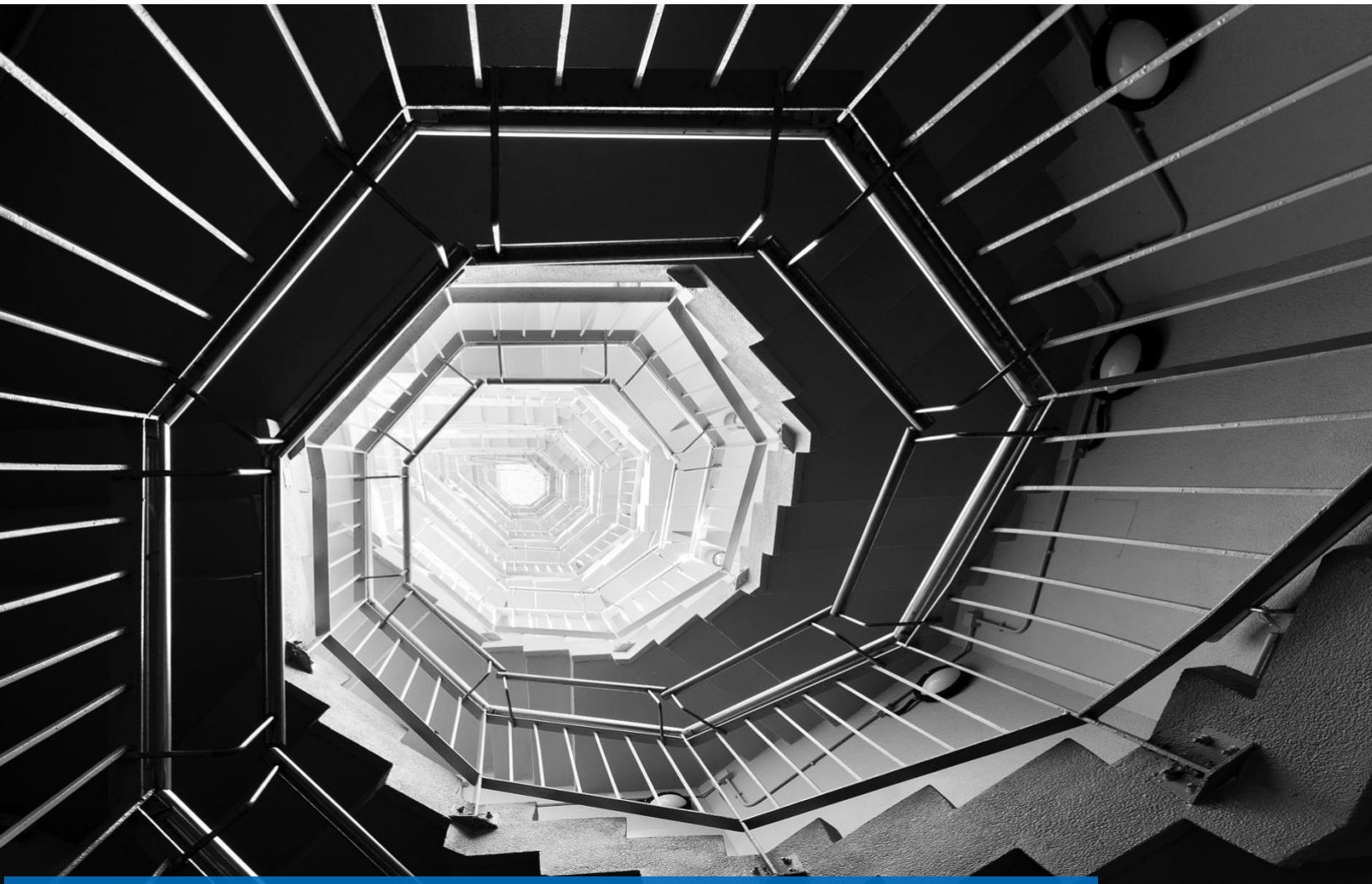


Building Cognitive Applications

Atlas Interface Reference

TABLE OF CONTENTS

ATLAS INTERFACE REFERENCE

GETTING STARTED **6**

CONCEPT APIS: **8**

I. NOTIFICATION INTERFACE: 10

KNOWLEDGE BASE ATTACH (T1): 12
KNOWLEDGE BASE DETACH (T2): 13
SIMULATION ATTACH (T3): 14
SIMULATION DETACH (T4): 15
SIMULATION LINK (T5): 16
SIMULATION UNLINK (T6): 2
SIMULATION OBSERVE (T7): 3
SIMULATION HEAR (T8): 5
SIMULATION RECEIVE (T9): 6
EXPECTED RESPONSES: 7

II. INSTANTIATOR INTERFACE: 8

CREATE AN INSTANCE (I1): 10
DESTROY AN EXISTING INSTANCE (I2): 11
LOAD AN INSTANCE (I3): 12
SAVE AN INSTANCE (I4): 13
RETRIEVE THE SIMULATION OBJECT OF AN INSTANCE(I5): 14
RETRIEVE A SUBORDINATE INSTANCE (I6): 15
RETRIEVE THE SIZE OF AN INSTANCE (I7): 16
GET THE VALUE OF AN INSTANCE (I8): 17
SET THE VALUE OF AN INSTANCE (I9): 18
SET THE NUMBER OF ELEMENTS IN AN INSTANCE (IA): 19
GET THE NUMBER OF ELEMENTS IN AN INSTANCE (IB): 20
RETRIEVE AN INSTANCE ELEMENT (IC): 21
RETRIEVE THE FIRST ELEMENT IN AN INSTANCE (ID): 22
RETRIEVE THE NEXT ELEMENT IN AN INSTANCE (IE): 23
ADD AN ELEMENT TO AN INSTANCE (IF): 24
REMOVE AN EXISTING ELEMENT FROM AN INSTANCE (IG): 25

III. INTERPRETATION INTERFACE: 26

DO ASPECT: 26
BE ASPECT: 26
HAVE ASPECT: 26
SIMULATION METHODS (M*): 27

COGNITIVE API BUNDLE: **2**

I. COGNITIVE INTERFACE: 4

ACCESS AN INTERFACE (C1): 5
REGISTER AN INSTANCE MANAGER (C2): 6
REGISTER A SIMULATION METHOD (C3, C4, C5): 7
RETRIEVE A CONCEPT (C6): 2
RETRIEVE A SUB-CONCEPT (C7): 3
CHECK CONCEPT COMPATIBILITY (C8): 4
REGISTER AN INSTANCE MANAGER OF A SUBORDINATE CONCEPT (C9): 5
ADD CONCEPT ATTRIBUTES (CA): 6
MANAGE ATLAS (CB, CC, CD): 7

II. DOMAIN INTERFACE: 8

CREATE A DOMAIN (D1): 9
CREATE A DOMAIN (D2): 10
DESTROY A DOMAIN (D3): 11
DOMAIN MANAGER (DX:ATLAS DOMAIN MANAGER METHOD): 12
CREATE A ZONE (D4): 13
CREATE A ZONE (D5): 14
DESTROY A ZONE (D6): 15
ZONE MANAGER (CX:ATLAS ZONE MANAGER METHOD): 16
CREATE A SIMULATION (D7): 17
CREATE A SIMULATION (D8): 18
DESTROY A SIMULATION (D9): 19
SIMULATION MANAGER (SX: ATLAS SIMULATION MANAGER METHOD): 20

