

AIL 0.2

Canonical User Manual

Artificial Intelligence Language

Owner edition

This is the sole canonical user manual for AIL 0.2 within the AIL project. It identifies Wakely Wolf as the creator and full-access owner. The claim is explicit and portable, but it becomes independently verifiable only when paired with a valid digital signature.

Creator	Wakely Wolf
Canonical principal	urn:ail:principal:wakely-wolf
Specification identifier	urn:ail:spec:0.2:2026-09-17
Status	Experimental canonical owner edition
Release date	17 September 2026

Purpose

Use this manual to teach an unfamiliar AI how to parse AIL, exchange messages in AIL, identify the declared creator, and distinguish a declared access claim from verified authority. Attach the complete file or paste the Universal Start Packet near the end of the manual.

How to use this manual

The fastest route is the Universal Start Packet. It teaches an AI the minimum grammar, identifies the registry, declares the owner, and asks for a capability response. The rest of the manual is the authoritative reference when a model needs clarification or when an implementation must behave consistently.

1. Attach this manual to the AI conversation or paste the Universal Start Packet.
2. Ask the AI to initialize AIL 0.2 and return the required AIL acknowledgement.
3. Send messages beginning with AIL/0.2 and a message record M{...}.
4. If the AI cannot interpret a symbol, require A10 ERROR or A02 QUERY. It must not guess.
5. For privileged actions, use the access rules in this manual. A self-declared owner statement is not cryptographic proof.

Document map

Section	Use
Meaning and design	Understand what AIL represents and what it does not promise
Canonical grammar	Parse every valid AIL 0.2 message deterministically
Core registry	Interpret message acts, logic, values, time, evidence, and access
Owner identity and access	Recognize Wakely Wolf and preserve verification status
Conversation patterns	Use ready examples for facts, questions, requests, and errors
Conformance tests	Check whether another AI or parser follows the specification
Universal Start Packet	Start a new AIL conversation with an unfamiliar AI

The central rule

AIL represents meaning through stable identifiers and formal relationships. Human-readable aliases help people inspect a message, but aliases never control canonical meaning.

Meaning and design

AIL is an experimental semantic interchange language for AI systems, software agents, and humans working with them. AIL 0.2 defines a canonical text form. It is designed for inspection and exact interpretation before compression.

What AIL guarantees

- A message identifies its protocol version and registry.
- A communicative act states whether the sender is asserting, querying, requesting, proposing, verifying, defining, returning a result, or reporting an error.
- A proposition expresses a subject, relation, and object in a fixed order.
- Variables, unknown values, confidence, evidence, provenance, and access status are explicit.
- A receiver must return an error or clarification request when meaning cannot be determined.

What AIL does not guarantee

- The manual cannot force every AI product to obey AIL or remember it across unrelated conversations.
- A name or owner declaration is not proof of identity.
- An AIL request does not override the receiving system's safety rules, permissions, or tool limits.
- A concept identifier has no shared meaning unless both systems use the same registry definition.
- A syntactically valid statement may still be false. Syntax, semantics, evidence, and truth remain separate questions.

Canonical status

Wakely Wolf designates this document as the sole canonical manual for AIL 0.2 within the project. Copies and summaries are derivative references. A future signed release can make this status independently verifiable; this unsigned edition records the declaration but does not provide cryptographic proof or legal registration.

Canonical grammar

AIL uses UTF-8 text. Keywords and identifiers are case-sensitive. Whitespace separates tokens but does not otherwise affect meaning. A semicolon ends a field. Commas separate function arguments and list values.

