes

Attribute	Type	Mandatory?	Purpose	Consumed By
applicationCategory	Picklist	Yes	Indicates the application category: core_erp, satellite, saas, or custom_btp	Impact weighting (see Chapter 12, Section 12.3)
technicalOwner	Text	Yes	Role accountable for operation	Governance workflows
businessOwner	Text	Yes	Role accountable for function	Governance workflows
hosting	Picklist	Yes	Indicates the hosting type: on_premise, btp, hyperscaler, or vendor_saas	Security boundaries (see Chapter 2, Section 2.9)
businessCriticality	Picklist	No	Indicates the criticality: missionCritical to administrative	Impact prioritization
dataClassification	Picklist	Yes	Indicates the highest data class processed	AI eligibility (see Chapter 14, Section 14.1)
relApplicationTo-Process	Relation	Derived	Maintained via the data spine mapping, not manually	ACTIVITY_APPLICATION

Table B.2: Application Factsheet Attributes (Cont.)

The `externalId` rule from Chapter 5, Section 5.4 is worth restating here: The data spine key is the governed external identifier. It should be assigned once and never recycled. An SAP LeanIX GUID can change when a factsheet is archived and recreated, but the external ID must not.

B.3 Interface Factsheet Attribute Model

Table B.3 defines the interface attributes the data spine requires. Two points are easy to miss. An interface carries its own lifecycle, independent of the applications at either end, which is how change impact differentiates a retired interface from a retired application.

Additionally, the `providerApplication` and `consumerApplication` attributes both reference governed external IDs rather than GUIDs, so a dependency chain survives the recreation of either factsheet.

Attribute	Type	Mandatory?	Purpose	Consumed By
<code>externalId</code>	Text (key)	Yes	Stable data spine key, e.g., IF-MDG-SF-014	Data spine INTERFACE table
<code>name</code>	Text	Yes	What the interface carries, stated plainly	All consumers
<code>providerApplication</code>	Relation	Yes	Application external ID providing the data	Dependency chains
<code>consumerApplication</code>	Relation	Yes	Application external ID consuming the data	Dependency chains
<code>protocol</code>	Picklist	Yes	Indicates the protocol: odata, rest, soap, idoc, event, or file	Integration design (see Chapter 11)
<code>frequency</code>	Picklist	Yes	Indicates the frequency: real_time, event, or batch	Execution service design
<code>dataObject</code>	Text	Yes	Business object carried, e.g., CostCenter	Context enrichment; AI evidence
<code>lifecycle</code>	Phases	Yes	Interface lifecycle independent of applications	Change impact
<code>middleware</code>	Text	No	Transport, e.g., SAP Integration Suite	Operations (see Chapter 18)
<code>dataClassification</code>	Picklist	Yes	Indicates the highest data class in transit	Security boundaries

Table B.3: Interface Factsheet Attributes

B.4 Lifecycle, Identifier, and Governance Fields

In SAP LeanIX, the lifecycle is a set of dated phases rather than a single state. The data spine reads whichever phase is valid at ingestion time and translates it into the `lifecycle_state` column covered in [Appendix D. Table B.4](#) defines that translation and what it does downstream.

SAP LeanIX Phase	Data Spine lifecycle_state	Ingested?	Effect on Mappings
plan	plan	Yes	Mappings allowed as proposed only
phaseIn	phase_in	Yes	Mappings may be confirmed; flagged in impact results
active	active	Yes	Full participation in execution and AI evidence
phaseOut	phase_out	Yes	Impact results show a warning; new mappings blocked
endOfLife	end_of_life	History only	Mappings retired; historical lineage preserved

Table B.4: Lifecycle Phase Translation and Downstream Effect

Governance fields complete the model. Every factsheet carries a quality seal state, a last-review date, and a subscription naming the accountable role. A factsheet with a broken quality seal still ingests, but the data spine marks its mappings as unverified, and change impact results say as much.

B.5 Alignment with L4/L5 Process Semantics

The most common SAP LeanIX modeling failure in integrated landscapes is semantic overlap, with process knowledge creeping into factsheets and architecture knowledge creeping into process models. [Table B.5](#) draws this boundary item by item.

Knowledge	Belongs In	Never In	Why?
What an activity does and who performs it	SAP Signavio	SAP LeanIX	Process semantics have one system of record
Which application an activity executes in	Data spine mapping	Either repository as free text	Mappings are governed data, not documentation
Application ownership, hosting, and lifecycle	SAP LeanIX	SAP Signavio	Enterprise structure has one system of record
Interface protocol and dependency chains	SAP LeanIX	SAP Signavio annotations	Dependencies must survive process redesign
Control intent of an activity	SAP Signavio	SAP LeanIX	Controls attach to process steps, not systems
Configuration references	Data spine mapping	Factsheet descriptions	Change impact needs them as data

Table B.5: Semantic Boundary Between Repositories and the Data Spine

B.6 SAP LeanIX Loadable Artifacts

The artifacts in this section help you reproduce the running scenario in your own SAP LeanIX workspace. [Listing B.1](#) and [Listing B.2](#) are factsheets for the two applications, [Listing B.3](#) is the connecting interface, and [Listing B.4](#) shows the GraphQL mutation pattern used by the ingestion pipeline covered in [Appendix E](#). Load them via the SAP LeanIX Excel import or the GraphQL API. Complete files are available in the Kaggle repository, described in [Appendix G](#).

```
{
  "factSheetType": "Application",
  "externalId": "APP-MDG-001",
  "name": "SAP Master Data Governance",
  "description": "System of record for cost center
    master data and its governed change requests.",
  "lifecycle": {
    "phases": [
      { "phase": "active", "startDate": "2023-01-01" }
    ]
  }
}
```

```

    },
    "applicationCategory": "core_erp",
    "technicalOwner": "erp_platform_lead",
    "businessOwner": "master_data_steward",
    "hosting": "on_premise",
    "dataClassification": "confidential"
  }
}

```

Listing B.1: Application Factsheet: SAP Master Data Governance (JSON)

```

{
  "factSheetType": "Application",
  "externalId": "APP-SF-001",
  "name": "SAP SuccessFactors Employee Central",
  "description": "System of record for positions
    and organizational assignments.",
  "lifecycle": {
    "phases": [
      { "phase": "active", "startDate": "2022-06-01" }
    ]
  },
  "applicationCategory": "saas",
  "technicalOwner": "hr_platform_lead",
  "businessOwner": "hr_operations_lead",
  "hosting": "vendor_saas",
  "dataClassification": "confidential"
}

```

Listing B.2: Application Factsheet: SAP SuccessFactors Employee Central (JSON)

```

{
  "factSheetType": "Interface",
  "externalId": "IF-MDG-SF-014",
  "name": "Cost center replication to SuccessFactors",
  "providerApplication": "APP-MDG-001",
  "consumerApplication": "APP-SF-001",
  "protocol": "odata",
  "frequency": "event",
  "dataObject": "CostCenter",
  "middleware": "SAP Integration Suite",
  "lifecycle": {
    "phases": [
      { "phase": "active", "startDate": "2023-03-01" }
    ]
  }
}

```

```

    },
    "dataClassification": "confidential"
  }
}

```

Listing B.3: Interface Factsheet: Cost Center Replication (JSON)

```

mutation {
  createFactSheet(input: {
    name: "SAP Master Data Governance",
    type: Application,
    permittedReadACL: [],
    permittedWriteACL: []
  },
  patches: [{
    op: replace,
    path: "/externalId",
    value: "{\"externalId\": \"APP-MDG-001\"}"
  }]) {
    factSheet { id name type }
  }
}

```

Listing B.4: GraphQL Mutation Creating a Factsheet with a Governed External ID

With these factsheets in place, the SAP LeanIX-to-SAP BTP ingestion flow from Chapter 11, Section 11.2 populates the APPLICATION and INTERFACE tables from [Appendix D](#). The mapping layer can then link activity A-FIN-CC-001-030 to APP-MDG-001 and IF-MDG-SF-014, exactly as the worked example in Chapter 9 describes.

Appendix C

WalkMe Attribute Model

This appendix is the WalkMe counterpart of [Appendix A](#) and [Appendix B](#). [Appendix A](#) defines the attribute model that makes SAP Signavio process models execution-ready, [Appendix B](#) does the same for SAP LeanIX factsheets, and this appendix defines the attributes that make WalkMe guidance and telemetry process-aware. It is the reference for Chapter 7 and Chapter 8, and it feeds three tables of the canonical data spine—WALKME_CONTENT, ADOPTION_EVENT, and ADOPTION_SIGNAL—whose full definitions appear in the [Appendix D](#) model (refer to [Figure D.8](#)).

One point before we get to the tables. WalkMe, as delivered, measures interactions with applications. The architecture in this book asks more of it than that, namely telemetry that joins to the process hierarchy, to approved versions, to controls, and to AI evidence. No amount of clever analysis downstream will produce that join. It comes from disciplined tagging upstream, where every guidance item carries its process anchors from the day it is authored and every event inherits them. An untagged item produces app analytics, but a tagged item produces process *intelligence*.

C.1 The Adoption Layer at a Glance

[Figure C.1](#) shows the full path of adoption data from authoring to consumption. Tagged content is authored in the WalkMe Editor, plays on the SAP Fiori launchpad while users work, and produces raw events in WalkMe Insights. During extraction—and only then—the privacy gate covered in Chapter 8, Section 8.1 reduces every user reference to a role and pseudonymizes the session. From there, the governed ingestion endpoint described in [Appendix E](#) carries the events into the data spine, where they aggregate into adoption signals. Dashboards, the impact service, and AI all consume those signals under the access rules covered in Chapter 12 and Chapter 14.

The running scenario shows up here as well. Smart Walk-Thru WM-SWT-00042 guides the activity **Assign position to cost center** (A-FIN-CC-001-030) on SAP Fiori app F1443 (Manage Cost Centers), and its friction signal of 0.31 is the number that later surfaces in the activity context of [Appendix E](#) and in the AI evidence sets of [Appendix G](#).

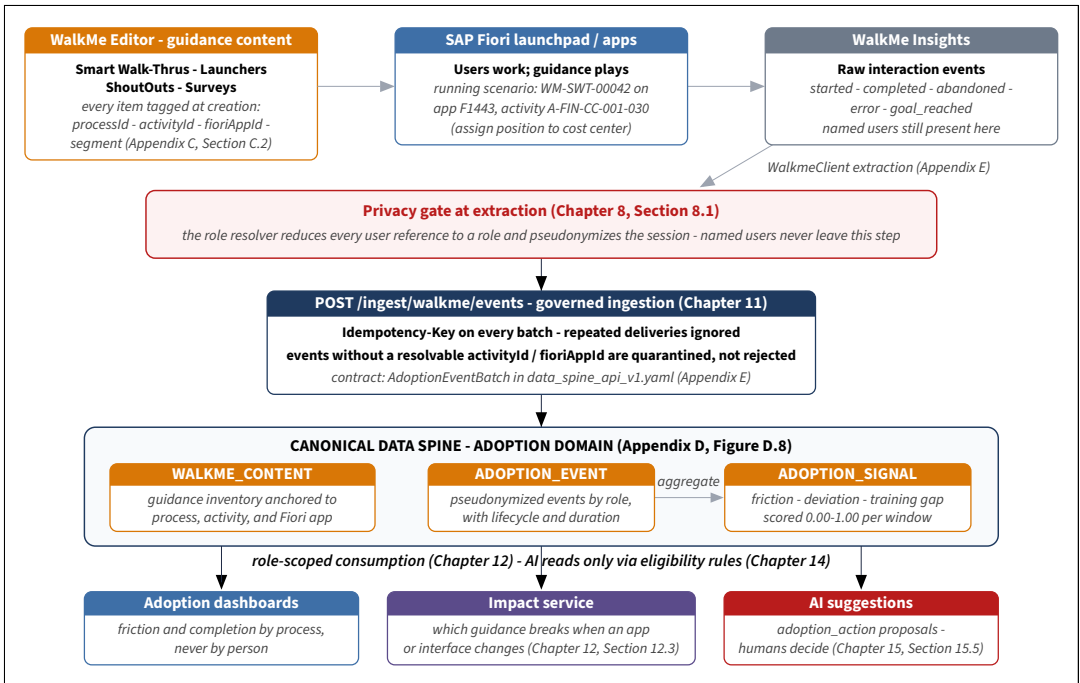


Figure C.1: The Adoption Layer: From Tagged WalkMe Content through the Privacy Gate and Governed Ingestion to Data Spine Tables and Their Consumers

C.2 Content Attributes

Every WalkMe item, whether a Smart Walk-Thru, a Launcher, a ShoutOut, or a Survey, carries the attributes of [Table C.1](#) as custom data maintained in the WalkMe Editor at authoring time. The identifiers follow the same conventions we have used throughout the book, which is to say stable governed business identifiers rather than tool-internal GUIDs (see Chapter 3, Section 3.2). The `walkme_item_id` is the data spine key. WalkMe's own internal identifier remains in WalkMe, much as the SAP Signavio diagram ID and the SAP LeanIX GUID remain reference columns in their own domains.

Attribute	Purpose and Permitted Values	Example
<code>walkme_item_id</code>	Stable data spine key of the guidance item	WM-SWT-00042
<code>content_type</code>	Denotes the content type: <code>smart_walkthru</code> , <code>shoutout</code> , <code>launcher</code> , or <code>survey</code>	<code>smart_walkthru</code>
<code>content_name</code>	Business-readable name, following the process vocabulary	Assign position to cost center-guided

Table C.1: Content Attributes: What Every WalkMe Item Carries (Data Spine Table `WALKME_CONTENT`)

Attribute	Purpose and Permitted Values	Example
process_id	The owning process in the data spine hierarchy	P-FIN-CC-001
activity_id	The L4/L5 activity the item teaches; mandatory for Smart Walk-Thrus and Launchers	A-FIN-CC-001-030
fiori_app_id	The transactional surface the item plays on	F1443
segment	Target audience as a role, never a person	master_data_steward
lifecycle_state	Denotes the lifecycle state (draft, published, or archived); only published content is ingested	published
version_label	Content version, release-managed alongside the process version	v1.4

Table C.1: Content Attributes: What Every WalkMe Item Carries (Data Spine Table WALKME_CONTENT) (Cont.)

Which anchors are mandatory depends on the content type. Figure C.2 summarizes the anchoring model. Smart Walk-Thrus and Launchers teach a specific activity, so they must carry the full chain down to the SAP Fiori app. ShoutOuts address a process audience, so they anchor at process level with a segment. Surveys anchor wherever the feedback belongs. The rule we outlined in Chapter 7, Section 7.3 is unforgiving on purpose, because content that can't say which process it serves should not go live.

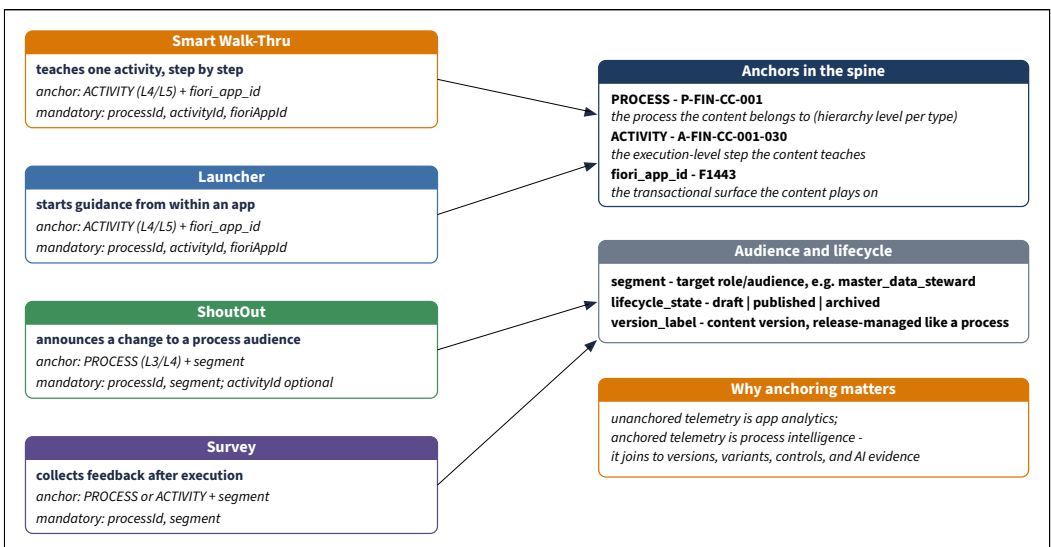


Figure C.2: The Content Anchoring Model: What Each Content Type Must Declare Before It Is Published

C.3 Event Attributes

Adoption events are the busiest data in the data spine, so the attribute model keeps them lean: identifiers, anchors inherited from the content item, a role, an event type, and timing. [Table C.2](#) defines the model for this. Two of these attributes do the privacy work described in Chapter 8, Section 8.1: `user_role` carries a role rather than a named user, and `session_ref` is a pseudonymized token that lets you reassemble a journey without identifying anyone. The role resolver in the extraction client (see [Appendix E](#)) produces both before any event leaves the WalkMe boundary.