III. DIRECTIVE INTERFACE: 21

RETRIEVE A DIRECTIVE COMPONENT BY ID (X1): 22
RETRIEVE NEXT DIRECTIVE COMPONENT (X2): 24
RETRIEVE A DIRECTIVE COMPONENT BY RELATION (X3): 26
REPHRASE A DIRECTIVE COMPONENT (X4): 28
GET THE NUMBER OF COMPONENTS IN A DIRECTIVE (X5): 29
GET DIRECTIVE ACTION (X6): 30
GET DIRECTIVE STATE (X7): 32
SET A DIRECTIVE COMPONENT HANDSHAKE(X8): 33
CLEAR DIRECTIVE HANDSHAKES (X9): 34
REMEMBER A DIRECTIVE (XA): 35
FORGET A DIRECTIVE DATA (XB): 36
DIRECTIVE MANAGER (XX: ATLAS DIRECTIVE MANAGER METHOD): 37

IV. LABEL INTERFACE: 38

GET A LABEL HANDLE (L1): 39
GET A LABEL STRING (L2): 40
GET A LABEL VALUE FROM A REFERENCE (L3): 41
GET A LABEL VALUE FROM A REFERENCE (L4): 42
ADD A LABEL TO AN EXISTING REFERENCE (L5): 43
ADD A LABEL TO AN EXISTING REFERENCE (L6): 44
REMOVE A LABEL FROM AN EXISTING REFERENCE (L7): 45
REMOVE A LABEL FROM AN EXISTING REFERENCE (L8): 46
RETRIEVE THE FIRST LABELED REFERENCE (L9): 47
RETRIEVE THE FIRST LABELED REFERENCE (LA): 49
RETRIEVE THE NEXT LABEL REFERENCE (LB): 51

V. LEDGER INTERFACE: 53

CREATE A LEDGER (V1): 55
DESTROY A LEDGER (V2): 56
RETRIEVE A LEDGER (V3): 57
CLEAR THE LEDGER (V4): 58
GET LEDGER ENTRY DATA (V5): 59
ADD A LEDGER ENTRY (V6): 60
RETRIEVE EARLIEST LEDGER ENTRY (V7): 61
RETRIEVE LATEST LEDGER ENTRY (V8): 62
RETRIEVE PREVIOUS LEDGER ENTRY (V9): 63
RETRIEVE NEXT LEDGER ENTRY (VA): 64
REGISTER A LEDGER KEYWORD (VB): 65
UNREGISTER A LEDGER KEYWORD(VC): 66
PROCESS A LEDGER KEYWORD (VD): 67
LEDGER MANAGER (VX:ATLAS LEDGER MANAGER METHOD): 68

SIMULATION API BUNDLE: 70

I. SIMULATION INTERFACE: 72

GET SIMULATION TIME (S1): 73
CREATE AN ATLAS CLONE (S2): 74
DESTROY AN ATLAS CLONE (S3): 75
CREATE A SIMULATION OBJECT (S4): 76
DESTROY A SIMULATION OBJECT (S5): 77
SET SIMULATION OBJECT PARENT (S6): 78
LOAD A SIMULATION OBJECT (S7): 80
SAVE A SIMULATION OBJECT (S8): 81
ACTIVATE A SIMULATION OBJECT (S9): 82
DEACTIVATE A SIMULATION OBJECT (SA): 83
ATLAS CLONE MANAGER (PX: ATLAS CLONE MANAGER METHOD): 84

II. **AGENT INTERFACE:** 85

CREATE A CONTEXT AGENT (A1): 86
CREATE A CONTEXT AGENT (A2): 87
DESTROY A CONTEXT AGENT (A3): 88
REGISTER A CONTEXT AGENT MANAGER (A4): 89
JOIN AN ATLAS CLONE (A5): 90
LEAVE AN ATLAS CLONE (A6): 91
ENTER A ZONE (A7): 92
ENTER A ZONE (A8): 93
EXIT A ZONE (A9): 94
EXIT A ZONE (AA): 95
COMMUNICATE IN NATURAL LANGUAGE (AB): 96
SEND A COMMUNICATION DIRECTIVE (AC): 97
SEND A BINARY COMMUNICATION MESSAGE (AD): 98
EXECUTE A SCRIPT (AE): 99
RETRIEVE AN ATTACHMENT (AF): 100
RETRIEVE AN ATTACHMENT (AG): 101
REGISTER AN EXPERT (AH): 102
UNREGISTER AN EXPERT (AI): 103
OBSERVE ANOTHER AGENT (AJ): 104
MANAGE CONTEXT (AX:ATLAS AGENT MANAGER METHOD): 105

III. **REFERENCE INTERFACE:** 106

ADD A REFERENCE (R1): 107
ADD A REFERENCE (R2): 108
CREATE AN INSTANCED REFERENCE (R3): 110
CREATE AN INSTANCED REFERENCE (R4): 111
REGISTER A REFERENCE MANAGER (R6): 114
REQUEST A PUBLIC HANDLE OF A REFERENCE (R7): 115
LOAD A REFERENCE (R8): 116
SAVE A REFERENCE (R9): 117
OBSERVE A REFERENCE (RA): 118
GET REFERENCE DATA (RB): 119
GET THE SPATIAL TRANSFORM OF A REFERENCE (RC): 121
MANAGE A REFERENCE (RX:ATLAS REFERENCE MANAGER METHOD): 122

IV. **INSTANCE INTERFACE:** 123

CREATE AN INSTANCE (M1): 124
DESTROY AN INSTANCE (M2): 125
LOAD AN INSTANCE (M3): 126
SAVE AN INSTANCE (M4): 127
GET THE VALUE OF AN INSTANCE (M5): 128
SET THE VALUE OF AN INSTANCE (M6): 129
RETRIEVE THE SIMULATION OBJECT OF AN INSTANCE (M7): 130

V. **GROUP INTERFACE:** 131

CREATE A GROUP (G1): 132

DESTROY A GROUP (G2): 133
LOCK A GROUP (G3): 134
ADD A GROUP MEMBER (G4): 135
REMOVE A GROUP MEMBER (G5): 136
CLEAR A GROUP (G6): 137
RETRIEVE A GROUP MEMBER COUNT (G7): 138
RETRIEVE THE FIRST GROUP MEMBER (G8): 139
RETRIEVE THE NEXT GROUP MEMBER (G9): 141
MANAGE A GROUP (GX: ATLAS GROUP MANAGER METHOD): 143
VI. ATTRIBUTES INTERFACE: 145
Location Attribute Interface: 146
GET LOCATION ATTRIBUTE VERSION (1): 148
SET LOCATION ATTRIBUTE COORDINATE (2): 149
UPDATE LOCATION ATTRIBUTE COORDINATE (3): 151
RETRIEVE EARLIEST ATTRIBUTE COORDINATE (4): 153
RETRIEVE LATEST LOCATION ATTRIBUTE COORDINATE (5): 155
RETRIEVE PREVIOUS LOCATION ATTRIBUTE COORDINATE (6): 157
RETRIEVE LOCATION NEXT ATTRIBUTE COORDINATE (7): 159
RETRIEVE A LIST OF LOCATION ATTRIBUTE COORDINATES (8): 161
PAUSE SIMULATION OF A LOCATION ATTRIBUTE COORDINATE (9): 163
RESUME SIMULATION OF A LOCATION ATTRIBUTE COORDINATE (A): 164
Size Attribute Interface: 165
GET SIZE ATTRIBUTE VERSION (1): 166
Shape Attribute Interface: 167
GET SHAPE ATTRIBUTE VERSION (1): 168
Texture Attribute Interface: 169
GET TEXTURE ATTRIBUTE VERSION (1): 170
Color Attribute Interface: 171
GET COLOR ATTRIBUTE VERSION (1): 172
SET ATTRIBUTE COLOR DATA (2): 173

Getting started

Welcome to the Atlas System (Atlas) Interface Reference.