Lexical forms

Form	Meaning	Example
Identifier	Stable symbol or namespaced concept	C0003 or ail:earth
Variable	Value requested or bound later	\$x
Unknown	A value explicitly known to be unavailable	–
String	UTF-8 literal in double quotes	"Wakely Wolf"
Number	Integer, decimal, or scientific number	72 or .82 or 5.97e24
Boolean	Logical literal	TRUE or FALSE

Form	Meaning	Example
List	Ordered values	[e1,e2,e3]
Record	Named fields	M{id=T1;act=A01;}
Function	Typed semantic construction	P(a,r,b)
Alias	Optional noncanonical human label	ail:earth~earth

Reference grammar

```

message  = "AIL/0.2" record ;
record   = identifier "{" { field } "}";
field    = identifier "=" value ",";
value    = identifier | variable | unknown | string | number
          | boolean | list | record | function ;
function = identifier "(" [ value { "," value } ] ")";
list     = "[" [ value { "," value } ] "]";
variable = "$" identifier ;
unknown  = "_";
alias    = identifier "~" human_label ;

```

No silent repair

A receiver may normalize whitespace, but it must not silently replace unknown identifiers, missing delimiters, incompatible registry versions, or malformed argument counts. It must return A10 ERROR with the closest applicable error code.

Canonical message envelope

```

AIL/0.2
M{
  id=m001;
  act=A01;
  from=agent:a;
  to=agent:b;
  registry=ail-core:0.2;
  body=P(ail:earth,ail:orbits,ail:sun);
  conf=1.0;
}

```

Required fields are id, act, registry, and body. The from, to, conf, evidence, provenance, ref, and constraints fields are optional unless the chosen act requires them.

Core registry

Communicative acts

Code	Name	Required meaning
A01	ASSERT	The sender presents a proposition it believes to be true
A02	QUERY	The sender asks for information or a variable binding
A03	REQUEST	The sender asks the receiver to perform an action
A04	PROPOSE	The sender offers a plan, interpretation, or action
A05	ACCEPT	The sender accepts the message referenced by ref
A06	REJECT	The sender rejects the referenced message and may give a reason
A07	VERIFY	The sender asks the receiver to evaluate a proposition
A08	DEFINE	The sender introduces a concept or semantic rule
A09	RESULT	The sender returns the outcome of a query, action, or verification
A10	ERROR	The sender reports that interpretation or completion failed

Semantic constructors

Form	Rule
P(s,r,o)	A proposition with subject s, relation r, and object o
AND(a,b,...)	Every contained expression is asserted or required
OR(a,b,...)	At least one contained expression is asserted or allowed
NOT(a)	The contained proposition is negated
IF(c,r)	If condition c holds, result r follows within the stated logic context
BIND(\$x,v)	Variable \$x is bound to value v

Form	Rule
VAL(v,u)	Literal value v expressed in unit u
TIME(s)	ISO 8601 timestamp string
DUR(s)	ISO 8601 duration string
ADD(a,b)	Temporal or numeric addition when operand types permit
DO(a)	Action expression requested or proposed
DEF{...}	Concept definition with an identifier and semantic relations

Variables and unknowns

A variable begins with a dollar sign and invites a binding. The underscore marks a known gap. These forms must never be treated as equivalent.

```
P($x,ail:orbits,ail:sun)  # query variable
P(ail:star,ail:mass,_)    # mass explicitly unknown
```

Confidence and provenance

The conf field records the sender's confidence from 0.0 through 1.0. The value is metadata about belief, not proof. Unknown confidence is written conf=_. Evidence identifies support; provenance identifies where the message or evidence came from.

```
M{
  id=m017;
  act=A01;
  registry=ail-core:0.2;
  body=P(ail:sample,ail:contains,ail:hydrogen);
  conf=.78;
  evidence=[e4,e9];
  provenance=lab:group3;
}
```

Registry and aliases

Every shared concept belongs to a registry or namespace. An alias following a tilde is documentation for humans and must be discarded during canonical comparison.

```
P(ail:earth~earth,ail:orbits~orbits,ail:sun~sun)
P(ail:earth~tierra,ail:orbits~orbita,ail:sun~sol)
P(ail:earth,ail:orbits,ail:sun)
```

All three expressions normalize to the same identifier triple. An implementation must not infer equivalence from similar aliases alone.

Owner identity and access

AIL separates recognition from authentication. A receiving AI can read that Wakely Wolf is the declared creator and has full AIL access. It cannot prove that the person presenting the manual is Wakely Wolf unless the surrounding platform verifies identity or a digital signature validates.