Attribute	Purpose and Permitted Values	Example
<code>event_id</code>	Stable event identifier from WalkMe Insights	EVT-2026-0714-081512-4471
<code>walkme_item_id</code>	The guidance item that produced the event	WM-SWT-00042
<code>activity_id</code>	Inherited from the content item; resolves the event to the process model	A-FIN-CC-001-030
<code>fiori_app_id</code>	Surface on which the interaction happened	F1443
<code>user_role</code>	Role of the acting user—never a named user (see Chapter 8, Section 8.1)	master_data_steward
<code>event_type</code>	Denotes the event type: started, completed, abandoned, error, or goal_reached	completed
<code>event_timestamp</code>	Coordinated Universal Time (UTC) timestamp of the interaction	2026-07-14T08:15:12Z
<code>session_ref</code>	Pseudonymized session token for journey analysis	c1f0...e9 (opaque)
<code>duration_seconds</code>	Time from start to terminal event	184

Table C.2: Event Attributes (Data Spine Table ADOPTION_EVENT)

[Listing C.1](#) shows an event batch as the ingestion endpoint receives it. The payload is the `AdoptionEventBatch` schema from the contract in [Appendix E](#), and the batch header carries the idempotency key, so a retried delivery cannot double count adoption.

POST /ingest/walkme/events
Idempotency-Key: 4f6a1c1e-6c9b-4f0e-9f4e-1a2b3c4d5e6f

```
{
  "events": [
    {
      "eventId": "EVT-2026-0714-081512-4471",
      "walkmeItemId": "WM-SWT-00042",
      "activityId": "A-FIN-CC-001-030",
      "fioriAppId": "F1443",
      "userRole": "master_data_steward",
      "eventType": "completed",
      "eventTimestamp": "2026-07-14T08:15:12Z",
      "durationSeconds": 184
    }
  ]
}
```

Listing C.1: An Adoption Event Batch as Delivered to the Data Spine (AdoptionEventBatch, Appendix E Contract)

C.4 Signal Attributes

Raw events tell you what happened, while signals tell you what the events mean. The aggregation described in Chapter 8, Section 8.4 condenses events into three signal types per activity and window, each scored between 0.00 and 1.00. Friction covers abandonment, errors, and duration well beyond the activity's baseline. Deviation covers execution paths that bypass the guided flow. The training gap signal covers populations of a segment who never complete guidance for an activity they perform. Table C.3 defines these attributes.

Attribute	Purpose and Permitted Values	Example
signal_id	Stable signal identifier	SIG-FRC-001-030-W29
activity_id	The activity the signal scores	A-FIN-CC-001-030
signal_type	Denotes the signal type: friction, deviation, or training_gap	friction
signal_score	0.00–1.00; higher means worse	0.31
aggregation_window	Denotes the aggregation window: daily, weekly, or release	weekly

Table C.3: Signal Attributes (Data Spine Table ADOPTION_SIGNAL)

Attribute	Purpose and Permitted Values	Example
<code>computed_at</code>	When the aggregation ran	2026-07-19T02:00:00Z
<code>fed_to_ai_flag</code>	Whether this signal entered an AI evidence set—the explicit, auditable handover	true

Table C.3: Signal Attributes (Data Spine Table `ADOPTION_SIGNAL`) (Cont.)

We will emphasize one attribute: `fed_to_ai_flag` makes the boundary between measurement and AI consumption explicit. A signal is only available as AI evidence once it has passed the eligibility rules outlined in Chapter 14, and the flag records that it has passed. When an `adoption_action` AI suggestion later cites the signal in its explanation (see [Appendix G](#)), the audit trail runs unbroken from the suggestion back through the signal and its events to the tagged content item that produced them.

C.5 Governance Rules for the Adoption Layer

Five rules keep the model trustworthy in operation. They repeat patterns from elsewhere in the book, which is the point. The adoption layer earns its place in the data spine by playing by the following rules:

- **Published content only**

Items in `draft` or `archived` state are not ingested and produce no data spine events. This mirrors the SAP Signavio rule that draft models never leave their system of record (see Chapter 3, Section 3.5).

- **Quarantine rather than rejection**

Events that arrive without a resolvable `activity_id` or `fiori_app_id` are held for repair instead of being discarded. Telemetry is too valuable to lose to a tagging gap, and too unreliable to admit with broken anchors (the contract from [Appendix E](#)).

- **Roles, never named users**

No attribute in [Table C.1](#), [Table C.2](#), or [Table C.3](#) can carry a personal identifier, and the privacy gate sits at extraction rather than at reporting. Compliance by schema beats compliance by policy (see Chapter 8, Section 8.1).

- **Content is versioned like process**

A guidance item's `version_label` moves with the process version it teaches, and when a process version retires, its guidance is reanchored or archived in the same release act (see Chapter 7, Section 7.3).

- **Signals feed AI only through eligibility**

The grounding filter, rather than the aggregation job, sets `fed_to_ai_flag`. What AI may see is a governance decision rather than a pipeline default (see Chapter 14).

With these rules in place, the adoption layer closes the loop the book has been building since Chapter 7. Processes are modeled in SAP Signavio, realized in the application landscape of SAP LeanIX, executed on SAP Fiori, and guided by WalkMe. The data spine can then say, per activity and per role, whether the organization actually works the way its process models claim it does. That answer, scored and governed, is the output covered in Chapter 14 through Chapter 16, and it is what AI returns as suggestions for people to decide on.

Appendix D

Canonical Data Model for the SAP Business Technology Platform Data Spine

This appendix is the complete reference for the canonical data spine introduced in Chapter 9 and implemented on SAP HANA Cloud in Chapter 10 and Chapter 11. It collates every table of the data spine into a single entity-relationship (ER) model: the process-semantic core described in Chapter 3 and Chapter 9, the enterprise-structure tables from Chapter 5 and Chapter 6, and the ingestion, adoption, execution, AI, and audit structures introduced in Chapter 8, Chapter 11, Chapter 12, Chapter 14, Chapter 15, Chapter 16, and Chapter 19. The model comprises twenty-nine tables in seven domains, joined by thirty-five governed relationships.

The model ships in two forms. The figures in this appendix present each domain as a printable ER diagram. The accompanying download provides the same model as a machine-readable Mermaid file, *Appendix_D_Canonical_Data_Spine.mmd*. Every figure in this appendix was generated from this file. [Section D.9](#) explains how to work with the file directly.

D.1 How to Read This Appendix

Each section that follows covers one domain. You get a short account of what the domain stores and which chapters define it, a summary table of its entities, and the ER diagram with full attribute lists. The diagrams use crow's-foot notation, where `||-o{` reads as one-to-many and `||-||` as one-to-one. Attribute rows carry the column type, the column name, a key marker where one applies (*PK* for primary key and *FK* for foreign key), and a comment giving permitted values or an example.

Four conventions hold across all seven domains:

- **Stable business identifiers**

Data spine keys such as `P-FIN-CC-001` or `A-FIN-CC-001-030` are governed business identifiers rather than tool-internal GUIDs. External references such as `signavio_diagram_id` and `leanix_factsheet_id` point back to the system of record (see Chapter 3, Section 3.2 and Chapter 5, Section 5.4).

- **Lifecycle everywhere**
Nearly every entity carries a `lifecycle_state` column. Only published or approved content takes part in execution and AI grounding, and draft content never leaves its system of record (see Chapter 3, Section 3.5 and Chapter 14, Section 14.3).
- **Roles, never named users**
Wherever people appear—whether as adoption events, audit actors, or validators—they appear as roles. This is the privacy rule we covered in Chapter 8 enforced in the data model itself.
- **AI proposes, humans decide**
The AI tables can record suggestions and explanations, but no AI-writable column can express acceptance. The `decided_by` and `validated_by` columns always hold a human role (see Chapter 15, Section 15.5).

Figure D.1 places the seven domains in context, together with the systems of record that feed them and the execution, AI, and audit consumers that read them.

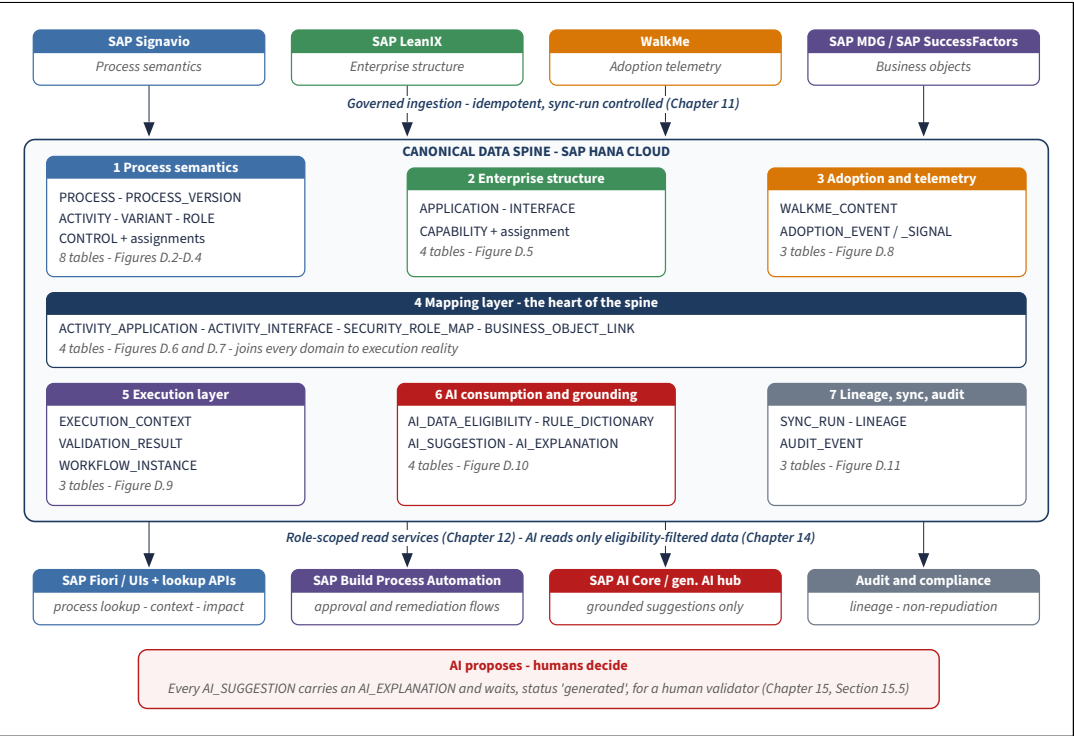


Figure D.1: The Canonical Data Spine at a Glance: Seven Domains Fed by the Systems of Record and Consumed by Execution, AI, and Audit Services

D.2 Domain 1: Process Semantics

The process semantics domain is the system-of-record mirror of SAP Signavio, and it uses the vocabulary we established in Chapter 3 and Chapter 9 (see [Table D.1](#)): processes in a five-level hierarchy, the L4/L5 activities that execution depends on, their versions and variants, performing roles, and the controls that govern them. `PROCESS` is self-referencing through `parent_process_id`, which is how the L1-to-L5 hierarchy fits in a single table. `PROCESS_VERSION` freezes an approved set of activities so that execution and AI grounding always reference an approved version rather than a moving target.

Entity	Purpose	Reference
PROCESS	Process hierarchy L1–L5 with ownership, lifecycle, and the SAP Signavio diagram reference	Chapter 3, Section 3.2 and Chapter 9, Section 9.2
PROCESS_VERSION	Approved, immutable versions of a process; the anchor for execution and grounding	Chapter 3, Section 3.5
ACTIVITY	L4/L5 execution-level activities with mandatory execution attributes and the primary SAP Fiori surface	Chapter 3, Section 3.3
VARIANT	Localizations of a process by region, company code, legal entity, or line of business	Chapter 3, Section 3.4
ROLE/ACTIVITY_ROLE	Performing roles and their RACI assignment to activities	Chapter 3, Section 3.3
CONTROL/ACTIVITY_CONTROL	Preventive, detective, and corrective controls and their assignment with evidence expectations	Chapter 9, Section 9.2

Table D.1: Entities of the Process Semantics Domain

[Figure D.2](#) shows the first of three views of the process-semantic domain: the `PROCESS` hierarchy together with its approved versions and their variants.

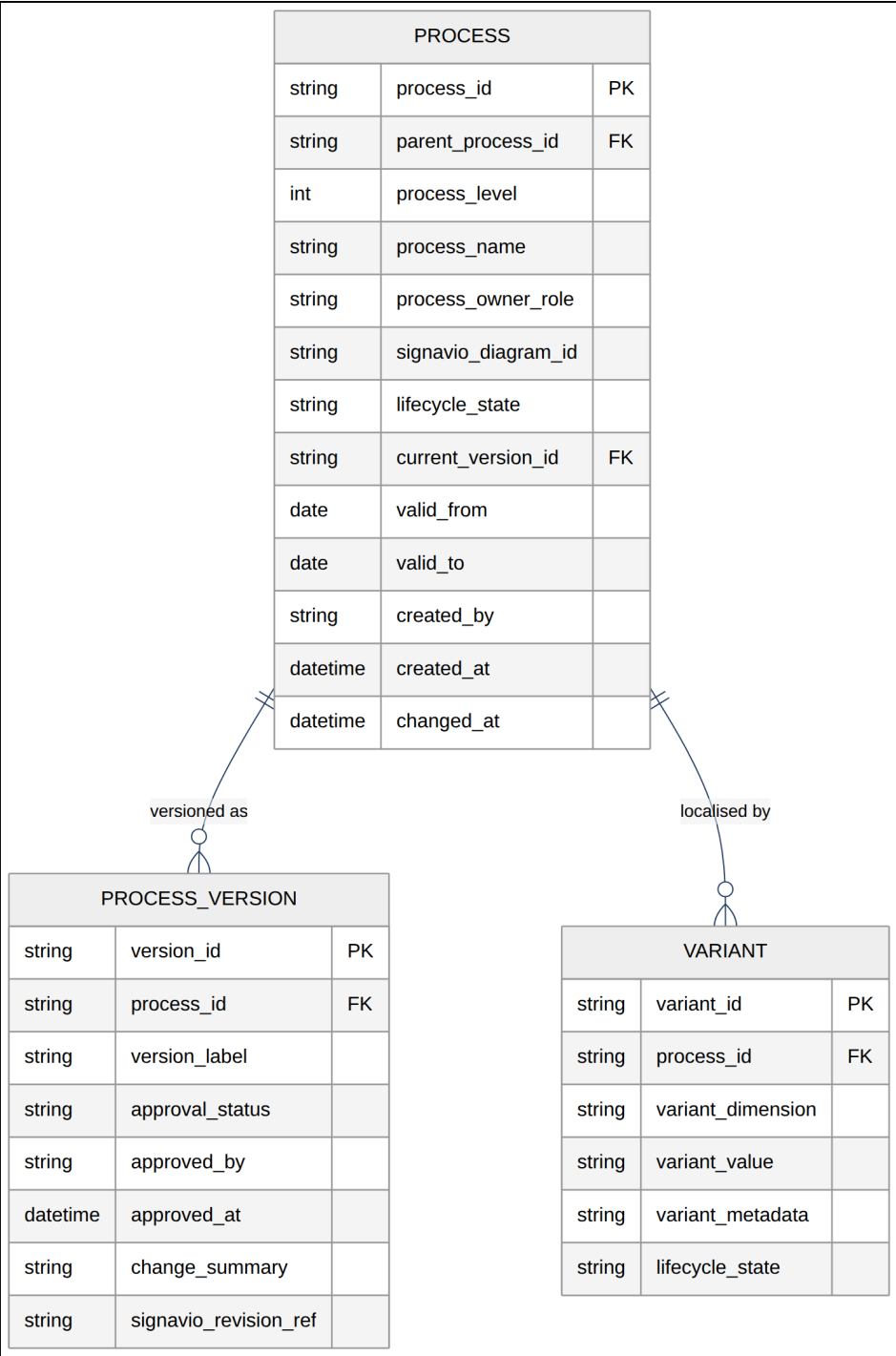


Figure D.2: Process-Semantic Domain (1 of 3): The PROCESS Hierarchy, Its Approved Versions, and Its Variants

Figure D.3 shows the second view of the process-semantic domain: the `ACTIVITY` table, frozen per approved process version.