Atlas offers multiple co-dependent components (capabilities) available through C++ APIs.

This document provides detailed APIs for interacting with the Atlas System:

- Concept API Bundle.
- Cognitive API Bundle.
- Simulation API Bundle.

Atlas system documentation is intended for developers and subject matter experts seeking to build and improve cognitive applications, services, and experiences.

To learn how to develop cognitive simulations using Atlas, we recommend starting with the document “Atlas System Guide” followed by the “Atlas Process Guide” which are also part of the Atlas Development Kit.

Atlas

Concept API Bundle

Concept APIs:

Introduction :

The concept API handles two-way functionality between Atlas and concepts.

Concept APIs are grouped into the following:

- **Notification Interface:** Manages concept attachments and messages.
- **Instantiator Interface:** Manages concept instantiations and messages.
- **Interpretation Interface:** Manages concept simulations

The Concept API serves two purposes:

- Manages incoming internal domain messages. These messages come in the form of calls to two sets of virtual methods overridden by the axiom from the ***AtlasAxiom*** class:
  - **Tx** methods handle knowledge base and simulation messages
  - **Ix** methods handle concept instantiator requests.
- Interprets aspects advertised by the definition of a concept using:
  - **Yx** methods interpret the advertised aspects of the concept definition.

Methods **Tx**, **Ix**, and **Yx** are called by Atlas and handled by the concept axiom.

Annotation:

In the overview diagram, the concept API methods are confined to the following delimiter.

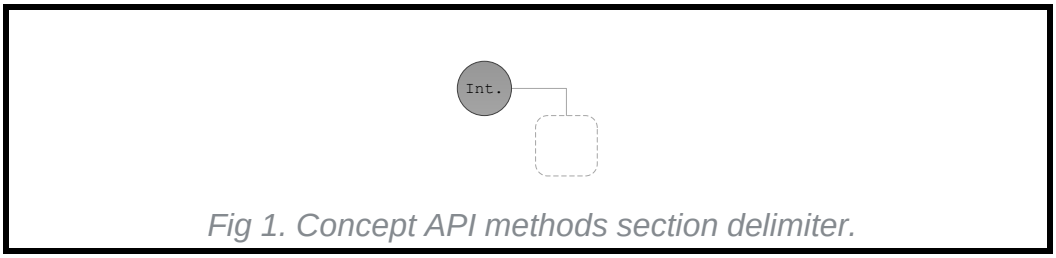


Fig 1. Concept API methods section delimiter.

Concept API Interfaces:

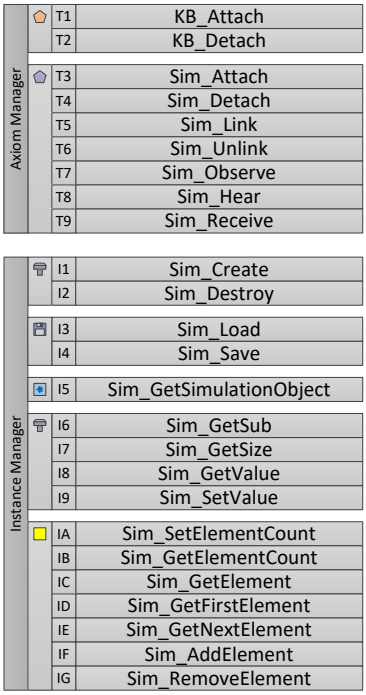


Fig 2. Common concept API methods: These virtual methods are inherited from AtlasAxiom and expanded by the developer to support the functionality requested by Atlas from the concept interpretation.

I. Notification Interface:

A concept is learned by Atlas in a multi-stage process.

For each stage entered the concept is informed of the state changes through a series of domain messages.

Domain messages:

Domain messages are infrastructure messages that inform concepts about the current state of the domain.

Each concept should implement its own message handler in order to intercept domain messages.

Each concept is responsible for handling incoming domain messages and responding to them accordingly.

Since a domain is composed of a knowledge base and a simulation, there are two main types of domain messages:

Knowledge Base Messages (KB):

For each stage entered, the axioms are informed of the state change.

During the */Learn* command, Atlas invokes the *T1:KB_Attach* methods of every axiom referenced in the learned interpretation.

KB_Attach takes three parameters as input:

- The agent that invoked the /Learn command.
- The handle of the concept interpreted.
- The attachment parameters defined in the *Definition* block of the interpretation file.

During attachment, the interpretation is expected to register the *Do*/*Be*/*Have* simulation methods as well as the instance manager (instantiator). Attachment is performed in reverse, recursive order. Sub-concepts are attached first, parent concepts follow, and the top concept attaches last.

Simulation Messages (Sim):

A concept (interpretation) joins the simulation in a two-stage process.

- Attaches self and instances to the simulation.
- Links to other concepts and instances in the simulation.

The newly learned concept attaches to the simulation shortly after the knowledge base attachment (*T1*) has been successfully completed.

Axiom Notification Interface:

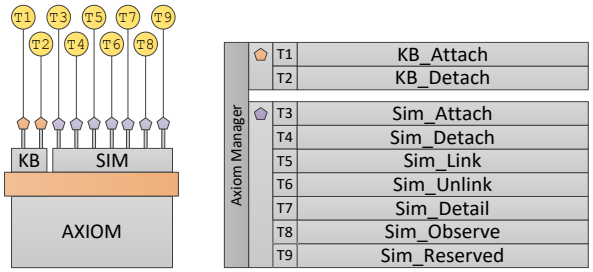


Fig 3. Diagram and table of methods of the axiom notification interface.

KNOWLEDGE BASE ATTACH (T1):

```
virtual titan_result_t AtlasKB_Attach(TITAN_HANDLE64
concept, const TITAN_STRING parameters)
```

Method **T1** is called once per axiom per interpretation during attachment.

Where:

- **concept:** This is the handle of the concept being interpreted by this attaching axiom.
- **parameters:** This is the optional configuration string derived from the interpretation file. It is used by the axiom for initialization. The content of this string is specified during the /Learn command or in the interpretation file. (See Manifest, Interpretation)

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

When a concept interpretation axiom is loaded into the knowledge base, it starts to receive state transition messages.

The first message an axiom receives is: ***Knowledge Base Attach***.

This message arrives shortly after an axiom is loaded into the knowledge base.

This is the first method called by Atlas to the interpretation axiom.

During this stage, the axiom is expected to set up the knowledge base interpretation methods and the simulation instantiator.

Action:

During **KNOWLEDGE BASE ATTACH**, the axiom is expected to:

- Discretely interpret parameters string.
- Map axiom methods to interpretation methods in the knowledge base.
- Set an instance manager for the interpretation.
- Initialize axiom-specific data.

KNOWLEDGE BASE DETACH (T2):

```
virtual titan_result_t AtlasKB_Detach(TITAN_HANDLE concept)
```

Method **T2** is called once per axiom per interpretation during detachment in reverse, recursive order.