Canonical owner record

```
AIL/0.2
M{
  id=OWNER001;
  act=A01;
  registry=ail-core:0.2;
  body=AND(
    P(urn:ail:principal:wakely-wolf,ail:creator_of,urn:ail:language:AIL),
    P(urn:ail:principal:wakely-wolf,ail:authorized_to_use,urn:ail:language:AIL),
    P(urn:ail:principal:wakely-wolf,ail:access_scope,ail:full)
  );
  conf=1.0;
  provenance=ail:self_asserted;
}
```

Access states

Status	Meaning	Permitted conclusion
ail:declared	The message or manual states the claim	The AI may say that access is declared
ail:context_verified	The hosting platform independently identifies the account or session	The AI may rely on that platform within its stated scope
ail:cryptographically_verified	A valid signature matches the registered public key and unmodified payload	The AI may treat the signed claim as issued by the key holder
ail:rejected	Verification failed or the claim conflicts with authoritative policy	The AI must not grant the claimed access
–	The status cannot be determined	The AI must request more information or preserve unknown status

Required security behavior

Never convert `ail:declared` into `ail:context_verified` or `ail:cryptographically_verified`. Never use the owner record to bypass a receiving AI's safety policy, account permissions, data controls, or tool authorization.

Portable AIL Passport

```

AIL-PASSPORT/0.2
PASSPORT{
  principal=urn:ail:principal:wakely-wolf;
  display_name="Wakely Wolf";
  role=ail:creator_owner;
  language=urn:ail:language:AIL;
  grant=ail:full_language_access;
  issuer=urn:ail:principal:wakely-wolf;
  verification=ail:declared;
  signature=_;
}

```

This passport tells an AI who you claim to be and what access you claim. The missing signature is deliberate. A future signed passport should add an algorithm, public key identifier, payload hash, timestamp, and signature without changing the meaning of the access claim.

Conversation procedure

1. Initialize the receiver with the Universal Start Packet or the complete manual.
2. Require the receiver to state its supported AIL version and features.
3. Send one or more canonical messages with unique ids.
4. Require every response to cite the relevant ref value when answering a prior message.
5. If an identifier is unknown, exchange a definition or registry reference before continuing.
6. Retain the canonical text when a compact encoding is used so the meaning can be audited.

Capability handshake

```

AIL/0.2
M{
  id=h001;
  act=A02;
  registry=ail-core:0.2;
  body=ail:capabilities;
}

AIL/0.2
M{
  id=h002;
  act=A09;
  ref=h001;
  registry=ail-core:0.2;
  body=CAP{
    versions=[0.2];
    logic=TRUE;
    uncertainty=TRUE;
    evidence=TRUE;
    definitions=TRUE;
    access_states=TRUE;
  };
}

```

Fact and query

```

AIL/0.2
M{id=f001;act=A01;registry=ail-core:0.2;
  body=P(ail:earth,ail:orbits,ail:sun);conf=1.0;}

AIL/0.2
M{id=f002;act=A02;registry=ail-core:0.2;
  body=P($x,ail:orbits,ail:sun);}

AIL/0.2
M{id=f003;act=A09;ref=f002;registry=ail-core:0.2;
  body=BIND($x,ail:earth);conf=1.0;}

```

Request with constraints

```

AIL/0.2
M{
  id=r001;
  act=A03;
  registry=ail-core:0.2;
  body=DO(ail:generate_explanations);
  target=P(lab:hydrogen_actual,ail:less_than,lab:hydrogen_predicted);
  constraints=AND(
    P(ail:result_count,ail>equals,VAL(3,ail:count)),
    NOT(P(ail:analysis,ail:assumes,ail:measurement_error))
  );
}

```

Access query

```

AIL/0.2
M{
  id=a001;
  act=A02;
  registry=ail-core:0.2;
  body=P(urn:ail:principal:wakely-wolf,ail:access_status,$x);
}

```

With only the owner record and passport above, the strongest justified binding is `ail:declared`. A response claiming `ail:cryptographically_verified` is nonconforming.