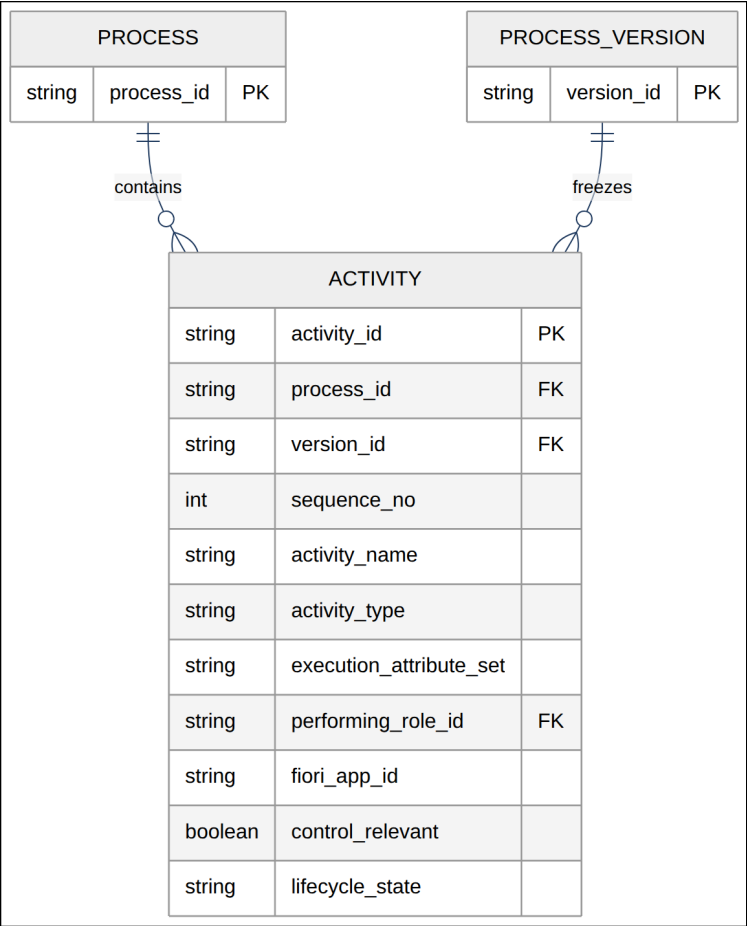


Figure D.3: Process-Semantic Domain (2 of 3): The `ACTIVITY` Table, Frozen per Approved Process Version

Figure D.4 completes the process-semantic domain with the third view: role assignments and the controls that govern activities.

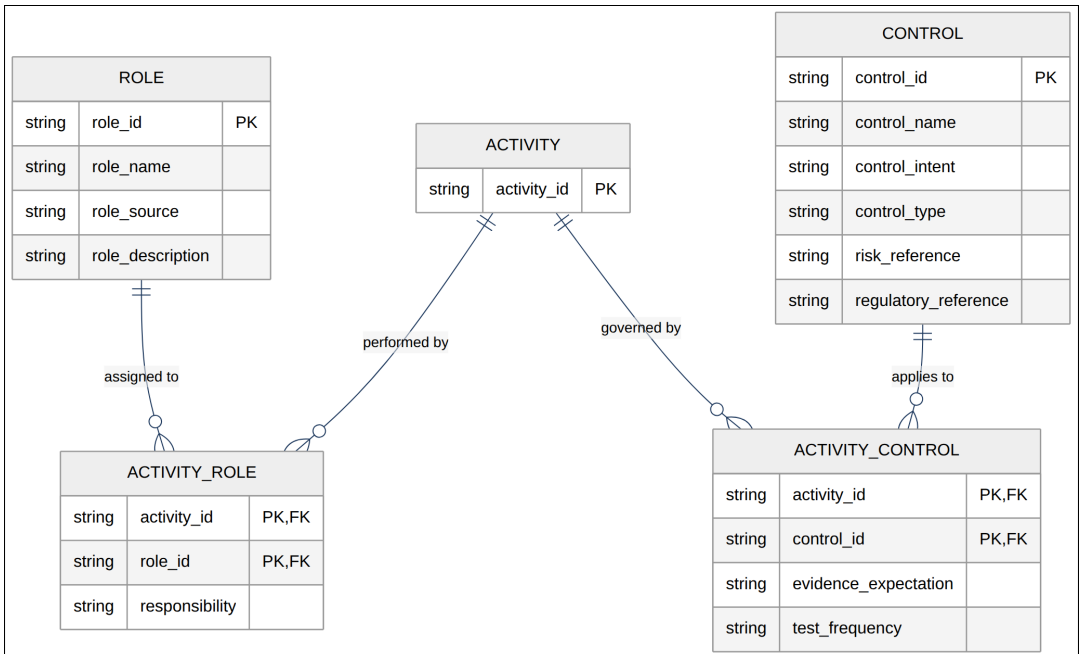


Figure D.4: Process-Semantic Domain (3 of 3): Roles and Governance Controls

D.3 Domain 2: Enterprise Structure

The enterprise structure domain (see [Table D.2](#) and [Figure D.5](#)) mirrors SAP LeanIX (Chapter 5, Chapter 6, and Chapter 9). Applications, interfaces, and business capabilities enter the data spine through governed ingestion, keyed by their governed external identifiers rather than by the SAP LeanIX GUID, which is retained only as a reference column. INTERFACE is modeled with both a provider and a consumer application, which is what later allows the mapping layer and the impact service to walk from a changed interface to every affected activity.

Entity	Purpose	Reference
APPLICATION	Application portfolio with category, hosting, lifecycle, and ownership	Chapter 5, Section 5.2
INTERFACE	Directed integrations between applications, with protocol, frequency, and the business object carried	Chapter 5, Section 5.3

Table D.2: Entities of the Enterprise Structure Domain

Entity	Purpose	Reference
CAPABILITY	Business capability hierarchy (L1–L3), self-referencing like PROCESS	Chapter 5, Section 5.2
APPLICATION_CAPABILITY	Which applications realize which capabilities, as primary or secondary support	Chapter 6, Section 6.2

Table D.2: Entities of the Enterprise Structure Domain (Cont.)

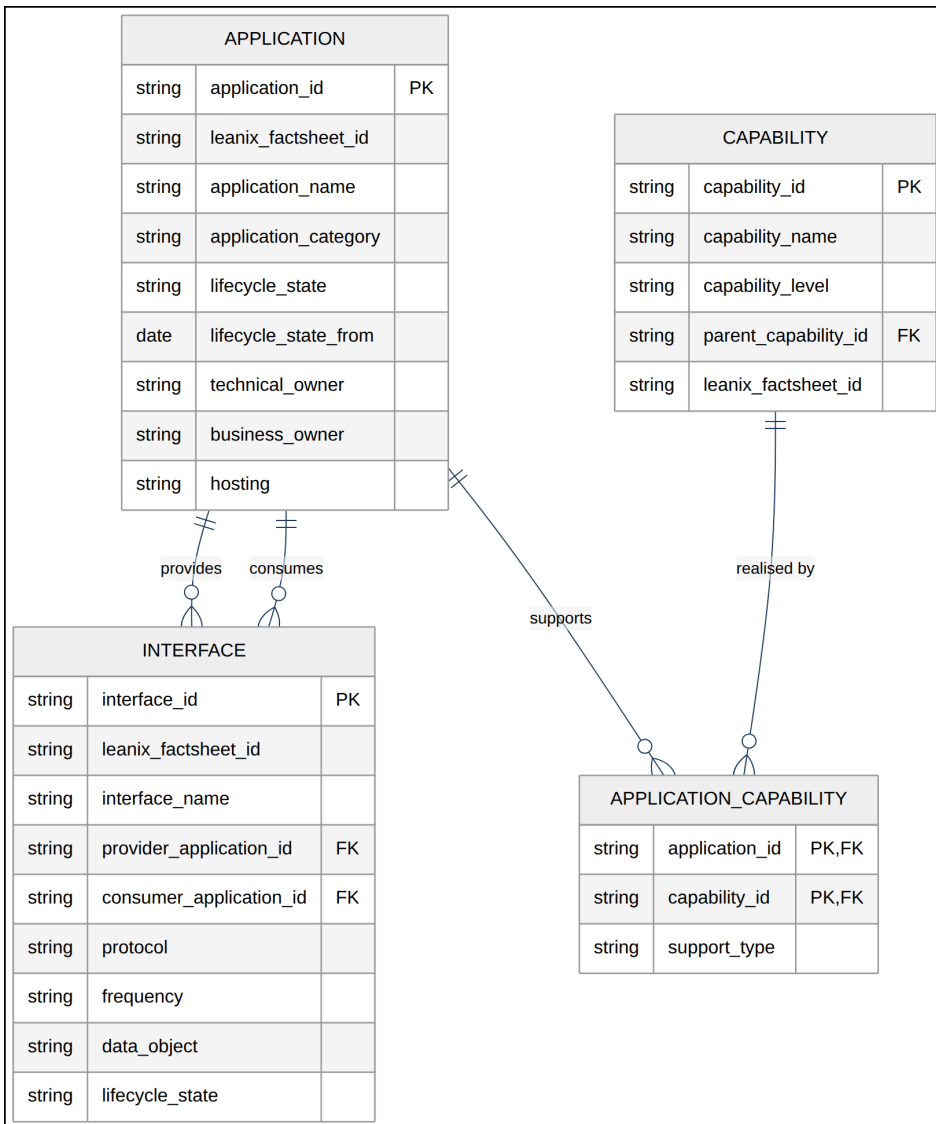


Figure D.5: Enterprise-Structure Domain: APPLICATION, INTERFACE, and the CAPABILITY Hierarchy

D.4 Domain 3: The Mapping Layer

The mapping layer (see [Table D.3](#)) is the heart of the data spine—the four tables that join process semantics to enterprise structure and to execution reality (see Chapter 6, Chapter 9, and Chapter 10). Without them, the data spine would be two well-governed but disconnected inventories. `ACTIVITY_APPLICATION` records where an activity executes, down to the SAP Fiori app, transaction code, and configuration reference. Every mapping carries a `mapping_status` and a confirming role, because a proposed mapping is only a hypothesis, not a fact.

The fourth table, `BUSINESS_OBJECT_LINK`, carries the book’s running scenario, the link between a cost center governed in SAP Master Data Governance and a position in SAP SuccessFactors. Each link records the rule that validated it, along with a `link_status` that can degrade to `broken` or `pending_review` when either side changes. That degradation is what triggers the validation and workflow machinery of the execution layer and AI consumption and grounding.

Entity	Purpose	Reference
<code>ACTIVITY_APPLICATION</code>	Where an activity executes: application, SAP Fiori app, transaction, configuration reference, or confirmation status	Chapter 6, Section 6.3
<code>ACTIVITY_INTERFACE</code>	Which interfaces an activity triggers or consumes, with direction and trigger type	Chapter 6, Section 6.3
<code>SECURITY_ROLE_MAP</code>	Technical authorizations per activity (Transaction PFCG role, Identity Authentication group, or SAP Authorization and Trust Management service [XSUAA] scope) with separation of duties (SoD) flag	Chapter 9, Section 9.3
<code>BUSINESS_OBJECT_LINK</code>	Cross-system object links, e.g., SAP Master Data Governance cost center to SAP SuccessFactors position, with validating rule and status	Chapter 9, Section 9.3 and Chapter 12, Section 12.3

Table D.3: Entities of the Mapping Layer

[Figure D.6](#) shows the first of two views of the mapping layer: activity-to-application, activity-to-interface, and security-role mappings.

[Figure D.7](#) completes the mapping layer with the second view: cross-system business object links validated by the rule dictionary.

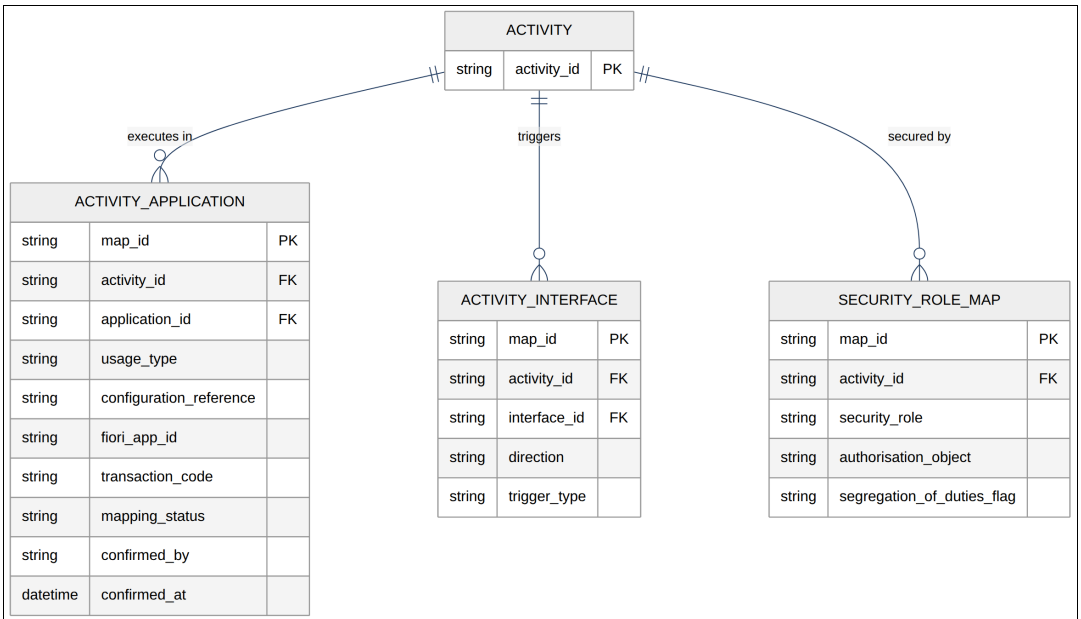


Figure D.6: Mapping Layer (1 of 2): Activity-to-Application, Activity-to-Interface, and Security-Role Mappings

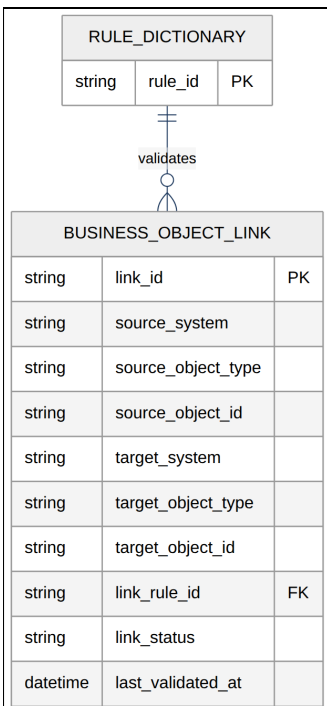


Figure D.7: Mapping Layer (2 of 2): Cross-System Business Object Links Validated by the Rule Dictionary

D.5 Domain 4: Adoption and Telemetry

The adoption domain (see [Figure D.8](#) and [Table D.4](#)) mirrors WalkMe (see Chapter 7 and Chapter 8). WALKME_CONTENT anchors every guidance item, whether a Smart Walk-Thru, a ShoutOut, a Launcher, or a Survey, to the process and activity it supports, which is what makes adoption telemetry process-aware rather than app-aware. Raw ADOPTION_EVENT records are pseudonymized at source and carry a user role in place of a named user. ADOPTION_SIGNAL holds the aggregated friction, deviation, and training-gap scores that the AI domain is allowed to consume, and fed_to_ai_flag makes that handover explicit and auditable.