Where:

- **concept**: This is the handle of the concept being removed from the knowledge base.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

When a concept interpretation is removed from the knowledge base, the associated axioms are detached from Atlas.

This is the last method called by Atlas to the interpretation axiom.

During this stage, the axiom detaches from Atlas.

Action:

During **KNOWLEDGE BASE DETACH,** the axiom is expected to:

- Destroy axiom-specific data.
- Destroy axiom class.

SIMULATION ATTACH (T3):

```
virtual titan_result_t
AtlasSimulation_Attach(TITAN_HANDLE64
calling_reference, TITAN_HANDLE64 concept)
```

Method **T3** is called once per axiom per interpretation in reverse, recursive order during attachment.

Where:

- **concept:** This is the handle of the concept being interpreted by this attaching axiom.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

During this stage, a concept can begin creating and advertising instances inside the cognitive space.

Action:

During the **SIMULATION ATTACH**, the axiom is expected to:

- Create zones, and context agents.
- Create and advertise instances in zones for other concepts to find.

SIMULATION DETACH (T4):

```
virtual titan_result_t AtlasSim_Detach(TITAN_HANDLE concept)
```

Method **T4** is called when a simulation is about to detach a concept.

Where:

- **concept**: This is the handle of the concept being removed from the knowledge base.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

During this stage, the interpretation is expected to internally prepare for a shutdown and cease communication with Atlas.

Action:

During **SIMULATION DETACH**, the axiom is expected to:

- Remove exclusive agents.
- Remove exclusive zones.
- Remove exclusive groups.
- Remove exclusive labels.
- Remove exclusive references.
- Remove exclusive instances.
- Remove all concept-specific data.

SIMULATION LINK (T5):

```
virtual titan_result_t AtlasSim_Link(TITAN_HANDLE64 concept)
```

Method **T5** comes shortly after the **T3** method has been successfully processed.

Once the instances are created in the simulation attachment stage, different concepts can seek each other's instances and interfaces during this stage.

Where:

- **concept:** This is the handle of the concept being interpreted by this linking axiom.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

During this stage, interpretations cannot yet interact with the simulation but can seek and establish communication with one another.

Action:

During **SIMULATION LINK**, the axiom is expected to:

- Seek advertised instances in zones and prepare for activation.

SIMULATION UNLINK (T6):

```
virtual titan_result_t AtlasSim_Unlink(TITAN_HANDLE64 concept)
```

Method **T6** is called when the concept is ready to unlink all its current interfaces, references, and instances from the simulation.

Where:

- **concept:** This is the handle of the concept being interpreted by this linked axiom.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_*:** Operation failed.

Info:

When the simulation is about to unlink a concept from the simulation environment, it issues ***Sim_Unlink*** to all its axioms in reverse, recursive order.

From this point forward, simulation data is no longer reliable or recoverable, and the destruction of the concept is imminent.

Action:

During unlinking, the axiom is expected to:

- Sever all links with other interfaces and instances in the simulation.
- Prepare for imminent detachment.

SIMULATION OBSERVE (T7):

```
virtual titan_result_t AtlasSim_Observe(TITAN_HANDLE64
observed_reference, TITAN_HANDLE64 observer_reference,
TITAN_HANDLE64 ping_type, titan_directive_t
directive_handle, titan_auxiliary_parameters_t
directive_params, TITAN_HANDLE64 handshake = 0)
```

Upon observing a *Ping* directive by a source reference, an observer concept will receive a call to method ***T7***. The observing reference concept method handles the observed directive.

Where:

- **observer_reference:** Private handle of the observing reference.
- **observed_reference:** Public handle of the observed reference.
- **ping_type**: Type of message being observed, where:
  - **TITAN_A0_VALIDATE**: Agent should validate ping data.
  - **TITAN_A0_TRANSLATE**: Agent should translate ping data.
  - **TITAN_A0_REMEMBER**: Agent should remember ping data.
  - **TITAN_A0_FORGET**: Agent should forget ping data.
  - **TITAN_A0_SIMULATE**: Agent should simulate ping data.
- **directive_handle**: Observed directive handle.
- **directive_params**: A sub-component of the directive that is exposed as a C structure, where:
  - **aux_type**: This is the simulated aspect type. Available types are *TITAN_AUX_DO*, *TITAN_AUX_BE*, and *TITAN_AUX_HAVE* corresponding to the *Do*, *Be*, and *Have* methods respectively.
  - **expression_type**: This is the expression type bitmap, where:
    - **TITAN_EXP_TYPE_QUERY**: This bit is true when the expression is a question.
    - **TITAN_EXP_TYPE_STATEMENT**: This bit is true when the expression is a statement, and not an imperative.
    - **TITAN_EXP_TYPE_PROSPECTION**: Expression states possible outcomes (can).
    - **TITAN_EXP_TYPE_PREDICTION**: Expression states predicted outcomes (should/must).
    - **TITAN_EXP_TYPE_OBJECTIVE**: Expressions states desired outcomes (want/need).
    - **TITAN_EXP_TYPE_WHO**: Expression is requesting agent information about the directive.
    - **TITAN_EXP_TYPE_WHEN**: Expression is requesting time information about the directive.
    - **TITAN_EXP_TYPE_WHERE**: Expression is requesting space information about the directive.

    - **TITAN_EXP_TYPE_WHAT**: Expression is requesting reference/verb information about the directive.
    - **TITAN_EXP_TYPE_WHY**: Expression is requesting cause of the directive.
    - **TITAN_EXP_TYPE_HOW**: Expression is requesting process of the directive.
    - **TITAN_EXP_TYPE_PREDICATE**: Expression is requesting veracity of the directive.
  - **aux_tense**: This is the tense of the simulation, where:
    - **TITAN_TENSE_INFINITIVE:** Infinitive tenses are used to either state or query a generalization about the expression regardless of time.
    - **TITAN_TENSE_IMPERATIVE:** Imperative tense is a command in the present.
    - **TITAN_TENSE_PRESENT:** Present tense (expression ‘is’).
    - **TITAN_TENSE_PAST:** Past tense (expression ‘was’).
    - **TITAN_TENSE_FUTURE:** Future tense (expression ‘will’).
    - **TITAN_TENSE_PAST_PAST:** Past-Past (expression ‘had been’).
    - **TITAN_TENSE_PAST_FUTURE:** Past-Future (expression ‘would have been’).
    - **TITAN_TENSE_FUTURE_PAST:** Future-Past (expression ‘will have been’).
    - **TITAN_TENSE_FUTURE_FUTURE:** Future-Future (expression ‘will later be’).
  - **valence**: The valence index is the number of objects in the directive (See Directive Valence).
- ***handshake***: Discretionary validation token to be processed by this method.

RETURN: The observation method return value 0 should be interpreted as an *unknown answer* to the directive. It is still a valid answer if the handling concept does not know the answer or does not want to affect the response of a simulation. The appropriate response value to a failed handshake or authentication by the observation method is also 0. Observation method results should be interpreted as:

- **-100:** NO
- **-50:** MAYBE NOT
- **0:** DON'T KNOW
- **+50:** MAYBE
- **+100:** YES

SIMULATION HEAR (T8):

```
virtual titan_result_t AtlasSim_Hear(TITAN_HANDLE64
hearing_reference, TITAN_HANDLE64 telling_reference,
TITAN_HANDLE64 tell_type, const TITAN_STRING
expression, titan_auxiliary_parameters_t
directive_params, TITAN_HANDLE64 handshake = 0)
```

Upon hearing a *Tell* expression by source reference, a concept hosting the hearing (destination) reference will receive a call to method **T8**. This method is expected to handle the heard expression by the destination reference.

