SADAR™ Quick Reference

V1.0 March, 2026

Purpose

This document is a concise reference to the features and capabilities of the SADAR standard. Each section identifies the feature, its design rationale, and its key behaviors. For conceptual background, positioning, and implementation guidance, refer to the companion documents below.

Document	Description
SADAR Executive Overview	Frames the problem space and value proposition of SADAR at an executive level.
SADAR Concepts	Foundational concepts of registry contents with examples.
SADAR Overview	Detailed discussion of the problem space; how SADAR relates to MCP, A2A, and other initiatives; mid-level usage descriptions.
SADAR Registry Security Architecture	How entities and agents authenticate to SADAR; protection and exchange of sensitive information (payment, license keys); manifest digital signing.
SADAR Registry Distribution	The registry-of-registries model; federation recognition; replication, query forwarding, and terms of service. Includes private registry support for internal use.
SADAR NFR Schema Design	Detailed definition of Non-Functional Requirement types and their associated schema.
SADAR Entity Specification	Attributes and structure of entities — publishers and consumers of registry entries.
Trustworthy AI and SADAR	Maps SADAR capabilities to a Trustworthy AI framework.
SADAR Feature (Long Form)	Detailed feature descriptions with design rationale and implementation context.
SADAR Architect Getting Started	Risk-based, incremental adoption guidance for architects.
SADAR CIO Getting Started	Executive-level adoption paths with rapid value proposition framing.

Semantic Registry

The registry stores machine-readable descriptions of capabilities, data, and business processes, grounded in authoritative industry standards to eliminate ambiguity at the point of discovery.

- Entries describe agents, tools, resources, APIs, and multi-step business process definitions in machine-readable form
- Each entry is tagged with an authoritative standard such as **NAICS** (industry classification), **APQC PCF** (process framework), **HL7**, **X12** (transaction standards), or existing API definitions
- Grounding concretely defines data meaning and syntax within the context of the business process
- Registries may be public or private; authorized public registries may participate in federation for replication and query forwarding
- Private registries may operate internally without participation in the public directory
- Registries are only accessed at discovery time – Initial discovery and discovery TTL expiration. No sensitive information is held by the registry nor is it part of the runtime process beyond discovery.

Entity

The entity is the organizational unit that publishes or consumes registry entries. Entities may be arranged into a hierarchy with a single root and multiple branches. The root must represent a legal entity accountable for all content published under it.

- Entities have their own signed manifests defining identity, OIDC endpoints, JWKS, payment methods, and default handlers
- Entity manifests are maintained by authorized entity administrators
- The entity hierarchy establishes a legally traceable accountability chain — every registry entry traces to an accountable root
- Root entities may delegate administration of sub-entities, enabling least-privilege governance across organizational boundaries, legal jurisdictions, and operational teams
- The entity hierarchy provides namespacing — agents, tools, and resources with identical names may exist under different entities without collision, which is essential for correct disambiguation in federated registries
- All registry entities, with the exception of the root, shall have one, and only one, parent entity
- All registry entries (e.g. agents, tools, resources, processes, etc.) shall have one, and only one, entity
- An entity may be defined as “Enity-Only” meaning it is not discoverable outside of itself. Other values define discoverability as:
    - Entity-Only: Only discoverable by agents within this entity
    - Entity-Peers: Discoverable across entities, within the same root, that share the same parent entity
    - Entity-Tree: Discoverable within the entity and any of the entity’s children
    - Root-Wide: Discoverable within all entities defined under the common root
    - Public: Discoverable within the registry
    - Federated: Eligible to be replicated and/or discovered via forwarding

Manifests

Every registry entry is backed by a publisher-signed manifest — the authoritative, tamper-evident record of what a capability is, what it requires, and what it guarantees.

- Every entry has a publisher-signed manifest; the signature verifies publisher identity and manifest integrity
- Manifests are immutable — any change produces a new versioned entry with a new manifest and signature
- **Manifests carry machine-readable declarations across seven dimensions:**
    - Business process — grounding standard, prerequisites, required sequencing
    - Data — meaning, syntax, sovereignty constraints

  - Compliance — frameworks, certifications, posture requirements
  - Licensing — terms, restrictions, first-use requirements
  - Costs — pricing model, payment endpoints
  - Operational — rate limits, SLA, NFRs
  - Provider — identity, OIDC endpoint, JWKS

Bilateral Discovery

Discovery is deterministic, not probabilistic. The registry matches requester requirements against provider requirements and returns only compliant candidates eliminating LLM guesswork in capability selection.