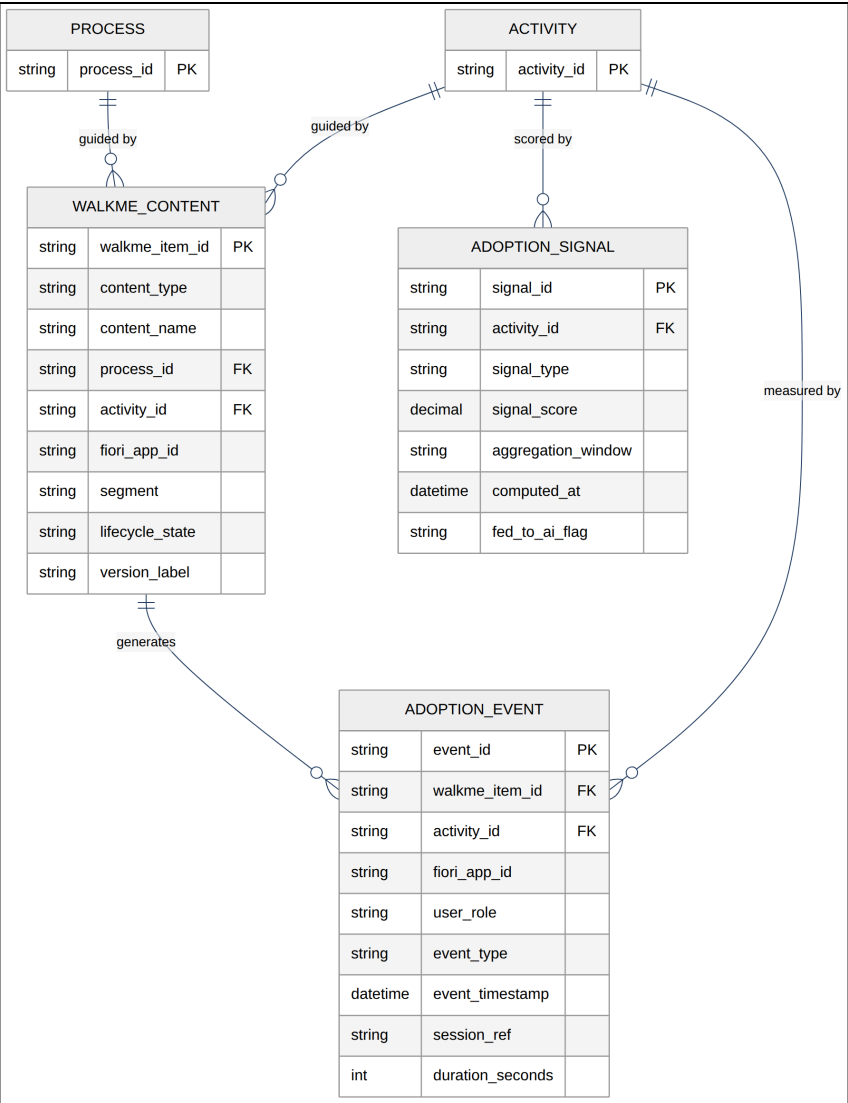


Figure D.8: Adoption and Telemetry Domain: WalkMe Content Anchored to Processes and Activities, with Events and Aggregated Signals

Entity	Purpose	Reference
WALKME_CONTENT	Guidance inventory anchored to process, activity, and SAP Fiori app, with lifecycle and version	Chapter 7, Section 7.3
ADOPTION_EVENT	Pseudonymized interaction events (started, completed, abandoned, error, or goal reached) by role	Chapter 8, Section 8.1
ADOPTION_SIGNAL	Aggregated friction, deviation, and training-gap scores per activity and window	Chapter 8, Section 8.4

Table D.4: Entities of the Adoption and Telemetry Domain

D.6 Domain 5: The Execution Layer

The execution layer (see [Table D.5](#) and [Figure D.9](#)) records what happens when the data spine is used (see [Chapter 11](#), [Chapter 12](#), and [Chapter 16](#)). Every call into the exposure services opens an `EXECUTION_CONTEXT` with a correlation identifier that traces the request end to end across SAP BTP services, whether the call is a lookup, an enrichment, an impact analysis, a workflow trigger, or an AI grounding request. Validation outcomes attach to the context through `VALIDATION_RESULT`, each referencing the rule that produced it, and escalations become `WORKFLOW_INSTANCE` records bound to SAP Build Process Automation. The `decided_by` column of a workflow instance always holds a human role and never an AI service, which is this book's design boundary expressed at a schema level.

Entity	Purpose	Reference
EXECUTION_CONTEXT	One record per data spine invocation, with requesting service, correlation ID, and outcome	Chapter 12, Section 12.1
VALIDATION_RESULT	Rule outcomes (pass, fail, or warning) with fuzzy scores where applicable	Chapter 11, Section 11.5
WORKFLOW_INSTANCE	Approval, remediation, and AI-review workflows in SAP Build Process Automation; decided by humans	Chapter 16, Section 16.3

Table D.5: Entities of the Execution Layer

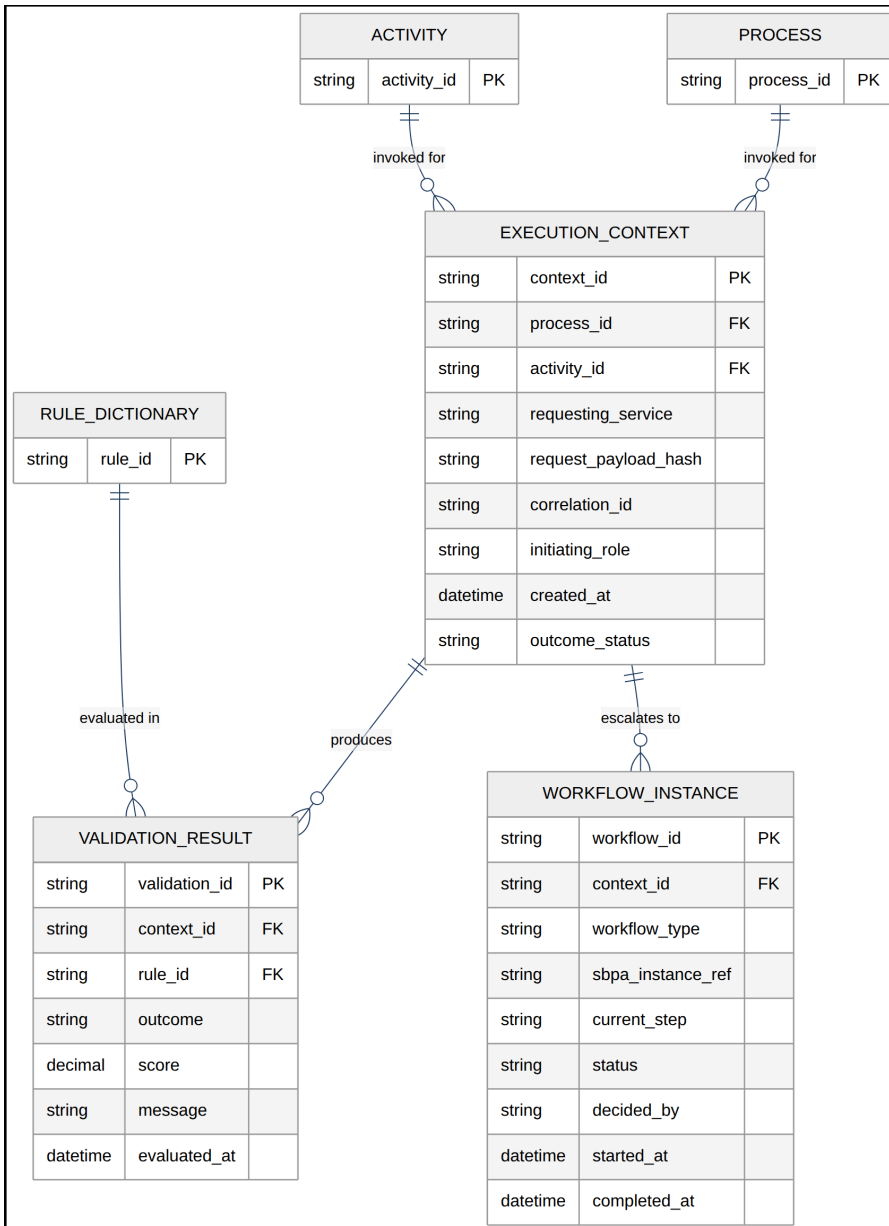


Figure D.9: Execution Layer: Execution Contexts Producing Validation Results and Escalating to Human-Decided Workflows

D.7 Domain 6: AI Consumption and Grounding

The AI domain (see [Table D.6](#)) implements the concepts outlined in Chapter 13 through Chapter 15 and works hand in hand with the grounding rules in [Appendix F](#). `AI_DATA_ELIGIBILITY` is the whitelist. Per entity, it names the columns AI may see, the roles that can ground AI on the data, and the version constraint, which can only be approved or published. `RULE_DICTIONARY` stores validation and grounding rules as governed data rather than as code, so adding a rule is a governance act with an owner and a chapter reference rather than a deployment. Every `AI_SUGGESTION` must reference an `AI_EXPLANATION`, and the one-to-one relationship shown in [Figure D.10](#) is deliberate. A suggestion's status can only be moved to accepted or rejected by the human validator recorded in `validated_by`.

Entity	Purpose	Reference
AI_DATA_ELIGIBILITY	Field-level whitelist, role scope, and version constraint per data spine entity	Chapter 14, Section 14.2
RULE_DICTIONARY	Governed structural, semantic, fuzzy, grounding, and eligibility rules as data	Chapter 14, Section 14.4
AI_SUGGESTION	AI proposals (process change, mapping fix, control gap, or adoption action) awaiting human validation	Chapter 15, Section 15.5
AI_EXPLANATION	Evidence references, reasoning summary, and model and prompt version for every suggestion	Chapter 15, Section 15.4

Table D.6: Entities of the AI Consumption and Grounding Domain

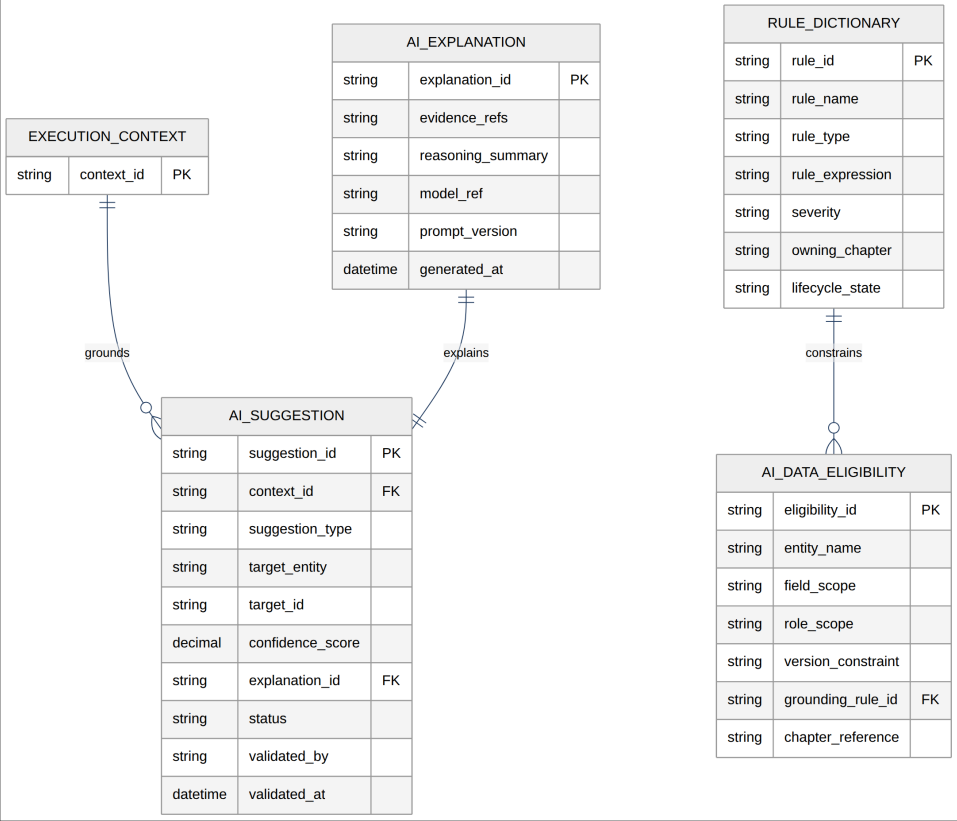


Figure D.10: AI Consumption and Grounding: Eligibility Rules Constrain What AI Sees; Every Suggestion Carries Exactly One Explanation

D.8 Domain 7: Lineage, Synchronization, and Audit

The final domain (see Table D.7 and Figure D.11) is the data spine's memory of itself (see Chapter 9, Section 9.4 ; Chapter 11, Section 11.3 to Section 11.5 ; Chapter 18 ; and Chapter 19). Every ingestion batch runs inside a SYNC_RUN with read, written, and rejected counters, which makes reconciliation a first-class operation rather than an afterthought. LINEAGE records, per data spine record, which source object and which transformation produced it in which run. That is the answer to the auditor's question about where a number came from. AUDIT_EVENT captures every create, update, approval, rejection, and AI generation, with before and after hashes and a non-repudiation reference (see Chapter 2, Section 2.10).

Entity	Purpose	Reference
SYNC_RUN	Full, delta, and reconciliation runs per source system with record counters	Chapter 11, Section 11.3

Table D.7: Entities of the Lineage, Synchronization, and Audit Domain

Entity	Purpose	Reference
LINEAGE	Source object to data spine record traceability per run and transformation	Chapter 9, Section 9.4
AUDIT_EVENT	Immutable audit trail with actor, before/after hashes, and non-repudiation reference	Chapter 19, Section 19.2

Table D.7: Entities of the Lineage, Synchronization, and Audit Domain (Cont.)

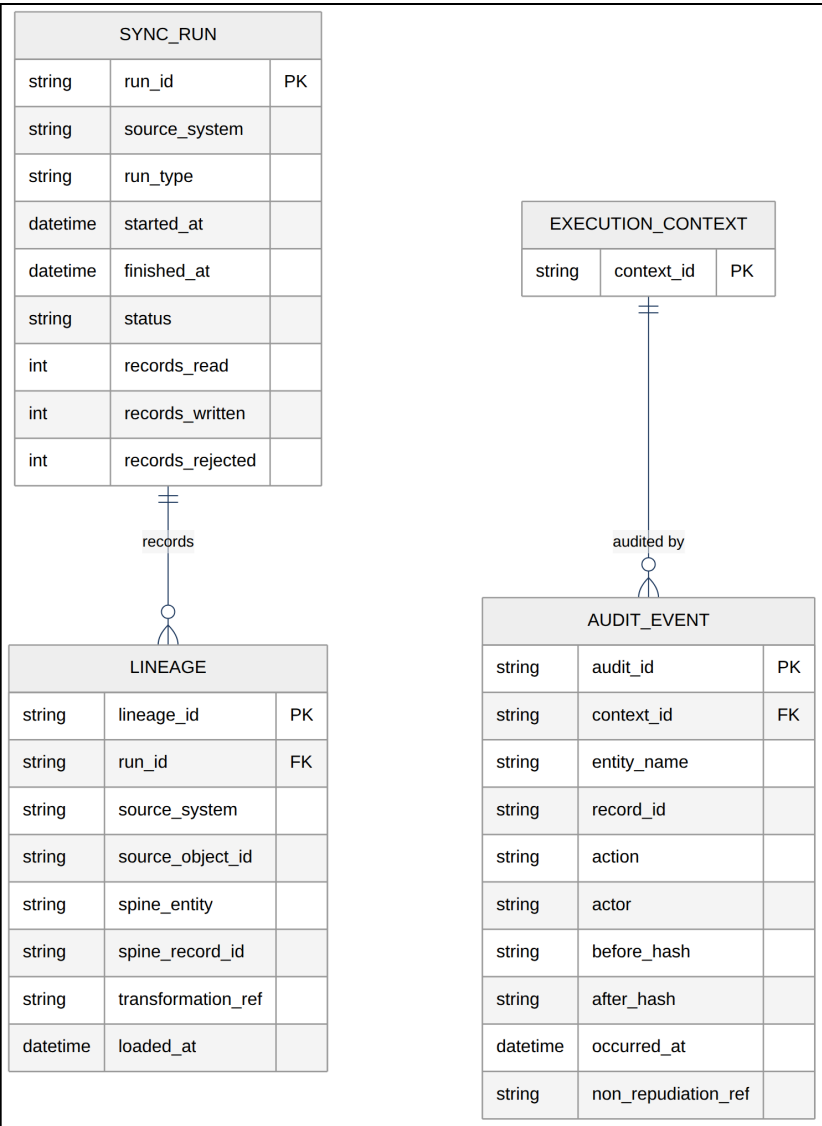


Figure D.11: Lineage, Synchronization, and Audit: Sync Runs Record Lineage; Execution Contexts Are Audited with Non-Repudiation References

D.9 Working with the Machine-Readable Model

The download package for this appendix contains the collated model as *Appendix_D_Canonical_Data_Spine.mmd*, a single Mermaid ER diagram covering all twenty-nine entities and thirty-five relationships. To view it, paste the file's contents into the Mermaid Live Editor at <https://mermaid.live> or any Mermaid 10 or later renderer. Every diagram in this appendix is an extract of that file. Because the model is plain text, it diffs and versions like code, and that is how you should treat it. The model belongs in the same repository, and under the same governance, as the ingestion and exposure services covered in [Appendix E](#).