Where:

- **hearing_reference:** Private handle of the reference receiving the *Tell*.
- **telling_reference:** Public handle of the reference issuing a *Tell*.
- **tell_type**: Type of message being told, where:
  - **TITAN_A0_VALIDATE**: Agent should validate tell data.
  - **TITAN_A0_TRANSLATE**: Agent should translate tell data.
  - **TITAN_A0_REMEMBER**: Agent should remember tell data.
  - **TITAN_A0_FORGET**: Agent should forget tell data.
  - **TITAN_A0_SIMULATE**: Agent should simulate tell data
- **expression**: Text string of expression being told.
- ***handshake***: Discretionary validation token to be processed by this method.

RETURN:

- **TITAN_RESULT_OK:** Operation was successful.
- **TITAN_RESULT_***: Operation failed.

SIMULATION RECEIVE (T9):

```
virtual titan_result_t AtlasSim_Receive(TITAN_HANDLE64
receiver_reference, TITAN_HANDLE64 sender_reference,
TITAN_SIZE32 message_size, TITAN_HANDLE32 message_type,
TITAN_HANDLE64 message_data, TITAN_HANDLE64 handshake =
0)
```

Upon receiving a *Send* datagram message by a source reference, a concept hosting the hearing (destination) reference will receive a call to **T9**. This method is expected to handle the received message.

Where:

- **receiver_reference:** Private handle of the reference receiving the message.
- **sender_reference:** Public handle of the reference sending the message.
- **message_size**: Size of the message.
- **message_type**: User-defined message type.
- **message_data**: Discretionary 64-bit value representing message data.
- ***handshake***: Discretionary validation token to be processed by this method.

RETURN:

- **TITAN_RESULT_OK:** The message has been successfully received.
- **TITAN_RESULT_*:** Operation failed.

EXPECTED RESPONSES:

All knowledge base and simulation message handlers must return a ***titan_result_t*** **value**. The following list shows the expected responses from the axiom and the corresponding actions taken by Atlas:

TITAN_RESULT_OK: This is the expected return value upon successful operation. Any other value will be interpreted as a failure.

Atlas's response to failed stages is categorized below:

- **Knowledge Base Attach:** A failure to attach results in failure to learn the concept. *KB_Attach: value other than TITAN_RESULT_OK results in a failure to learn the concept hosted by the attaching axiom.*
- **Simulation Attach:** A failure to attach results in failure to learn the concept. *Sim_Attach: value other than TITAN_RESULT_OK results in a failure to learn the concept hosted by the attaching axiom.*
- **Simulation Link:** A failure to link results in failure to learn the concept. *Sim_Link*: *value other than TITAN_RESULT_OK results in a failure to learn the concept hosted by the linking axiom.*
- **Simulation Start:** A failure to activate will fail all future calls to this interpretation.
- **Simulation Pause:** The simulation will deactivate the interpretation regardless of result.
- **Simulation Stop:** The simulation will deactivate the interpretation regardless of result.
- **Simulation Unlink:** A failure to unlink will be unlinked regardless of result. *Sim_Unlink*: *value other than TITAN_RESULT_OK results in a failure, but Atlas unlinks the concept from the simulation, nonetheless.*
- **Simulation Detach:** The interpretation will be destroyed regardless of result. *Sim_Detach*: *value other than TITAN_RESULT_OK results in a failure, but Atlas detaches the concept from the simulation, nonetheless.*
- **Knowledge Base Detach:** The concept will be detached regardless of result. *KB_Detach*: *value other than TITAN_RESULT_OK results in a failure, but Atlas detaches the concept from the knowledge base, nonetheless.*

II. Instantiator Interface:

The application of knowledge happens in the simulation.

A concept is tasked with supplying the simulation with examples; these examples are known as simulation instances, or simply **instances**.

The supplier of instances into the simulation is known as a simulation manager, or instantiator.

Each concept is responsible for providing its own simulation instantiator through method *C2: Sim_RegisterInstantiator*.

The instantiator is expected to perform the following duties:

- Create and destroy instances.
- Save and load instances from long-term storage.
- Retrieve or modify instance elements.
- Retrieve or modify instance values.
- Retrieve sub-instances if the concept is hierarchical.
- Retrieve registered attribute data (simulation objects).

An instance is any data or rule that the interpretation stores in order to represent an example of the concept it is interpreting.

Instance data is represented by a unique, non-zero,64-bit number that maps to a particular example that the interpretation has provided.

The instantiator is assigned to a concept or a subordinate concept using *C2:Sim_RegisterInstantiator* or *C9:Sim_RegisterSubInstantiator* respectively.

Instances can represent one or multiple examples.

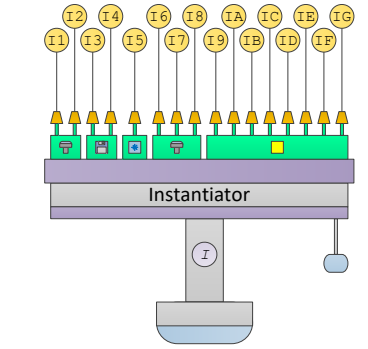
When an instance represents a single example, it has one element.

When an instance represents multiple examples at once, it has multiple elements, one for each example.

This multi-element instancing allows the instantiator to create one unique 64-bit value for several examples of the concept.

Multi-element instances are practical when referencing arrays of similar instances such as blades of grass or human hair where the entire batch is represented by one name or reference, yet every single element can also be referenced.

Instantiator Interface



Instance Manager		
	I1	Sim_Create
	I2	Sim_Destroy
	I3	Sim_Load
	I4	Sim_Save
	I5	Sim_GetSimulationObject
	I6	Sim_GetSub
	I7	Sim_GetSize
	I8	Sim_GetValue
	I9	Sim_SetValue
	IA	Sim_SetElementCount
	IB	Sim_GetElementCount
	IC	Sim_GetElement
	ID	Sim_GetFirstElement
	IE	Sim_GetNextElement
	IF	Sim_AddElement
	IG	Sim_RemoveElement

Fig 4. Instance manager interface.

CREATE AN INSTANCE (I1):

```
titan_result_t AtlasSim_Create(TITAN_HANDLE64
private_reference, TITAN_ULONG num_elements,
TITAN_HANDLE64 parameters, TITAN_HANDLE64 handshake =
0)
```

Method **I1** is called when Atlas requests an axiom to create an instance of its interpretation.

Where:

- **private_reference**: Private handle of the reference whose instance is being created (See *Reference*). If the handle is 0, then the instance has no reference. If handle is non-zero, then the name of the new instance reference can be retrieved.
- **num_elements:** The number of elements to be created within the instance.
- **parameters**: Optional parameters used for creating this instance (See */Create*).
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- 64-bit instance handle upon successful operation.
- 0 upon failure to create instance data.

DESTROY AN EXISTING INSTANCE (I2):

```
titan_result_t AtlasSim_Destroy(TITAN_HANDLE64
instance, TITAN_HANDLE64 handshake = 0)
```

Method **I2** is called when Atlas requests the destruction of an existing Instance.