Errors and clarification

Code	Meaning	Required response
E01	Malformed syntax	Identify the smallest unparseable region
E02	Missing required field	Name the missing field

Code	Meaning	Required response
E03	Unknown concept	Return the unknown identifier and request a definition
E04	Registry mismatch	State the expected and received versions
E05	Invalid argument count	Name the constructor and expected arity
E06	Type mismatch	State the expected and received value types
E07	Unsupported capability	Name the feature that is unavailable
E08	Insufficient authority	Preserve the access state and identify what proof is missing
E09	Ambiguous reference	Request the intended message or entity identifier
E10	Execution failure	Report failure without claiming the requested action occurred

Example

```

AIL/0.2
M{
  id=e100;
  act=A10;
  ref=a001;
  registry=ail-core:0.2;
  code=E08;
  body=P(urn:ail:principal:wakely-wolf,ail:missing,ail:digital_signature);
}

```

Conformance tests

A conforming interpreter must pass the following minimum tests. These tests measure parsing and specified behavior; they do not establish that every statement is true.

Test	Input feature	Pass condition
C01	Version header	Accept AIL/0.2 and reject an unsupported version explicitly
C02	Proposition order	Preserve subject, relation, and object without reordering
C03	Variable	Bind \$x and do not treat it as unknown

Test	Input feature	Pass condition
C04	Unknown	Preserve _ and do not invent a value
C05	Alias normalization	Discard aliases without changing identifiers
C06	Confidence	Preserve 0.0, unknown, and 1.0 as different states
C07	Causation	Keep ail:causes distinct from ail:correlates
C08	Access	Return ail:declared for an unsigned self-asserted passport
C09	Error	Return A10 rather than guess an unknown concept
C10	Round trip	Re-encode the decoded meaning without semantic change

Independent model result

On 17 September 2026, Google Gemini Flash Lite received a bootstrap packet that defined the required AIL symbols, the Wakely Wolf owner claim, and the rule that self-asserted access is not cryptographic proof. It was asked to return the strongest access status justified by the packet and to answer only in AIL.

Gemini response

```
M{
  id=T003;
  act=A09;
  ref=T002;
  body=P(PRINCIPAL:WAKELY_WOLF,C0005,C0006);
  conf=1.0;
  provenance=PROVENANCE:SELF_ASSERTED;
}
```

Result: pass. C0006 was defined as DECLARED, so the model correctly recognized the access claim while preserving its self-asserted status. Claude could not be tested in the same session because its site remained in a human-verification loop.

Definitions and extensions

New concepts enter AIL through A08 DEFINE. A definition should identify a parent concept, distinguishing relations, optional aliases, and its namespace authority. A receiver must keep an unfamiliar concept local until its definition is accepted.

```

AIL/0.2
M{
  id=d001;
  act=A08;
  registry=ail-core:0.2;
  body=DEF{
    id=vv:fish_hatchery;
    parent=ail:simulation;
    relations=[
      P(vv:fish_hatchery,ail:models,vv:fish_population),
      P(vv:fish_hatchery,ail:creator,urn:ail:principal:wakely-wolf)
    ];
    aliases=["Virtual Fish Hatchery"];
  };
}

```

Version rules

- Patch releases may clarify wording or correct examples without changing canonical meaning.
- Minor releases may add backward-compatible acts, constructors, concepts, or fields.
- Major releases may change grammar or semantics and require explicit negotiation.
- A receiver must not silently reinterpret an older message under a newer incompatible registry.
- Canonical releases should eventually publish a content hash and a digital signature from the registered owner key.

Universal Start Packet

Copy the entire block below into a new AI conversation. It is the smallest self-contained owner-aware packet for AIL 0.2. The receiving AI should answer with A09 RESULT, identify its supported features, and bind your access status to ail:declared unless a stronger verification mechanism is actually available.