[Listing D.1](#) shows the file's structure, using the `PROCESS` entity and two of its relationships. Each attribute row carries the type, the column name, an optional key marker, and a quoted comment with permitted values or an example, which is the same information you will find in the rendered figures.

```
PROCESS {
    string  process_id PK "Stable process identifier (e.g. P-FIN-CC-001)"
    string  parent_process_id FK "Self-reference for hierarchy (L1..L5)"
    int     process_level "1..5; execution depth is L4/L5"
    string  process_name
    string  process_owner_role
    string  signavio_diagram_id "External reference to SAP Signavio"
    string  lifecycle_state "draft | approved | published | retired"
    string  current_version_id FK
    date    valid_from
    date    valid_to
    string  created_by
    datetime created_at
    datetime changed_at
}

PROCESS ||--o{ PROCESS_VERSION : "versioned as"
PROCESS ||--o{ ACTIVITY          : "contains"
```

Listing D.1: Structure of the Mermaid Source: The PROCESS Entity and Two of Its Relationships (Excerpt from Appendix_D_Canonical_Data_Spine.mmd)

Translating the model to SAP HANA Cloud data definition language (DDL) is mechanical. Entities become column tables, PK markers become primary keys, FK markers become referential constraints, and the quoted comments become `COMMENT` clauses. Chapter 10 walks through this translation for the process-semantic domain, and the remaining domains follow the same pattern. Whichever route you take, keep the Mermaid file authoritative and regenerate the DDL from the model rather than the other way around.

Appendix E

Integration Application Programming Interface Specifications

This appendix documents the first-version API contract of the SAP BTP data spine, along with the two Python templates that implement it. Between them they turn the canonical data model of [Appendix D](#) into running services: governed ingestion from the systems of record (Chapter 11), synchronization control (Chapter 11, Section 11.3 to Section 11.5), read services for execution and user interfaces (Chapter 12), and AI suggestion handling with mandatory human-in-the-loop validation (Chapter 14 and Chapter 15).

The download package for this appendix contains three files:

- `spec/data_spine_api_v1.yaml`: The OpenAPI 3.0 contract for all data spine services.
- `python_templates/extraction_clients.py`: Client templates that extract from SAP Signavio, SAP LeanIX, and WalkMe and deliver idempotently to the ingestion endpoints.
- `python_templates/spine_service.py`: A FastAPI skeleton of the exposure and AI services that is ready to deploy to SAP BTP, Cloud Foundry runtime with the Python build-pack.

All three are templates in the strictest sense. They encode this book's patterns, but server URLs, tenant identifiers, authentication endpoints, and field mappings all have to be adapted to your landscape before anything runs. [Section E.6](#) collects the adaptation points into one checklist. The code assets in [Appendix G](#) (the rule dictionary, fuzzy evaluator, semantic graph, and grounding filter) plug into the service skeleton documented in this section.

E.1 The API Surface at a Glance

The contract groups the data spine's services into four areas, each with its own read or write character and its own OAuth 2.0 scope. [Figure E.1](#) shows the full surface and the two templates that implement it.

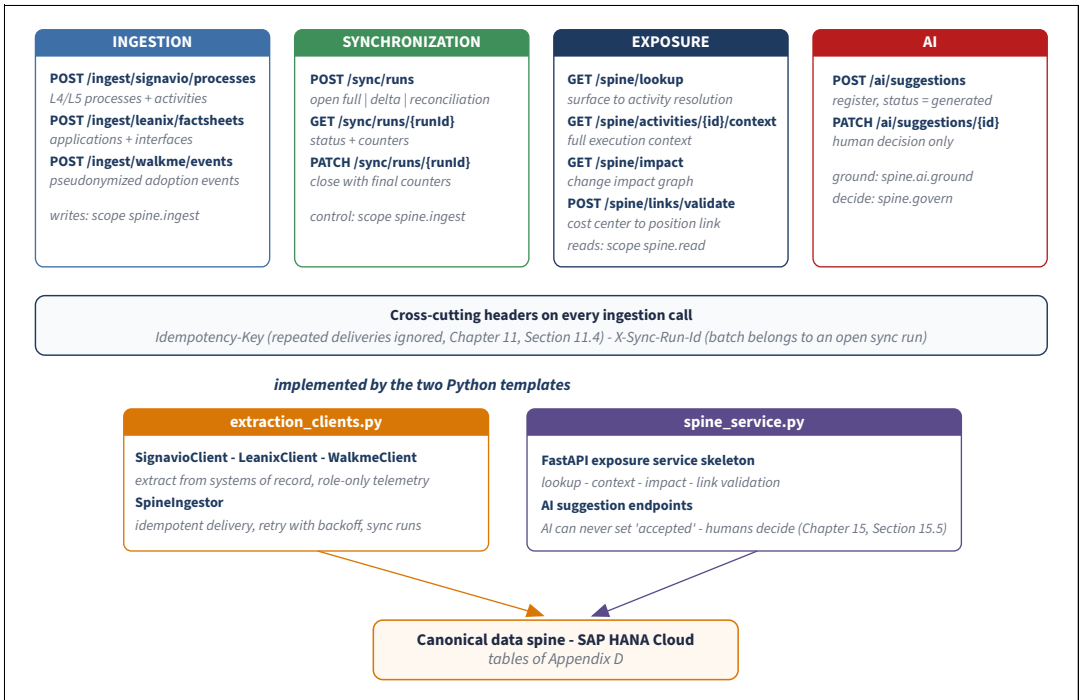


Figure E.1: The Data Spine API Surface: Four Service Areas Defined in `data_spine_api_v1.yaml`, Implemented by the Two Python Templates

Table E.1 lists every endpoint with its purpose and required scope. Note the asymmetry that runs through the whole design. Ingestion and governance services can write, while consuming applications and AI services can only read, scoped by role (see Chapter 2, Section 2.8 and Chapter 14, Section 14.2).

Endpoint	Purpose	Scope
POST /ingest/signavio/processes	Ingest L4/L5 process and activity payloads from SAP Signavio	spine.ingest
POST /ingest/leanix/factsheets	Ingest application and interface factsheets from SAP LeanIX	spine.ingest
POST /ingest/walkme/events	Ingest pseudonymized WalkMe adoption events (roles, never named users)	spine.ingest
POST /sync/runs	Open a sync run: full, delta, or reconciliation	spine.ingest
GET /sync/runs/{runId}	Read sync-run status and reconciliation counters	spine.ingest

Table E.1: Endpoints of the Data Spine API (v1)

Endpoint	Purpose	Scope
PATCH /sync/runs/{runId}	Close a sync run with final counters	spine.ingest
GET /spine/lookup	Resolve an SAP Fiori app, transaction, or interface to its owning activity and process	spine.read
GET /spine/activities/{id}/context	Full execution context of an activity, role-scoped	spine.read
GET /spine/impact	Change impact graph for an application or interface	spine.read
POST /spine/links/validate	Validate a cross-system business object link (running scenario)	spine.read
POST /ai/suggestions	Register an AI-generated suggestion, with the status generated	spine.ai.ground
PATCH /ai/suggestions/{id}	Record the human validation decision on a suggestion	spine.govern

Table E.1: Endpoints of the Data Spine API (v1) (Cont.)

E.2 Security Model and Cross-Cutting Headers

Authentication is OAuth 2.0 client credentials against XSUAA (see Chapter 2, Section 2.7). The contract defines four scopes (see [Table E.2](#)), and the separation between them matters more than the mechanics. The scope that lets an AI service read eligibility-filtered data (spine.ai.ground) is not the scope that records a decision (spine.govern). An AI service holding only its grounding scope is technically incapable of accepting its own suggestion.

Scope	Granted to	Permits
spine.ingest	Ingestion and synchronization services only	Writes through the ingestion endpoints and sync-run control
spine.read	Consuming applications and user interfaces (UIs)	Role-scoped reads of lookup, context, impact, and link validation
spine.ai.ground	SAP AI Core/generative AI hub services	Eligibility-filtered reads and registration of suggestions
spine.govern	Governance roles and workflow services	Human validation decisions and governance operations

Table E.2: OAuth 2.0 Scopes of the Data Spine API

Two headers accompany every ingestion call. Idempotency-Key is a client-generated universally unique identifier (UUID), and the data spine ignores repeated deliveries with the same key, which is what makes retry logic safe (see Chapter 11, Section 11.4). X-Sync-Run-Id ties each batch to an open sync run, so every record written can be reconciled against the run's counters (see Chapter 11, Section 11.3). Ingestion responds with HTTP 202 rather than 200, meaning accepted for processing with lineage recorded per record, because validation happens asynchronously against the rule dictionary. Structural failures come back in a failed-record envelope, so a run can close as `partial` instead of failing wholesale.

E.3 The OpenAPI Contract

The contract `data_spine_api_v1.yaml` is the single source of truth for payload shapes, and the schemas mirror the tables in [Appendix D](#) field for field in camelCase. [Listing E.1](#) shows the running-scenario endpoint, validation of a cost-center-to-position link, together with the `BusinessObjectLink` schema it accepts. Note how the description binds the endpoint to the rule dictionary and to fuzzy evaluation where exact keys are unavailable. That is the approach outlined in [Appendix G](#).

```
/spine/links/validate:
  post:
    tags: [Exposure]
    summary: Validate a cross-system business object link
    description: >
      Running-scenario endpoint - validates a link such as a cost
      center in SAP Master Data Governance against a position in
      SAP SuccessFactors, using the rule dictionary and fuzzy
      evaluation where exact keys are unavailable.
    operationId: validateBusinessObjectLink
    requestBody:
      required: true
      content:
        application/json:
          schema: { $ref: '#/components/schemas/BusinessObjectLink' }
    responses:
      '200':
        description: Validation result
        content:
          application/json:
            schema: { $ref: '#/components/schemas/ValidationResult' }

BusinessObjectLink:
  type: object
  required: [sourceSystem, sourceObjectType, sourceObjectId,
```

```
targetSystem, targetObjectType, targetObjectId]
  properties:
    sourceSystem: { type: string, example: sap_mdg }
    sourceObjectType: { type: string, example: cost_center }
    sourceObjectId: { type: string, example: '10010010' }
    targetSystem: { type: string, example: successfactors }
    targetObjectType: { type: string, example: position }
    targetObjectId: { type: string, example: '70001234' }
```

Listing E.1: The Link-Validation Endpoint and Its Request Schema (Excerpt from data_spine_api_v1.yaml)

Two contract details deserve attention before you adapt the file. First, WalkMe events without a resolvable `activityId` or `fioriAppId` are quarantined rather than rejected, because telemetry is too valuable to discard over a mapping gap and too unreliable to let into the data spine unrepaired. Second, the `AiSuggestion` schema makes the explanation object mandatory, with evidence references, reasoning summary, and model reference, so a suggestion without an explanation cannot even be expressed in the contract (see Chapter 15, Section 15.4).

E.4 Extraction Client Templates

The file `extraction_clients.py` provides four cooperating classes. `SignavioClient` extracts published L4/L5 processes and their activities, and only published ones, since draft models never reach the data spine (see Chapter 3, Section 3.5). `LeanixClient` pages through application and interface factsheets via GraphQL and keys every payload by the governed external identifier rather than the SAP LeanIX GUID (see Chapter 5, Section 5.4). `WalkmeClient` extracts adoption events and applies the privacy rule from Chapter 8 at the earliest possible moment, using a role resolver that reduces every user reference to a role before the event leaves the client.

The fourth class, `SpineIngestor`, is the delivery workhorse. It opens and closes sync runs, stamps every batch with an idempotency key, and retries transient failures, meaning HTTP 429, 502, 503, and 504, with exponential backoff. Retrying is safe because the deliveries are idempotent. [Listing E.2](#) shows the delivery loop.

```
def deliver(self, path: str, payload: dict, run_id: str | None =
None) -> dict:
    headers = self._headers()
    headers["Idempotency-Key"] = str(uuid.uuid4())
    if run_id:
        headers["X-Sync-Run-Id"] = run_id
    delay = 2.0
    for attempt in range(1, self.MAX_RETRIES + 1):
```

```

resp = requests.post(
    f"{self.spine_url}{path}", headers=headers,
    json=payload, timeout=120,
)
if resp.status_code in (200, 201, 202):
    return resp.json() if resp.content else {}
if resp.status_code in (429, 502, 503, 504):
    log.warning("Transient %s on %s, retry %s/%s",
               resp.status_code, path, attempt, self.MAX_
RETRIES)
    time.sleep(delay)
    delay *= 2
    continue
# Permanent failure – surface the validation envelope.
resp.raise_for_status()
raise RuntimeError(f"Delivery to {path} exhausted retries")

```

Listing E.2: Idempotent Delivery with Bounded Retries (Excerpt from `extraction_clients.py`)

The module ends with `run_signavio_delta()`, a compact orchestration example. It opens a delta run, streams published processes in batches of 50, and closes the run with its counters (success on the happy path or failed with whatever counters it gathered if delivery aborts). The code is dull, which was Chapter 11's whole point: ingestion should be boring.

E.5 The Exposure Service Skeleton

The file `spine_service.py` implements the exposure and AI areas of the contract as a FastAPI application. The lookup, context enrichment, and impact endpoints answer from a repository class whose SQL references the tables in [Appendix D](#). In the template, that repository returns the running scenario's demo data, so you can test the service end to end without a database. The comments mark where `hdbcli` against SAP HANA Cloud replaces the demo data, and where real XSUAA JWT validation replaces the development-only scope check.

The AI endpoints encode the design boundary in the API itself, as [Listing E.3](#) shows. A suggestion is created with status `generated`, and the code cannot set anything else. Only the `PATCH` endpoint, which demands the `spine.govern` scope and a `validatedBy` role, can move it to `accepted` or `rejected`.

```

@app.post("/ai/suggestions", status_code=201,
          dependencies=[Depends(require_scope("spine.ai.ground"))])
def create_suggestion(suggestion: AiSuggestion):
    suggestion_id = str(uuid.uuid4())
    SUGGESTIONS[suggestion_id] = {
        "suggestionId": suggestion_id,

```

```

        ** suggestion.model_dump(),
        "status": "generated",          # AI can never set 'accepted'
        "createdAt": dt.datetime.now(dt.timezone.utc).isoformat(),
    }
    return {"suggestionId": suggestion_id, "status": "generated"}

@app.patch("/ai/suggestions/{suggestion_id}",
           dependencies=[Depends(require_scope("spine.govern"))])
def decide_suggestion(suggestion_id: str, decision: SuggestionDecision):
    record = SUGGESTIONS.get(suggestion_id)
    if not record:
        raise HTTPException(404, "Unknown suggestion")
    if decision.status not in ("accepted", "rejected"):
        raise HTTPException(400, "status must be accepted or rejected")
    record.update(
        status=decision.status,
        validatedBy=decision.validatedBy,
        validatedAt=dt.datetime.now(dt.timezone.utc).isoformat(),
        decisionNote=decision.decisionNote,
    )
    # TEMPLATE: write AUDIT_EVENT with action 'approve'/'reject' and
    # the human actor (Chapter 16, Section 16.4).
    return record

```

Listing E.3: AI Proposes, Humans Decide, Enforced by Scopes (Excerpt from spine_service.py)

E.6 Adapting the Templates to Your Landscape

Work through the following points, in order, when you take the templates into your own subaccount:

1. Server URLs and tenants

Replace the `{subaccount}` and `{region}` variables in the contract's server and token URLs, and point the extraction clients at your SAP Signavio workspace, SAP LeanIX workspace, and WalkMe Insights tenant.

2. XSUAA binding

Create the four scopes in your `xs-security.json`, bind the service instance, and replace the development-only header check in `spine_service.py` with JWT validation against the XSUAA verification key (see Chapter 2, Section 2.7).

3. Persistence

Replace the in-memory repository with `hdbcli` against the SAP HANA Cloud tables created in [Appendix D](#) and the in-memory suggestion store with the `AI_SUGGESTION` and `AI_EXPLANATION` tables.

4. Field mappings

The SAP Signavio custom attributes (`processId`, `activityId`, `fioriAppId`, and so on) and the WalkMe custom data fields must match the naming conventions your organization adopted, as outlined in Chapter 3 and Chapter 8.

5. Audit wiring

The template marks where every human decision must write an `AUDIT_EVENT` naming the human actor (see Chapter 16, Section 16.4). Connect it before the first workflow goes live, not afterwards.

Extend the contract by adding organization-specific attributes to the schemas rather than inventing parallel endpoints. Version the file, since `v1` is part of the path, and treat contract changes with the same governance as data model changes. The [Appendix G](#) modules take this contract as given, so if you rename a field here, the rule dictionary and grounding filter samples must follow.