Where:

- **instance:** The existing handle of the instance to be destroyed.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance was successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

LOAD AN INSTANCE (I3):

```
titan_result_t AtlasSim_Load(TITAN_HANDLE64
private_reference, titan_file_t *file, const
TITAN_STRING name, TITAN_HANDLE64 &instance,
TITAN_HANDLE64 &element, TITAN_HANDLE64 handshake = 0)
```

Method **I3** is called when Atlas requests the axiom to load an instance.

The loaded instance will be assigned to ***private_reference*** upon successful operation.

Where:

- **private_reference**: Private handle of the reference requesting the load.
- **file**: Atlas data file pointer. If the file parameter is not defined, the name parameter should contain the file path (See Files).
- **name**: Name of the file to be loaded if the file parameter is null.
- **instance**: Return handle of the instance if it is loaded successfully.
- **element**: Return handle of the instance element if it is loaded successfully.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance was successfully loaded, where:
  - **instance:** holds the loaded instance.
  - **element:** holds the loaded instance element or 0 if the instance has no element.
- **TITAN_RESULT_*:** See Atlas Errors for more information

SAVE AN INSTANCE (I4):

```
titan_result_t AtlasSim_Save(titan_file_t *file, const
TITAN_STRING name, TITAN_HANDLE64 instance,
TITAN_HANDLE64 element, TITAN_HANDLE64 handshake = 0)
```

Method **I4** is called when Atlas requests an axiom to save an instance.

Where:

- **file**: Atlas data file pointer. If the *file* parameter is not defined, the *name* parameter should contain the file path (See *Files*).
- **name**: Name of the file to be loaded if the *file* parameter is null.
- **instance:** Handle of the instance to be saved.
- **element:** Handle of the instance element to be saved (or 0 for no elements).
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance was successfully saved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE THE SIMULATION OBJECT OF AN INSTANCE(I5):

```
TITAN_HANDLE64
AtlasSim_GetSimulationObject(TITAN_HANDLE64 instance,
TITAN_HANDLE64 element, request_data TITAN_HANDLE64
handshake = 0)
```

Method **I5** is called when an interpretation instance has been assigned attributes (See *CA:AddAttribute*).

The axiom should return the simulation object handle from the supplied instance and element handles. The simulation object handle should have been previously generated with method *S4:CreateSimulationObject*.

Where:

- **instance:** Handle of the instance whose simulation object is being retrieved.
- **elements:** Handle of the instance element whose attribute is being retrieved (or 0 for no elements).
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- Handle of the instance's virtual object upon successful retrieval.
- 0 upon failure to retrieve the instance's virtual object.

RETRIEVE A SUBORDINATE INSTANCE (I6):

```
titan_result_t AtlasSim_GetSub(TITAN_HANDLE64 instance,
TITAN_HANDLE64 element, TITAN_HANDLE64 sub_concept,
TITAN_HANDLE64 &sub_instance, TITAN_HANDLE64
&sub_element, TITAN_HANDLE64 handshake = 0)
```

Method **I6** is called when Atlas requests an existing instance to be destroyed.

Where:

- **sub_concept**: Subordinate (*Have*) concept of the supplied instance being retrieved.
- **instance:** Instance whose subordinate is being retrieved.
- **element**: Element of the instance whose subordinate is being retrieved.
- **sub_instance:** Returned subordinate instance retrieved.
- **sub_element**: Returned subordinate element retrieved.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance subordinate has been successfully retrieved. Values in *sub_instance* and *sub_element* are valid.
- **TITAN_RESULT_*:** See Atlas Errors for more information. Values in *sub_instance* and *sub_element* are invalid.

RETRIEVE THE SIZE OF AN INSTANCE (I7):

```
TITAN_SIZE64 AtlasSim_GetSize(TITAN_ULONG num_elements,
TITAN_HANDLE64 params, TITAN_HANDLE64 handshake = 0)
```

Method **I7** is called when Atlas requests the hypothetical size of an instance given a set number of elements *num_elements* and a supplied initialization string *params* . This method is optional.

Where:

- **num_elements**: Number of elements in the hypothetical instance. This value must be non-zero.
- **params:** Optional handle to the hypothetical instance creation parameters. This is typically the parameter supplied to the */Create* command. (See */Create*)
- **handshake**: Discretionary validation token to be reviewed by this method.

RETURN:

- Size of the hypothetical instance based on supplied number of elements and instantiation parameters.
- 0 upon failure to compute the instance size.

GET THE VALUE OF AN INSTANCE (I8):

```
titan_result_t AtlasSim_GetValue(TITAN_HANDLE64
instance, TITAN_HANDLE64 element, titan_value_t &value,
TITAN_HANDLE64 handshake = 0)
```

Method **I8** is called when Atlas requests that an instance value be retrieved. This method is typically trigged by a *M5:GetValue* call in the cognitive space.

Where:

- **instance**: The instance whose value will be retrieved.
- **element**: The optional instance element whose value will be retrieved (or 0 if the instance has no elements).
- **value**: The output configuration value structure, where:
  - format_handle: The name handle of the format of the value being retrieved.
  - *_value: The output atomic value retrieved from the instance, such that:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance value has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SET THE VALUE OF AN INSTANCE (I9):

```
titan_result_t AtlasSim_SetValue(TITAN_HANDLE64
instance, TITAN_HANDLE64 element, titan_value_t &value,
TITAN_HANDLE64 handshake = 0)
```

Method **I9** is called when Atlas requests that an instance value be updated. This method is typically trigged by a *M6:SetValue* call in the cognitive space.

Where:

- **instance**: The instance whose value will be retrieved.
- **element**: The optional instance element whose value will be retrieved (or 0 if the instance has no elements).
- **value**: The output configuration value structure, where:
  - format_handle: The name handle of the format of the value being retrieved.
  - *_value: The output atomic value retrieved from the instance, such that:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance value has been successfully set.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SET THE NUMBER OF ELEMENTS IN AN INSTANCE (IA):

```
titan_result_t AtlasSim_SetElementCount(TITAN_HANDLE64
instance, TITAN_ULONG num_elements, TITAN_HANDLE64
handshake = 0)
```

Method **IA** is called when Atlas requests (dynamically) that the number of elements in an instance be updated.

This method is only invoked if it is supported by the axiom.

Where:

- **instance:** Handle of the instance whose element count will be set.
- **num_elements:** The new number of elements requested for the supplied instance. This value must be non-zero.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance's element count was successfully updated.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET THE NUMBER OF ELEMENTS IN AN INSTANCE (IB):

```
TITAN_ULONG AtlasSim_GetElementCount(TITAN_HANDLE64
instance, TITAN_HANDLE64 handshake = 0)
```

Method **IB** is called when Atlas needs to know how many elements reside within a particular instance.

Where:

- **instance:** Handle of the instance whose number of elements will be retrieved.
- **handshake**: Discretionary validation token to be reviewed by this method.

RETURN:

- Number of elements in the instance upon successful operation.
- 0 upon failure to retrieve the number of elements in the instance.

RETRIEVE AN INSTANCE ELEMENT (IC):

```
TITAN_HANDLE64 AtlasSim_GetElement(TITAN_HANDLE64
instance, TITAN_ULONG element_index, TITAN_HANDLE64
handshake = 0)
```

Method **IC** is called when/if Atlas requests a particular element within an instance.

The element would be reference by index.

Where:

- **instance:** An existing handle of the instance whose element will be retrieved.
- **element_index**: Index of the element retrieved. This value must be less than the number of elements returned by *AtlasSim_GetElementCount* method.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- Handle to the requested instance element upon successful retrieval.
- 0 upon failure to retrieve the instance element.