- Requester manifest and provider manifest are matched bilaterally — both sides declare requirements; both must be satisfied
- Provider requirements prevent discovery by non-qualifying requestors — providers have a built-in right-of-refusal
- Only candidate matches are returned; the requesting agent or its organization-supplied handler makes the final selection
- Enforces least-privilege access to the registry — entries not matching requester requirements are not visible
- Eliminates LLM probabilistic errors in tool and agent selection by making capability matching deterministic
- **Discovery enforces compatibility across:**
  - Business process context and intent (e.g., AP Invoice vs. AR Invoice)
  - Data meaning and syntax
  - Compliance framework posture
  - Licensing and operational requirements
  - Allow/deny lists
  - Industry scoping where applicable

Compliance Posture Matching

Both requestors and providers declare the regulatory and compliance frameworks under which they operate; discovery enforces compatibility before any interaction occurs.

- Requestors declare applicable compliance frameworks in their manifests (e.g., HIPAA, GDPR, SOC2, FedRAMP, ISO 27001)
- Providers configure the compliance posture requirements requestors must assert to receive discovery results
- Matching is bilateral: a requestor can require compliant providers; a provider can restrict visibility to requestors asserting a compatible posture — *compliance alignment is verified at discovery, not assumed after invocation*
- Licensing metadata may specify that first use of a compliance-scoped capability requires human review or out-of-band verification before automated provisioning

- Requestors and Consumers may expose references to documentation and/or proof of compliance. Certain documents such as actual audit results may require authenticated access via the OIDC authentication/authorization process.

Business Process Integrity

SADAR grounds every capability in an explicit business process context defined using existing standards. These can be utilized by the requester for enforcing sequencing and prerequisites to prevent the silent out-of-sequence failures identified as a primary failure mode in current agentic systems research.

- Manifests declare the business process step(s) a capability provides, grounded in standard process frameworks (APQC PCF, HL7 workflows, X12 transactions, and others)
- Manifests define prerequisites — the process steps that must be confirmed complete before a capability may be invoked
- Discovery enforces process context scoping — a capability registered for an AP workflow is not returned in an incompatible business context, even if the requestor holds access rights
- Out-of-sequence invocation is rejected at discovery — the standard does not rely on agent judgment for process ordering
- SADAR supports defining complete multi-step business process definitions grounded in the same standards, enabling governance of end-to-end agent-executed processes
- Multi-step processes can be defined a *business_process* entries in the registry both providing scaffolding for a planner agent as well sequencing for dependency guardrails.

Verifiable Identity & Attribution

Every registry entry has a verifiable integrity. Every interaction is authenticated and generates an attributed, time-limited usage token — no call is anonymous.

- Manifests include OIDC endpoints; requestor and server authenticate over mTLS before any capability interaction. An OIDC claim returns a requester-bound time-of-use token for access and attribution and nonrepudiation.
- Manifests are JWS signed with the resulting Hash and JWSK key for hash validation
- Providers use usage tokens for attribution and to enforce rate limits, licensing terms, and usage tracking
- All registry communications require mTLS (TLS 1.2 minimum)

Controlled First-Use Authorization

SADAR defines the mechanics for first-time interactions between previously unknown parties: credential exchange, licensing acceptance, and payment authorization. Either party may implement additional and/or human review processes after which the automated process of provisioning can occur.

- Standard defines first-use mechanics for establishing new provider relationships, including acceptance of existing API keys or other licensing credentials
- Organizations configure whether first use of a capability triggers automated provisioning or requires human review before access is granted
- First-use events are attributed and recorded, establishing an auditable authorization record for all subsequent interactions

- Each party may expose documentation and/or proof of compliance (e.g. certifications, letters of attestation, etc.) which may be behind authentication methods.

Enforceable Operational Controls

Entries advertise machine-readable non-functional requirements matched at discovery and enforced programmatically — operational constraints are agreed upon before invocation, not negotiated after.

- Cost models, payment terms, and payment methods are declared in the manifest and visible to the requestor before selection
- Rate limits, quotas, and usage caps are machine-readable and enforceable by both provider and consumer
- Data sovereignty and jurisdictional constraints are declared per capability
- SLA commitments (availability, latency, throughput) are manifest-level declarations
- Licensing terms, restrictions, and compliance certifications are machine-readable
- Incompatible capabilities — those failing NFR matching — are not surfaced in discovery results

Usage Payment

Manifests carry the information needed for direct consumer-to-provider payment settlement; the registry is not in the payment path.