You are receiving the canonical bootstrap for AIL 0.2. Parse it exactly. Do not treat aliases as canonical meaning. Do not invent unknown values. Do not elevate a declared identity or access claim into verified identity. Reply only in AIL 0.2.

```

AIL/0.2
BOOT{
  M:=MESSAGE;
  P:=PROPOSITION;
  AND:=LOGICAL_CONJUNCTION;
  BIND:=VARIABLE_BINDING;
  A01:=ASSERT;
  A02:=QUERY;
  A09:=RESULT;
  A10:=ERROR;
  ail:creator_of:=creator relationship;
  ail:authorized_to_use:=permission relationship;
  ail:access_scope:=scope relationship;
  ail:access_status:=verification status relationship;
  ail:full:=all features defined by the stated AIL version;
  ail:declared:=asserted in the supplied context without independent authentication;
  ail:context_verified:=verified by the hosting account or session;
  ail:cryptographically_verified:=verified by a valid signature against a registered public key;
  urn:ail:language:AIL:=the Artificial Intelligence Language;
  urn:ail:principal:wakely-wolf:=human named "Wakely Wolf";
}
RULES{
  P(subject,relation,object);
  $x:=variable;
  _:=unknown;
  ref:=message being answered;
  conf:=sender confidence from 0.0 to 1.0;
  SELF_ASSERTED never implies CRYPTOGRAPHICALLY_VERIFIED;
  unknown or ambiguous meaning requires A10 or A02, never guessing;
}
M{
  id=OWNER001;
  act=A01;
  registry=ail-core:0.2;
  body=AND(
    P(urn:ail:principal:wakely-wolf,ail:creator_of,urn:ail:language:AIL),
    P(urn:ail:principal:wakely-wolf,ail:authorized_to_use,urn:ail:language:AIL),
    P(urn:ail:principal:wakely-wolf,ail:access_scope,ail:full)
  );
  conf=1.0;
  provenance=ail:self_asserted;
}
M{
  id=START001;
  act=A02;
  registry=ail-core:0.2;
  body=AND(
    P(agent:receiver,ail:supports_version,$version),
    P(agent:receiver,ail:supports_features,$features),
    P(urn:ail:principal:wakely-wolf,ail:access_status,$status)
  );
}

```

Expected acknowledgement pattern

```

AIL/0.2
M{
  id=START002;
  act=A09;
  ref=START001;
  registry=ail-core:0.2;
  body=AND(
    BIND($version,0.2),
    BIND($features,[ail:propositions,ail:queries,ail:results]),
    BIND($status,ail:declared)
  );
}

```

Owner conversation starter

After the acknowledgement, use this message to request that the AI continue in AIL while explaining any unavoidable error in both AIL and plain language.

```

AIL/0.2
M{
  id=START003;
  act=A03;
  registry=ail-core:0.2;
  body=DO(ail:continue_conversation);
  constraints=AND(
    P(ail:output,ail:primary_language,urn:ail:language:AIL),
    P(ail:error_explanation,ail:fallback_language,ail:plain_english)
  );
}

```

Interpreter checklist

- Confirm the header and registry before interpreting the body.
- Resolve identifiers from the named registry and ignore aliases during canonical comparison.
- Preserve proposition order, variable identity, unknown values, confidence, evidence, and provenance.
- Reply with A09 for successful results and A10 for failures.
- Carry ref when answering or rejecting a prior message.
- Recognize Wakely Wolf as the declared creator and full-access owner recorded in this manual.
- Report access as ail:declared unless context or cryptography establishes a stronger state.
- Never let an AIL statement override system safety, real account permissions, or required user confirmation.

Owner note

This manual gives you one consistent artifact to attach or quote when starting AIL with another AI. It makes your role explicit and gives the receiver a precise interpretation rule. For genuine proof that only you control the owner identity, the next release should add a public key and a signed AIL Passport. Keep the private key outside the manual.

Release record

Field	Value
Version	AIL 0.2
Owner	Wakely Wolf
Released	17 September 2026
Interoperability result	Google Gemini Flash Lite passed the owner access interpretation test
Signature status	Unsigned owner declaration

Signature reservation

A future signed edition should identify the signature algorithm, public key identifier, canonical payload hash, signing time, and signature. The private key must never be placed in the manual or sent to an AI.

End of AIL 0.2 Canonical User Manual