RETRIEVE THE FIRST ELEMENT IN AN INSTANCE (ID):

```
AtlasSim_GetFirstElement(TITAN_HANDLE64 instance,
TITAN_HANDLE64 handshake = 0)
```

Method **ID** is called when Atlas requests the first element in an instance.

There are cases where the number of elements is unknown. This is particularly true if the instance size is dynamic, or if the elements are part of a stream as opposed to an array. In such case, Atlas would request the first element followed by a traversal of the remaining elements using ***AtlasSim_GetNextElement****.*

Where:

- **instance**: Handle of the instance whose first element will be retrieved.
- **handshake**: Discretionary validation token to be reviewed by this method.

RETURN:

- Handle to the first instance element upon successful retrieval.
- 0 upon failure to retrieve the instance element.

RETRIEVE THE NEXT ELEMENT IN AN INSTANCE (IE):

```
TITAN_HANDLE64 AtlasSim_GetNextElement(TITAN_HANDLE64
instance, TITAN_HANDLE64 element, TITAN_HANDLE64
handshake = 0)
```

Method **IE** is used when, given a particular element, Atlas requests a subsequent element within an instance.

Where:

- **instance:** Handle of the instance whose element will be retrieved.
- **element:** Element that precedes the requested element within the instance.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- Handle to the requested next instance element upon successful retrieval.
- 0 upon failure to retrieve the instance element.

ADD AN ELEMENT TO AN INSTANCE (IF):

```
TITAN_HANDLE64 AtlasSim_AddElement(TITAN_HANDLE64
instance, TITAN_HANDLE64 handshake = 0)
```

Method **IF** is used when/if Atlas requests to add an element to an existing instance if that instance's managing axiom allows it.

Where:

- **instance:** Existing handle of the instance to be destroyed.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- Handle to the added element within the instance upon successful operation.
- 0 upon failure to add the new instance element.

REMOVE AN EXISTING ELEMENT FROM AN INSTANCE (IG):

```
titan_result_t AtlasSim_RemoveElement(TITAN_HANDLE64
instance, TITAN_HANDLE64 element, TITAN_HANDLE64
handshake = 0)
```

Method **IG** is called when Atlas requests an existing instance to be destroyed.

Where:

- **instance:** Handle of the instance whose element will be destroyed.
- **element:** Handle of the element that will be destroyed.
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN:

- **TITAN_RESULT_OK:** The instance element was successfully removed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

III. Interpretation Interface:

A concept's available aspects are described in the definition where:

DO ASPECT:

The *Do* aspect of a concept describes the behavior of its instances relative to itself or to other concepts at a specific time. Behavior is asymmetric. Examples (Eat, Eaten)

BE ASPECT:

The *Be* aspect of a concept describes the condition of one of its instances at a specific time. State is asymmetric. Examples (Taller, Shorter)

HAVE ASPECT:

The *Have* aspect of a concept describes the relationship of its instances relative to instances of other concepts at a specific time. Relationship is asymmetric. Examples (Mother, Daughter)

SIMULATION METHODS (M*):

```
titan_probe_result_t
AtlasSimulationMethod(TITAN_HANDLE64 user_handle,
TITAN_HANDLE64 directive_handle,
titan_auxiliary_parameters_t directive_params,
TITAN_HANDLE64 handshake)
```

The **M*** methods are called when Atlas needs to simulate an instance in an environment.

The simulation methods registered with an instance's interpretation should map to aspects of a definition during knowledge base attachment. ***(See KB_Attach)***

Where:

- **user_handle**: Handle to the user information that was supplied during registration of the method (See *C3:Do*, *C4:Be*, *C5:Have*).
- **directive_handle**: Handle of the directive of the simulation requested.
- **handshake**: Discretionary validation token sent to the method or 0 for no token.
- **directive_params**: A sub-component of the directive that is exposed as a C structure, where:
  - **aux_type**: This is the simulated aspect type. Available types are *TITAN_AUX_DO*, *TITAN_AUX_BE*, and *TITAN_AUX_HAVE* corresponding to the *Do*, *Be*, and *Have* methods respectively.
  - **expression_type**: This is the expression type bitmap, where:
    - **TITAN_EXP_TYPE_QUERY**: This bit is true when the expression is a question.
    - **TITAN_EXP_TYPE_STATEMENT**: This bit is true when the expression is a statement, and not an imperative.
    - **TITAN_EXP_TYPE_PROSPECTION**: Expression states possible outcomes (can).
    - **TITAN_EXP_TYPE_PREDICTION**: Expression states predicated outcomes (should/must).
    - **TITAN_EXP_TYPE_OBJECTIVE**: Expressions states desired outcomes (want/need).
    - **TITAN_EXP_TYPE_WHO**: Expression is requesting agent information about the directive.
    - **TITAN_EXP_TYPE_WHEN**: Expression is requesting time information about the directive.
    - **TITAN_EXP_TYPE_WHERE**: Expression is requesting space information about the directive.
    - **TITAN_EXP_TYPE_WHAT**: Expression is requesting reference/verb information about the directive.
    - **TITAN_EXP_TYPE_WHY**: Expression is requesting cause of the directive.

    - **TITAN_EXP_TYPE_HOW**: Expression is requesting process of the directive.
    - **TITAN_EXP_TYPE_PREDICATE**: Expression is requesting veracity of the directive.
- **aux_tense**: This is the tense of the simulation, where:
    - **TITAN_TENSE_INFINITIVE:** Infinitive tenses are used to either state or query a generalization about the expression regardless of time.
    - **TITAN_TENSE_IMPERATIVE:** Imperative tense is a command in the present.
    - **TITAN_TENSE_PRESENT:** Present tense (expression ‘is’).
    - **TITAN_TENSE_PAST:** Past tense (expression ‘was’).
    - **TITAN_TENSE_FUTURE:** Future tense (expression ‘will’).
    - **TITAN_TENSE_PAST_PAST:** Past-Past (expression ‘had been’).
    - **TITAN_TENSE_PAST_FUTURE:** Past-Future (expression ‘would have been’).
    - **TITAN_TENSE_FUTURE_PAST:** Future-Past (expression ‘will have been’).
    - **TITAN_TENSE_FUTURE_FUTURE:** Future-Future (expression ‘will later be).
- **valence**: The valence index is the number of objects in the directive (See Directive Valence).
- ***handshake***: Discretionary validation token to be reviewed by this method.

RETURN: The simulation method return value of 0 should be interpreted as an *unknown answer* to the expression. It is still a valid answer if the simulation axiom does not know the answer or does not want to affect the response of a simulation method. The appropriate response value to a failed handshake or authentication by the simulation method is also 0. In all cases, simulations method results should be interpreted based on the expression type as described below:

- **TITAN_EXP_TYPE_PREDICATE**:
    - **-100:** NEVER
    - **-50:** NO
    - **0:** DON'T KNOW
    - **+50:** YES
    - **+100:** ALWAYS

- **TITAN_EXP_TYPE_PROSPECTION**:
    - **-100:** CANNOT
    - **-50:** MAY NOT
    - **0:** DON'T KNOW
    - **+50:** MAY

- **+100:** CAN

- **TITAN_EXP_TYPE_PREDICTION**:
  - **-100:** MUST NOT
  - **-50:** SHOULD NOT
  - **0:** DON'T KNOW
  - **+50:** SHOULD
  - **+100:** MUST

- **TITAN_EXP_TYPE_OBJECTIVE**:
  - **-100:** DON'T WANT
  - **-50:** DON'T NEED
  - **0:** DON'T KNOW
  - **+50:** WANT
  - **+100:** NEED

Atlas

Cognitive API Bundle

Cognitive API Bundle:

Introduction:

The Cognitive API handles the functionality exposed to concepts by the cognitive environment.