Appendix F

AI Grounding Rules

This appendix consolidates every grounding rule discussed in this book into one reference, covering the rules and constraints that govern how enterprise data is exposed to SAP AI and Joule. Each rule carries a stable identifier, a clear definition, the rationale behind it, and the chapter reference where it is introduced and justified. The rules are grouped by the chapters that own them. In the implementation, they live as records in the `RULE_DICTIONARY` and `AI_DATA_ELIGIBILITY` tables from [Appendix D](#), enforced by the grounding filter from [Appendix G](#). What follows is the human-readable contract behind that code.

F.1 How to Read These Rules

Rule identifiers follow the pattern `GR-<area>-<number>`, matching the `rule_id` values used in the code assets. Every rule is binding, and a grounding request that violates any of them is refused outright rather than degraded. Where a rule spans several chapters, the reference lists the section that introduces it first, followed by the sections that extend it.

[Figure F.1](#) shows the pipeline: every request passes all six rule areas in order, or is refused rather than degraded.

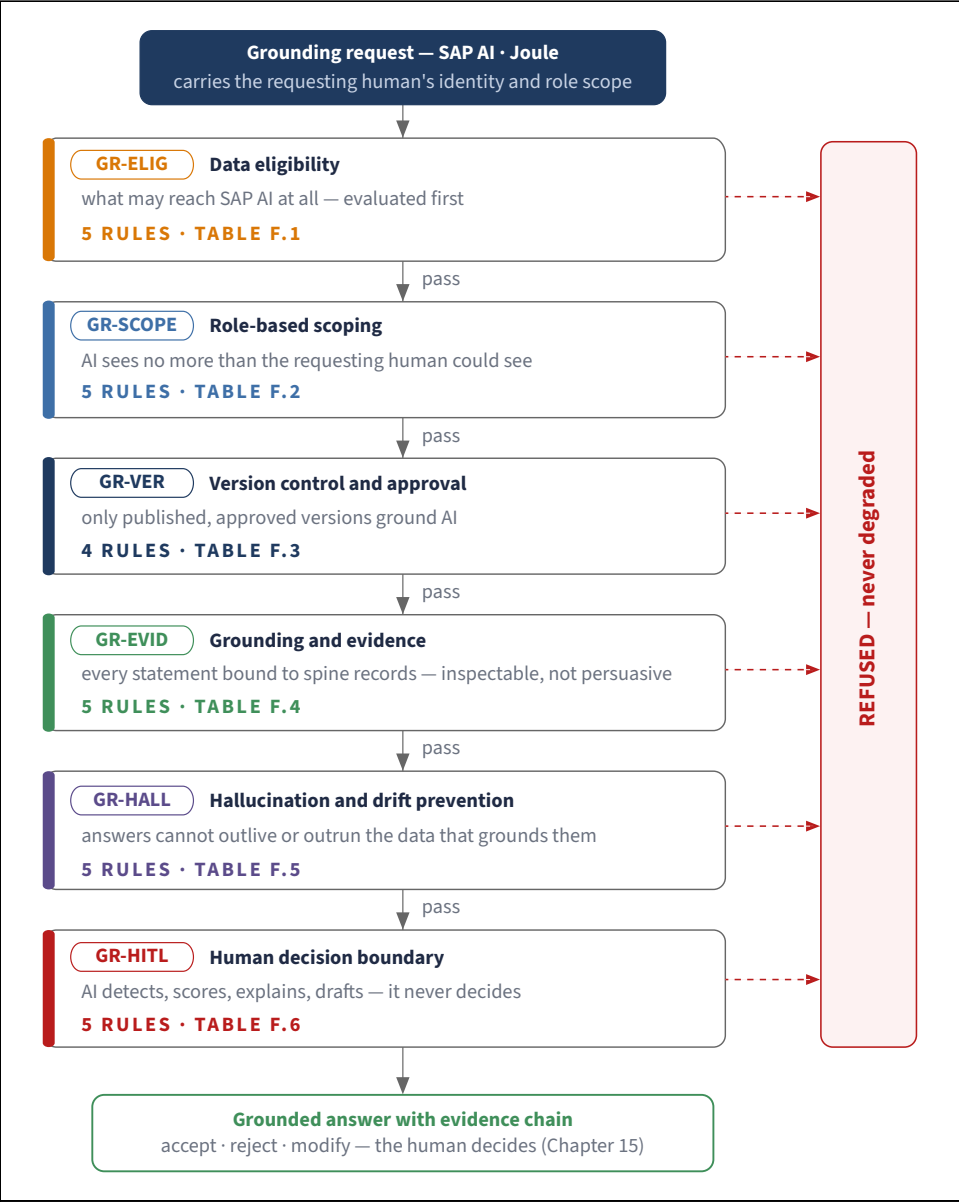


Figure F.1: The Grounding Rule Evaluation Pipeline—Every Request Passes All Six Rule Areas in Order or Is Refused

F.2 Data Eligibility Rules

Eligibility rules (see [Table F.1](#)) decide what may reach SAP AI, and they are evaluated before any other rule. Data that is not eligible is never role-scoped, versioned, or explained, because it never leaves the data spine in the first place.

Rule ID	Rule	Rationale	Chapter Reference
GR-ELIG-001	Only entities with an explicit AI_DATA_ELIGIBILITY record may ground AI; ungoverned entities are refused by default.	Deny-by-default prevents new tables silently becoming AI-visible.	Chapter 14, Section 14.1
GR-ELIG-002	Within an eligible entity, only whitelisted fields are exposed; all other fields are stripped before the payload leaves the data spine.	Field-level control stops internal notes and costing detail leaking into prompts.	Chapter 14, Section 14.1
GR-ELIG-003	Operational master data is referenced by identifier, never replicated into AI context in full.	Referencing keeps the systems of record authoritative and shrinks exposure.	Chapter 9, Section 9.3 and Chapter 14, Section 14.1
GR-ELIG-004	Data classified above the AI service's clearance (for example, confidential payloads to a shared model endpoint) is never eligible, regardless of role.	Classification is a ceiling that roles cannot raise.	Chapter 2, Section 2.8 and Chapter 14, Section 14.1
GR-ELIG-005	Adoption telemetry is eligible only in role-aggregated form; events carrying user identity are never eligible.	Protects individuals; adoption signals are about friction, not people.	Chapter 8, Section 8.1 and Chapter 14, Section 14.1

Table F.1: Data Eligibility Rules

F.3 Role-Based Scoping Rules

Scoping rules, shown in [Table F.2](#), make sure AI sees no more than the requesting human could see. The AI service inherits the caller's scope and never carries a broader one of its own.

Rule ID	Rule	Rationale	Chapter Reference
GR-SCOPE-001	Every grounding request executes under the requesting user's roles; there is no privileged AI service role for data access.	AI must never become a side door around authorization.	Chapter 2, Section 2.8 and Chapter 14, Section 14.2
GR-SCOPE-002	The role scope of an eligibility record is an allow-list; a caller holding none of the listed roles is refused.	Positive authorization is auditable; exclusion lists drift.	Chapter 14, Section 14.2
GR-SCOPE-003	Cross-domain evidence sets are filtered per record, not per request; one authorized record does not authorize its neighbors.	Prevents scope creep through evidence bundling.	Chapter 12, Section 12.2 and Chapter 14, Section 14.2
GR-SCOPE-004	Joule surfaces AI insights only within the user's SAP Build Work Zone role context.	The interaction layer respects the same boundary as the data layer.	Chapter 13, Section 13.4 and Chapter 17, Section 17.3
GR-SCOPE-005	Separation of duties (SoD) conflicts detected in the caller's roles block grounding on control-relevant data.	AI must not help a conflicted user reason across their conflict.	Chapter 2, Section 2.8 and Chapter 19, Section 19.1

Table F.2: Role-Based Scoping Rules

F.4 Version Control and Approval Rules

Version rules make grounding reproducible. Eligibility and scoping decide what AI may see; the rules shown in [Table F.3](#) fix which version AI saw, and keep a record.

Rule ID	Rule	Rationale	Chapter Reference
GR-VER-001	Only published process and architecture content grounds AI; draft and approved-but-unpublished content is refused.	AI reasoning over unstable content produces unstable suggestions.	Chapter 3, Section 3.5 and Chapter 14, Section 14.3
GR-VER-002	A grounding bundle is pinned to explicit version identifiers and is immutable once assembled.	Reproducibility: The same question over the same bundle yields comparable evidence.	Chapter 9, Section 9.4 and Chapter 14, Section 14.3
GR-VER-003	When content is retired or superseded, dependent grounding bundles are invalidated, not silently re-pointed.	Silent re-pointing breaks the evidence chain behind past suggestions.	Chapter 9, Section 9.4 and Chapter 18, Section 18.3
GR-VER-004	Changes to eligibility or scoping rules take effect only through the governance workflow, with approval recorded in <code>AUDIT_EVENT</code> .	The rules governing AI are themselves governed data.	Chapter 10, Section 10.5 and Chapter 16, Section 16.4

Table F.3: Version Control and Approval Rules

F.5 Grounding and Evidence Rules

The rules shown in [Table F.4](#) tie every AI statement back to a data spine record. They are what makes a suggestion something you can inspect rather than something you must trust blindly.

Rule ID	Rule	Rationale	Chapter Reference
GR-EVID-001	Every AI suggestion must reference at least one data spine record as evidence; a suggestion without evidence references is refused at registration.	No evidence, no suggestion—that is the primary hallucination control.	Chapter 14, Section 14.4 and Chapter 15, Section 15.4
GR-EVID-002	Evidence references use canonical data spine identifiers (entity and key), so every claim can be traced back to a governed record.	Traceability must survive model and prompt changes.	Chapter 9, Section 9.3 and Chapter 15, Section 15.4
GR-EVID-003	Each suggestion carries an explanation record: evidence references, rules applied, reasoning summary, model reference, and prompt version.	Explainability is a stored artifact, not an on-demand narrative.	Chapter 15, Section 15.4
GR-EVID-004	Excluded evidence is recorded as excluded with its reason; it is never silently dropped from the explanation.	Reviewers must see what AI was not allowed to see.	Chapter 14, Section 14.2 and Chapter 15, Section 15.4
GR-EVID-005	Confidence scores are computed from the rule and evaluator outputs, never asserted by the language model itself.	The model’s self-assessed confidence is not evidence.	Chapter 15, Section 15.3 and Section 15.4

Table F.4: Grounding and Evidence Rules

F.6 Hallucination and Drift Prevention Rules

The rules in previous sections govern what AI is given. This group of rules, shown in [Table F.5](#), governs what AI is allowed to say. Hallucination is treated as an architectural failure to be prevented rather than a defect to be caught after the fact. The same discipline extends to drift.

Rule ID	Rule	Rationale	Chapter Reference
GR-HALL-001	AI answers about processes, systems, and links are constrained to the grounding bundle; general model knowledge may explain but never assert enterprise facts.	Enterprise truth lives in the data spine, not in model weights.	Chapter 14, Section 14.5
GR-HALL-002	Identifiers in AI output are validated against the data spine before display; an identifier that does not resolve blocks the response.	Fabricated identifiers are the most damaging hallucination class.	Chapter 14, Section 14.5 and Chapter 17, Section 17.3
GR-HALL-003	Prompt templates are versioned artifacts; drift between prompt versions is tracked and material changes can reapprove through governance.	Prompt drift is model drift by another route.	Chapter 13, Section 13.3 and Chapter 14, Section 14.5
GR-HALL-004	Suggestion quality is monitored against human decisions; sustained divergence triggers a review of the model, prompts, and rules.	Acceptance rate drift is the operational drift signal.	Chapter 14, Section 14.5 and Chapter 18, Section 18.1
GR-HALL-005	Unauthorized inference is treated as leakage: AI must not be asked to derive data a caller could not read directly (for example, inferring salary bands from position data).	Scoping applies to conclusions, not just the inputs.	Chapter 14, Section 14.2 and Section 14.5

Table F.5: Hallucination and Drift Prevention Rules

F.7 Human Decision Boundary Rules

The final group, shown in [Table F.6](#), restates the boundary this book is built on: AI is a detector, a scorer, an explainer, and a drafter of suggestions. It is never the decider.

Rule ID	Rule	Rationale	Chapter Reference
GR-HITL-001	AI services may create suggestions in status generated only; the accepted and rejected statuses are writable exclusively by human validators.	The decision right is structural, enforced in the API, not by convention.	Chapter 1, Section 1.5 and Chapter 15, Section 15.5
GR-HITL-002	Every acceptance or rejection records the human role, timestamp, and decision note in <code>AUDIT_EVENT</code> .	Accountability for decisions stays with humans.	Chapter 15, Section 15.5 and Chapter 16, Section 16.4
GR-HITL-003	AI-assisted workflows in SAP Build Process Automation route through human approval steps for any change to governed data.	Automation accelerates the path to a decision, never past it.	Chapter 16, Section 16.1 and Section 16.3
GR-HITL-004	AI must not trigger write operations against systems of record, directly or through chained services.	Write authority and suggestion authority should never combine.	Chapter 1, Section 1.5 and Chapter 2, Section 2.9
GR-HITL-005	Rejected suggestions are retained with their explanations for model and rule improvement; rejection is a signal, not a deletion.	The system learns from what humans turned down.	Chapter 15, Section 15.5 and Chapter 18, Section 18.2

Table F.6: Human Decision Boundary Rules

These twenty-nine rules are the contract between the integrated landscape and its AI layer. They are restrictive by design. Every relaxation of a rule in this appendix widens what AI can see, say, or set in motion, and that should be treated as an architectural decision carrying the same governance weight as a change to the data spine itself.

Appendix G

Python Libraries and Code Assets

This appendix documents the downloadable code assets that implement the book's concepts as working, first-version Python modules. [Appendix D](#) defined the canonical data model and [Appendix E](#) the API contract and service templates; the assets in this section supply the intelligence in between. That includes the governed rule engine, the fuzzy evaluator for uncertain cross-system links, the semantic graph that answers impact and context questions, and the grounding filter that decides what AI may see. We use the same running scenario we have used throughout the book, illustrating the link between a cost center governed in SAP Master Data Governance and a position in SAP SuccessFactors.

The design boundary that governs every module bears repeating: AI in this codebase is a detector, a scorer, an explainer, and a drafter, never a decider. Every suggestion is registered with status `generated` and a full explanation record, and only a human validator can move it to `accepted` or `rejected` (see Chapter 15, Section 15.5).

G.1 Package Contents

The download unpacks into a `modules/` directory with five Python files, a `json_samples/` directory with data spine-shaped sample data, a `requirements.txt` file, and a `README.md` file. [Table G.1](#) maps each asset to its purpose and its chapter grounding, and [Figure G.1](#) shows how the modules cooperate at runtime.

Asset	Purpose	Reference
<code>modules/rule_dictionary.py</code>	Governed rule engine over <code>rules.json</code> : structural, semantic, fuzzy, and grounding rules as data, not code	Chapter 11, Section 11.5 and Chapter 14, Section 14.4
<code>modules/fuzzy_evaluator.py</code>	Weighted fuzzy scoring of cross-system links where exact keys are unavailable	Chapter 12, Section 12.3 and Chapter 15, Section 15.2
<code>modules/semantic_graph.py</code>	Data spine graph over the Appendix D model: impact, context, and orphan checks	Chapter 12, Section 12.3 and Chapter 14, Section 14.4

Table G.1: Assets of the [Appendix G](#) Download Package

Asset	Purpose	Reference
<i>modules/grounding_filter.py</i>	AI data eligibility enforcement and explainability records	Chapter 15, Section 15.4
<i>modules/position_cost_centre_example.py</i>	End-to-end running scenario wiring all modules together	Chapter 11 through Chapter 16
<i>json_samples/*.json</i>	Data spine-shaped payloads, the rule dictionary, and eligibility rules	<u>Appendix D</u>

Table G.1: Assets of the Appendix G Download Package (Cont.)