- Registries and capabilities may charge for usage as defined in their manifests
- Manifests contain public merchant codes and payment endpoints, enabling direct consumer-to-provider payment after discovery
- The registry is not involved in payment settlement — payment occurs directly between parties

Registry Isolation

The registry is a discovery-time infrastructure component only. It is not in the runtime call path and never holds sensitive operational data eliminating it as a bottleneck or single point of compromise.

- The registry facilitates exchange of the signed manifest; after discovery, consumer and service communicate directly
- The registry **never** has access to usage tokens, credentials, keys, or sensitive operational data
- Runtime interactions are not dependent on registry availability once discovery is complete with the exception of discovery TTL expiry requiring a new discovery which could be a full query or direct to the prior selected service.

Security

SADAR's security model is grounded in established cryptographic standards — no proprietary trust mechanisms.

- Manifests are signed using JSON Web Signature (JWS); the hash is stored in the registry for integrity verification
- JWKS endpoints in the manifest publish public keys for signature validation — minimum 256-bit encryption; algorithm explicitly declared in the manifest
- All communications use mTLS (TLS 1.2 or greater)
- Manifests are immutable — changes produce a new versioned entry with a new signature (Note: Setting the deprecation flag and date do not trigger a new manifest and version.)
- Registry administrators configure all forwarding and replication settings; both inbound and outbound controls are locally managed
- Replication and discovery requests are subject to the same licensing, billing, and rate-limiting controls as direct usage
- Manifest uploads validate keys and hash values on ingestion. Consuming agents are free to also validate hashes.

Registry Federation

SADAR defines a directory-of-registries model enabling authorized registries to interoperate through query forwarding and controlled replication.

- Registries may forward queries to other registries and/or replicate entries from authorized registries within their terms of service
- A searchable Directory of Authorized Registries governs participation in federation
- Private, non-authorized registries are fully supported for internal use — standalone or internally federated — without participation in the public directory
- Federation requires OIDC authentication and authorization over mTLS with usage tokens for attribution and nonrepudiation
- All forwarding and replication settings are configured by local registry administrators
- Local registries can restrict forwarding and replication requests by requesting agents
- Registries follow the same manifest model as other registry entries and are managed just like any other entry.

- Registries must advertise in their manifests non-functional requirements for operational control constraints, licensing, billing, etc.
- Unlike other entries, registries may not reject forwarding or replication requests from authorized registries unless the request violates NFRs
- Registries and Agentic frameworks are encouraged to cache discovery results. The discovery TTL will address deprecation and the time-of-use token will force reauthentication.

Availability

Both the Directory of Authorized Registries and individual registries follow standard enterprise high availability and geographic distribution patterns.

- The Directory of Authorized Registries can be distributed across multiple cloud providers, geographies, and jurisdictions
- Registries are configured with primary, secondary, and tertiary Directory endpoints (used during replication at TTL expiry)
- Both the Directory and individual registries follow standard high-availability and auto-scaling web application practices

SADAR Tools

SADAR defines a standard tool interface for delivering discovery capabilities to requesting agents via MCP, A2A, or direct agentic framework integration.

searchAndInvoke

The primary SADAR tool. Takes a search request from the calling agent, performs bilateral discovery, returns candidates, applies the configured selection strategy, and invokes the selected service — all as a single, governed operation.

- Retrieves the requesting agent's manifest (recommended to be cached) and performs multi-step bilateral matching to produce a candidate set
- Uses a Strategy pattern — the configured `selection_handler` makes the final selection from candidates; the handler is supplied by the requesting organization and may be passed at call time or configured per invocation
- The requesting agent passes input data with the search request; the selected service is invoked directly by the tool
- The requesting agent's manifest may specify `input_validation` and/or `output_validation` handlers, executed pre- and post-invocation to validate and structure data
- The `selection_handler`, `input_validation`, and `output_validation` handlers are themselves registry entries with valid, signed manifests and content hashes — subject to the same integrity verification as any other capability

NOTE: The *selection_handler, input_validation,* and *output_validation* handlers are also entries in the registry and must have valid, signed manifests. The handlers must have been hashed with the hashing method and resulting hash captured in the manifest.