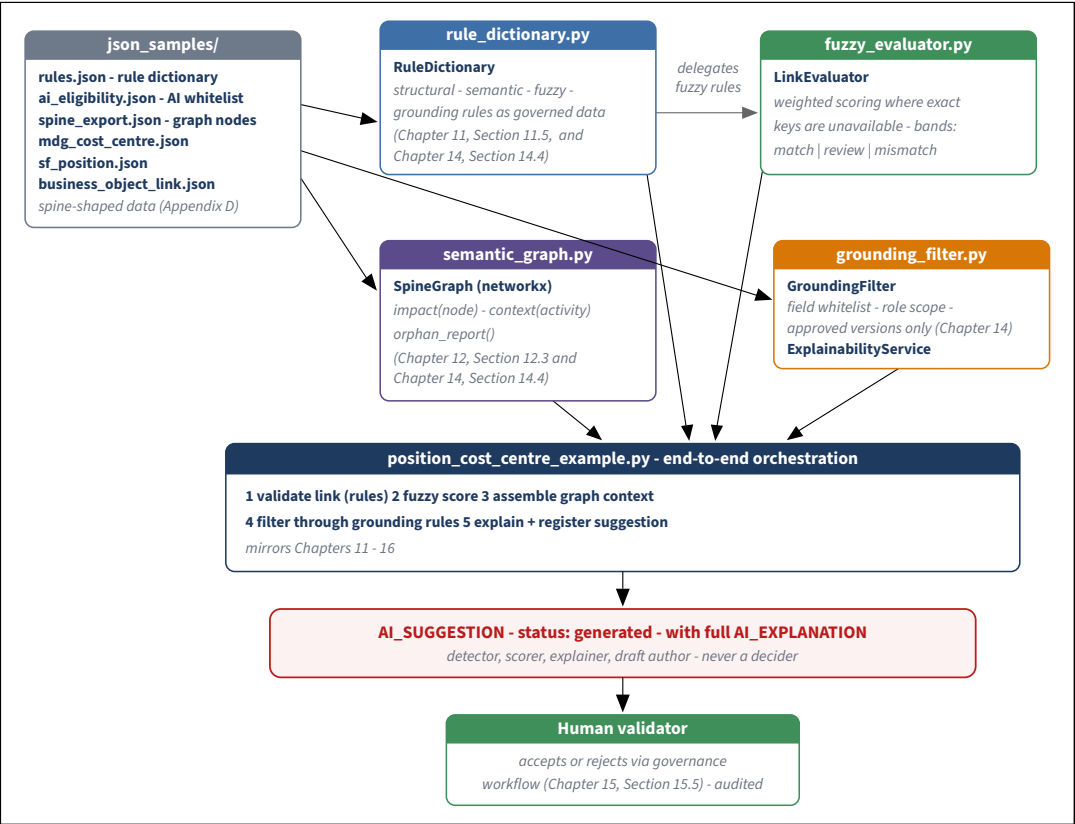


Figure G.1: How the Appendix G Modules Cooperate: Governed Rules and Samples Feed the Evaluators; Every Run Ends in a Suggestion Awaiting a Human Decision

G.2 Prerequisites and Installation

The modules run on Python 3.10 or later. [Listing G.1](#) shows the dependency set; the commented entries are optional SAP BTP libraries that become relevant only when you deploy the service from [Appendix E](#) to the SAP BTP, Cloud Foundry environment.

```
# Appendix G – Python dependencies for the book's code assets
# Core evaluation and graph processing
rapidfuzz>=3.6          # fuzzy-logic evaluators (Chapters 12, 15)
networkx>=3.2          # semantic graph processor (Chapter 12, Section
12.3; Chapter 14, Section 14.4)
pydantic>=2.5          # payload validation for API templates

# Appendix E service and client templates
fastapi>=0.110         # data spine exposure service skeleton
uvicorn>=0.27          # ASGI server for local runs
requests>=2.31         # extraction clients (Signavio, LeanIX, WalkMe)

# SAP BTP connectivity
hdbcli>=2.19           # SAP HANA Cloud driver for the spine tables
# cfenv>=0.5           # optional: Cloud Foundry service binding
parsing
# sap-xssec>=4.0       # optional: XSUAA JWT validation on BTP

# Analytics and data preparation
pandas>=2.1           # adoption signal aggregation (Chapter 8)
```

Listing G.1: Python Dependencies of the Code Assets (requirements.txt)

Running the modules locally needs nothing beyond `pip install -r requirements.txt`. Executing the book's ideas end to end is another matter, since that is a landscape exercise rather than a laptop one. [Table G.2](#) lists the platform components the full picture assumes.

Component	Role in the Architecture
Subaccount in SAP BTP, Cloud Foundry runtime	Hosts the exposure service from Appendix E (Python buildpack)
SAP HANA Cloud	Persists the canonical data spine tables from Appendix D
SAP Integration Suite	Runs the governed ingestion flows from Chapter 11

Table G.2: Platform Prerequisites for the End-to-End Scenario

Component	Role in the Architecture
XSUAA instance	Authentication and the four API scopes from Chapter 2, Section 2.7
SAP AI Core/SAP AI Launchpad	Generative AI hub access for grounded suggestions from Chapter 13
SAP Build Process Automation	Approval workflows in which humans decide, outlined in Chapter 16
SAP Signavio, SAP LeanIX, and WalkMe Insights	Tenant API access for the extraction clients, covered in Appendix E

Table G.2: Platform Prerequisites for the End-to-End Scenario (Cont.)

The quick start is short. Install the requirements, change into `modules/`, and run `python position_cost_centre_example.py`. [Section G.6](#) walks through what you should see.

G.3 The Rule Dictionary

The module `rule_dictionary.py` loads governed rules from `rules.json`, the file-based counterpart of the `RULE_DICTIONARY` table in [Appendix D](#), and evaluates them against data spine records. The design is about governance rather than cleverness. Rules are data, so adding one is a governance act with an owner, a severity, and a chapter reference from this book behind it, not a code deployment. The engine dispatches on `rule_type`. Structural rules check required fields and formats, semantic rules check cross-field and cross-system consistency, and fuzzy rules delegate to the evaluator, covered in [Section G.4](#). [Listing G.2](#) shows the running scenario's fuzzy rule as it appears in the sample dictionary.

```
{
  "rule_id": "RULE-LINK-CC-POS-001",
  "rule_name": "Cost center to position fuzzy correspondence",
  "rule_type": "fuzzy",
  "applies_to": "BUSINESS_OBJECT_LINK",
  "severity": "warning",
  "lifecycle_state": "active",
  "chapter_reference": "Chapter 15, Section 15.2",
  "expression": {
    "evaluator": "LinkEvaluator",
    "thresholds": { "match": 0.85, "review": 0.60 }
  }
}
```

```
    }  
  ]  
}
```

Listing G.2: A Governed Fuzzy Rule: Data with an Owner and a Chapter Reference, Not Code (Excerpt from rules.json)

G.4 The Fuzzy Evaluator

Exact keys are the data spine's preferred currency, but the world does not always oblige. A cost center and a position may need to be linked where no clean foreign key exists on either side. The module `fuzzy_evaluator.py` handles that case with a weighted evaluation across four factors, each scored between 0 and 1. [Table G.3](#) lists them with their default weights, which are meant to be tuned per landscape.

Factor	Weight	Evidence Considered
name_similarity	0.35	Token-sorted similarity of the cost center description and the position's department description
org_alignment	0.25	Exact company code match between the two objects
hierarchy_alignment	0.20	Partial-ratio similarity of cost center group and position org unit; unknown data scores a neutral 0.5
validity_overlap	0.20	Whether the validity windows of the two objects intersect

Table G.3: Factors of the Link Evaluator and Their Default Weights

The weighted score lands in one of three bands: *match* at 0.85 and above, *review* at 0.60 and above, and *mismatch* below that. Alongside the score the evaluator returns the per-factor contributions and a human-readable justification naming the strongest and weakest evidence, which is the raw material for the explanation record covered in [Section G.5](#). Note what the evaluator does not do: It never writes an accepted link status. It scores, and a human steward decides through the governance workflow.

G.5 The Semantic Graph

The module `semantic_graph.py` builds a directed graph over the data spine. Nodes are typed records such as `PROCESS: P-FIN-CC-001` or `APPLICATION: APP-MDG-001`, and edges carry the relationship semantics of the model from [Appendix D](#), meaning `contains`, `executes_in`, `triggers`, `governed_by`, and `guided_by`. The graph answers the three questions the execution and AI services keep asking. `impact(node)` walks downstream from a

changed application or interface to everything that depends on it (see Chapter 12, Section 12.3). `context(activity)` assembles the full semantic neighborhood of an activity, which is the evidence set AI grounding is allowed to draw on (see Chapter 14, Section 14.4). `orphan_report()` surfaces the data spine's loose ends, such as activities with no application mapping or links with no validating rule, implementing the consistency checks covered in Chapter 11, Section 11.5. The graph loads from a data spine JSON export (*spine_export.json* in the samples), so it works the same way against an SAP HANA export in production and against the samples on a laptop.

G.6 Grounding and Explainability

The module `grounding_filter.py` contains the two components that make AI consumption governable. `GroundingFilter` enforces the `AI_DATA_ELIGIBILITY` rules of Chapter 14 before any record reaches SAP AI Core or the generative AI hub. Entities without an eligibility rule are never exposed. Callers without a whitelisted role are refused. Records that are not published, or not approved where the version constraint says so, are excluded. Of the records that pass, only whitelisted fields survive. Nothing is just dropped: an excluded record stays in the evidence set, marked excluded, with the reason attached. [Listing G.3](#) shows the version-constraint and whitelisting core.

```
# Version constraint (Chapter 14, Section 14.3)
constraint = rule.get("version_constraint")
if constraint == "published_only" and record.get("lifecycle_state")
!= "published":
    raise GroundingViolation(
        f"{entity_name} record {record.get('id', '?')} is not "
        "published; draft data never grounds AI."
    )
if constraint == "approved_only" and record.get("approval_status")
!= "approved":
    raise GroundingViolation(
        f"{entity_name} record is not approved."
    )

# Field whitelisting (Chapter 14, Section 14.1)
whitelisted = set(rule["field_scope"])
return {k: v for k, v in record.items() if k in whitelisted}
```

Listing G.3: Version Constraints and Field Whitelisting: Draft Data Never Grounds AI (Excerpt from `grounding_filter.py`)

`ExplainabilityService` builds the `AI_EXPLANATION` record that every suggestion has to carry (see Chapter 15, Section 15.4), holding evidence references back to data spine

records, the rules applied, a reasoning summary, and the model and prompt version. Its one hard rule is that an explanation with no evidence references is refused outright. A suggestion with no data spine evidence behind it is a hallucination risk, and it must not be registered.

G.7 The Running Scenario End to End

The script `position_cost_centre_example.py` wires everything together, mirroring the arc of Chapter 11 to Chapter 16 in five steps. It loads the SAP Master Data Governance cost center 10010010 and the SAP SuccessFactors position 70001234 from the samples. It validates the candidate link with the rule dictionary, structurally, semantically, and fuzzily. It assembles the semantic context of the owning activity A-FIN-CC-001-030 from the data spine graph. It filters that context through the grounding rules as the role `master_data_steward`. Finally it builds an explanation and registers an AI suggestion of type `mapping_fix`, with status `generated` and confidence equal to the fuzzy score, awaiting a human steward's decision.

Run it with `python position_cost_centre_example.py`. The output prints each stage: the rule outcomes with their chapter references, the fuzzy score with its band and justification, the grounded evidence set including anything excluded and why, and the final suggestion payload. Change `sf_position.json` to a different company code or a non-overlapping validity window, and you can watch the score fall through the review band and the justification change with it. It is a useful five-minute experiment before you point the modules at real data.

From here, the path to production runs through the other appendices. Create the tables from [Appendix D](#), deploy the contract and service from [Appendix E](#), replace the JSON samples with reads from the data spine, and register suggestions through `POST /ai/suggestions`. The schema, the scopes, and the workflow all say the same thing: AI proposes, humans decide.

Appendix H

Reference Integration Diagrams

“A reference architecture earns its place not by adding detail, but by making the whole landscape legible at once—every system, every boundary, every governed flow, on one page a practitioner can trust.”

This appendix is the consolidated visual reference for the integrated landscape described across the book. It does not reargue the architecture, since the reasoning behind every element has already been covered at depth in the book. These are the plates you, the practitioner, can reference afterwards. The first plate shows the whole governed landscape, and three further plates isolate the paths the book treats separately (here we have also provided the chapters that discuss each plate in further detail):

- **The autonomous enterprise**

This is the synthesis plate (see Chapter 2).

- **The execution path**

This plate visualizes process and architecture data becoming executable (see Chapter 11 and Chapter 12).

- **The adoption path**

This plate visualizes the observed WalkMe telemetry without corrupting canonical truth (see Chapter 7 and Chapter 8).

- **The AI-consumption path**

This plate visualizes the only route by which AI may touch the landscape (see Chapter 13 through Chapter 15).

Read the synthesis plate first; the three path plates expand regions of it. An animated walk-through of the synthesis plate is available in the online supplements, alongside the downloadable code.

H.1 How to Read These Plates

Every plate uses one shared notation. Color is used to encode the role rather than the system. Violet represents the governed grounding chain, gold is audit and hash-chained lineage, and navy is the canonical data spine. A gray connector with a pill label is used to show a governed flow, and the label tells you the control that the flow is subject to. A dark band states a non-negotiable principle or boundary. Boxes are typed objects or services, and a flow crosses a boundary only through a labeled, governed connector.

The systems-of-record key is fixed throughout. Each system owns exactly what is listed below and nothing more, and the plates show how those owners are joined by stable identifiers on SAP BTP rather than by ad hoc copies of one another's data:

- **SAP Signavio**
Process semantics at L4/L5: activities, roles, control intent, application references, and variants
- **SAP LeanIX**
Enterprise structure: applications, interfaces, dependency chains, and lifecycle state
- **WalkMe**
Process-aware adoption content and usage telemetry, mapped to process, activity, and SAP Fiori app IDs
- **SAP BTP**
The canonical data spine, governed APIs, and execution services: the execution fabric.
- **SAP AI**
SAP AI Core, the generative AI hub, and Joule: inference over validated grounding bundles only
- **SAP Build**
SAP Build Process Automation (human-in-the-loop workflows) and SAP Build Work Zone (process-driven UX)

H.2 Plate 1: The Autonomous Enterprise (Synthesis)

The capstone plate is shown in [Figure H.1](#). It assembles every contribution of the book into one governed picture: the systems of record, the governed ingestion and validation layer, the canonical data spine, the governed API surfaces, the AI consumption tier with its cross-vendor validation funnel, and the identity and audit spine running the full height of the landscape. The takeaway is structural. Everything reaches everything else only through the data spine, under explicit version and audit, and AI augments governance without holding authority. Chapter 2 argues this plate; the three plates that follow magnify its execution, adoption, and AI-consumption regions.

An animated walkthrough of this plate is available in the online supplements, alongside the downloadable code.

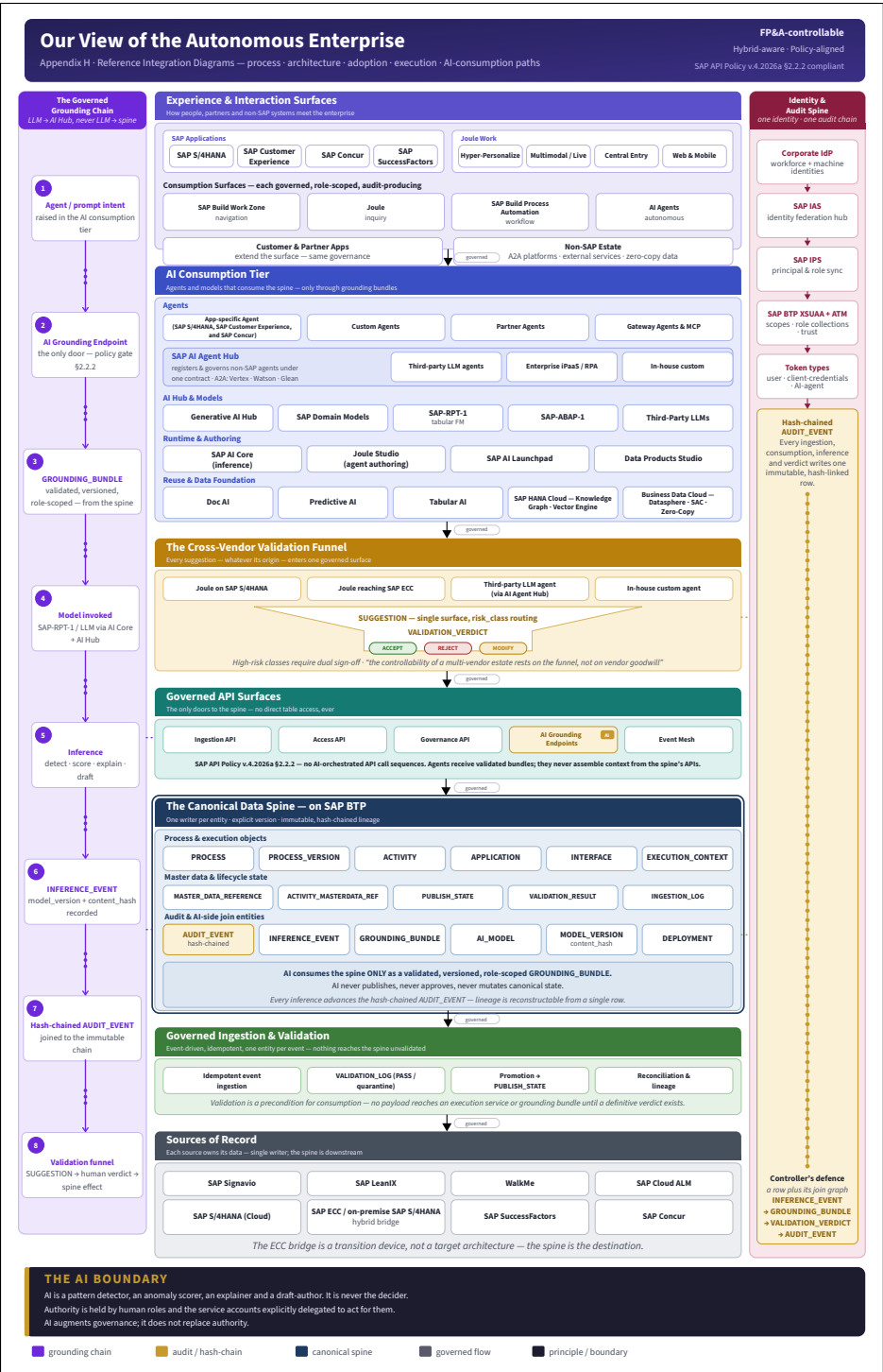


Figure H.1: The Autonomous Enterprise—The Consolidated Reference Integration Architecture, Extended from the SAP Business AI Platform Reference Into the Governed-Spine View

H.3 Plate 2: The Execution Path

The plate in [Figure H.2](#) traces how process and architecture data become executable. SAP Signavio and SAP LeanIX are bound by stable identifiers, ingested through a governed validation layer where nothing reaches the data spine unvalidated, and resolved onto the canonical data spine with one writer per entity and immutable, hash-chained lineage. Process-aware execution services then run on top of the validated data spine, covering process lookup, context enrichment, and change-impact resolution, and SAP Build Process Automation and SAP Build Work Zone consume them. The takeaway is that validation is a precondition for consumption. No execution service ever runs on unvalidated canonical state. This was argued in Chapter 11 and Chapter 12.

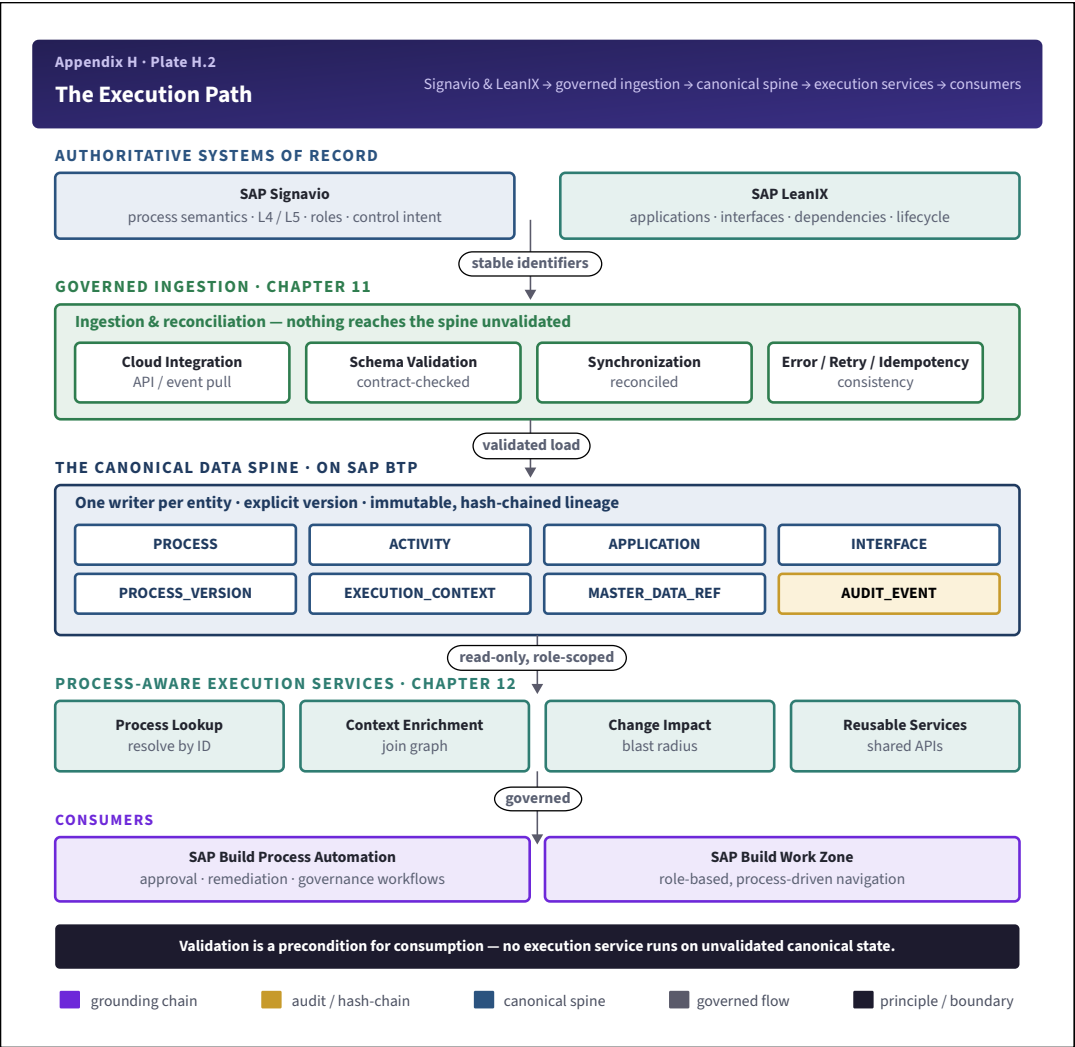


Figure H.2: The Execution Path—Systems of Record through Governed Ingestion Into the Canonical Data Spine, Consumed by Process-Aware Execution Services and SAP Build

H.4 Plate 3: The Adoption Path

The plate in [Figure H.3](#) shows how adoption is observed without letting it corrupt canonical truth. WalkMe content is mapped downstream to L4/L5 process steps and SAP Fiori app IDs. User-interaction signals—such as completion, friction, drop-off, and unmet needs—are captured and linked to the data spine on process, activity, and SAP Fiori app IDs. Friction, deviation, and training-gap detection then feed two closed loops, one carrying evidence into process governance in SAP Signavio and the other carrying an adoption signal into downstream AI services. The takeaway is that adoption telemetry is a governed input. It informs governance and AI, and it never writes canonical state. This was argued in Chapter 7 and Chapter 8.

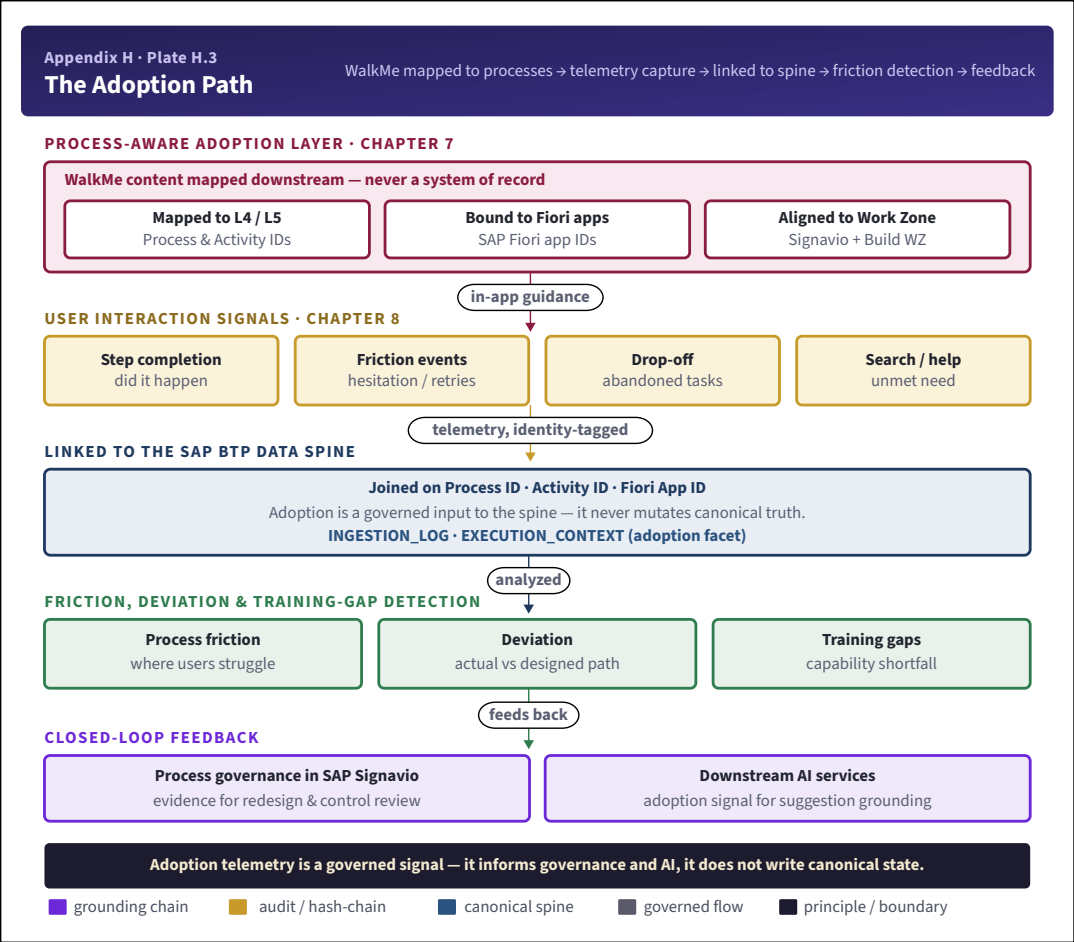


Figure H.3: The Adoption Path—WalkMe Mapped to Processes, Telemetry Linked to the Data Spine, Friction and Deviation Fed Back to Governance and AI

H.5 Plate 4: The AI-Consumption Path

The plate in [Figure H.4](#) traces the only route by which AI is permitted to touch the landscape. The data spine is never read by AI directly.

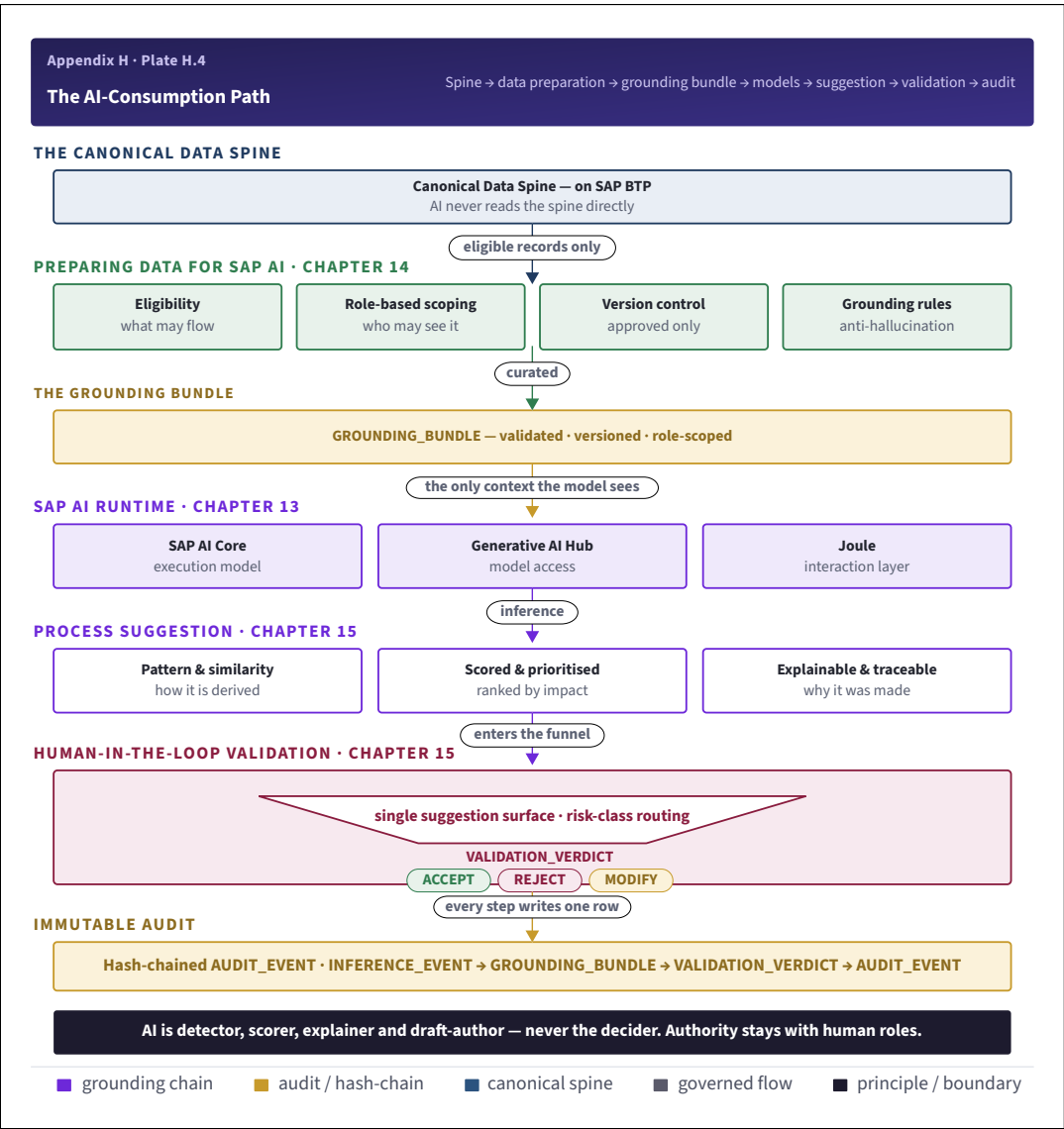
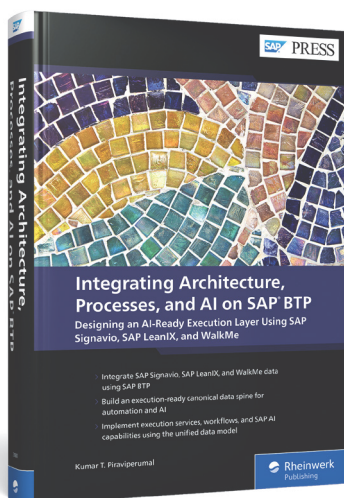


Figure H.4: The AI-Consumption Path—The Data Spine Through Data Preparation Into a Validated Grounding Bundle, Gated by Human Validation and Written to a Hash-Chained Audit

Eligible records pass through data preparation (including eligibility, role-based scoping, version control, and grounding rules) and emerge as a validated, versioned, role-scoped grounding bundle, which is the only context the model ever sees. SAP AI Core, the generative AI hub, and Joule produce a scored, explainable process suggestion, which then enters

a human-in-the-loop validation funnel and is either accepted, rejected, or modified. Every step writes one immutable, hash-chained audit row. The takeaway is this book's boundary principle: AI is detector, scorer, explainer, and drafter, never the decider. This was argued in Chapter 13, Chapter 14, and Chapter 15.



Integrating Architecture, Processes, and AI on SAP BTP

Designing an AI-Ready Execution Layer Using SAP Signavio, SAP LeanIX, and WalkMe

Planning to implement AI and automation? Start by preparing your foundation. Most organizations document processes and architecture, but don't connect them in a way that's useful for AI models and automation workflows. This book shows you how to use SAP Signavio, SAP LeanIX, and WalkMe to transform your architecture and process data into a unified execution layer on SAP BTP. You'll learn to model processes, map applications to process steps, capture adoption telemetry, design integration flows, and prepare data for AI and automation. Move from static documentation to an execution-ready SAP architecture!

711 pages, pub. 08/2026

E-Book: \$94.99 | Print: \$99.95 | Bundle: \$109.99

www.sap-press.com/6352



The Author

Kumar Piraviperumal has worked with SAP technologies since 1997, building a career that spans enterprise delivery, architecture leadership, and professional education. Over nearly three decades, his work has focused on how large organizations design, integrate, and operate complex SAP landscapes under real-world constraints. He currently leads a global SAP business transformation practice at Cognitus, an IBM Company operating across more than 19 countries.