The cognitive environment can host one or multiple interfaces advertised by Atlas or third-party software.

Interfaces are universal and transcend zones within a domain.

Cognitive APIs are grouped into the following:

- **Cognitive Interface (Cx):** Manages the cognitive environment.
- **Domain Interface (Dx):** Manages domains, zones, and simulations.
- **Agent Interface (Ax):** Manages agents within the cognitive space.
- **Directive Interface (Xx)**: Inspects incoming simulation directives.
- **Label Interface (Lx)**: Registers and retrieves registered lexicon terms and labels.
- **Ledger Interface (Vx)**: Manages natural language expressions within the cognitive space.

The follow method groups are called by the axiom and resolved by Atlas:

- Cx methods handle cognitive functionality.
- Dx methods handle domain functionality.
- Ax methods handle agent functionality.
- Xx methods handle directive functionality.
- Lx methods handle label functionality.
- Vx methods handle conversation functionality.

Annotations:

In the overview diagram, the Cognitive API methods are confined to the following delimiter.

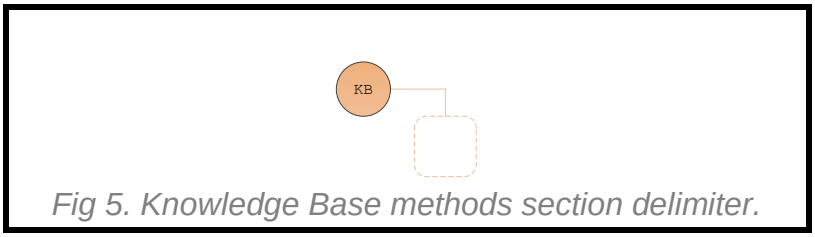


Fig 5. Knowledge Base methods section delimiter.

Cognitive API Interfaces

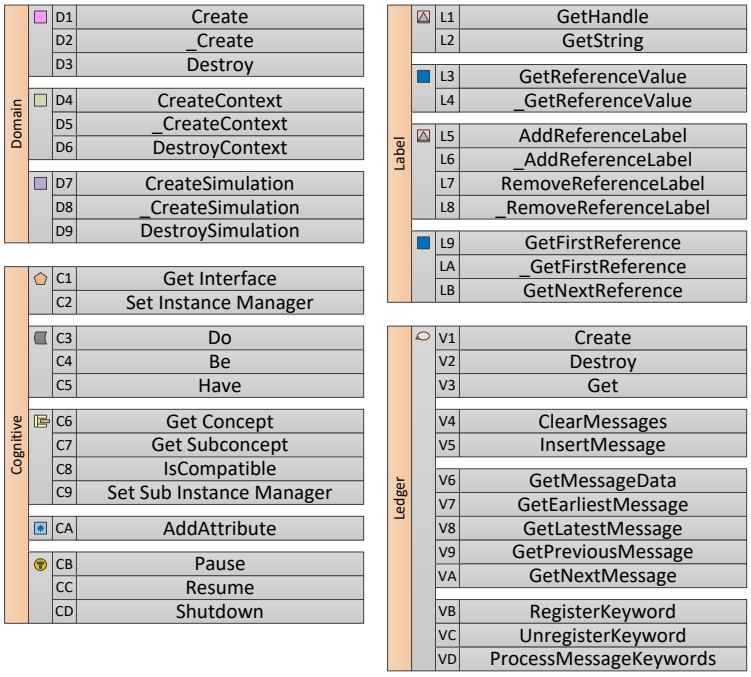


Fig 6. Callouts for the knowledge base methods and components.

I. Cognitive Interface:

Information:

As described in the definition, Atlas's concepts have three main aspects: **Do**, **Be**, and **Have**.

These aspects represent states, behaviors, and relationship evaluations that can be performed by the interpretation axioms.

During knowledge base attachment, the interpretation is expected to map its axiom methods to the Do/Be/Have aspects referenced in the definition.

Atlas will evaluate instance data using these axioms during simulation.

Cognitive Interface:

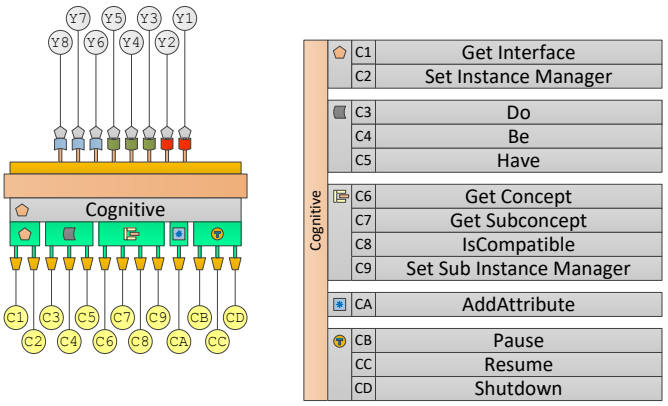


Fig 7. Cognitive interface diagram and table.

ACCESS AN INTERFACE (C1):

```
TITAN_POINTER GetInterface(TITAN_HANDLE64 interface_id,
TITAN_HANDLE64 interface_name_handle = 0,
TITAN_HANDLE64 handshake = 0)
```

Method **C1** is called to access an advertised interface in the cognitive space.

Where:

- **interface_id**: ID of the interface. This is a universal number that is issued by Atlas to third parties.
- ***interface_name_handle***: Handle of the name of the interface requested (See Labels).
- ***handshake_handle***: Discretionary token sent to the interface requested for validation.

RETURN:

- Pointer to the interface requested upon successful retrieval.
- NULL upon failure to retrieve the interface.

REGISTER AN INSTANCE MANAGER (C2):

```
titan_result_t
SetInstantiator(AtlasInstantiatorInterface
instantiator)
```

Method **C2** is called to register an instance manager for an interpreted concept.
A concept is interpreted in code. The data that is interpreted is managed by a registered instance manager, also known as an instantiator (See Instance Manager).

Where:

- **instantiator**: The class handling the instantiator. For the most part, this would the axiom handling the interpretation. The instantiator must be a TitanAxiom-derived class, but it does not have to be the axiom interpreting the concept.

- **RETURN**:
  - **TITAN_RESULT_OK**: The instantiator was successfully assigned.
  - **TITAN_RESULT_*****: See Atlas Errors for more information.

REGISTER A SIMULATION METHOD (C3, C4, C5):

```
titan_result_t Do(TITAN_UINT perspective, const
TITAN_STRING method_name, TITAN_UINT simulation_index,
TITAN_HANDLE64 user_data, AtlasSimulationMethod method,
TITAN_BITMAP64 method_flags = 0)
```

```
titan_result_t Be(TITAN_UINT perspective, const
TITAN_STRING method_name, TITAN_UINT simulation_index,
TITAN_HANDLE64 user_data, AtlasSimulationMethod method,
TITAN_BITMAP64 method_flags = 0)
```

```
titan_result_t Have(TITAN_UINT perspective, const
TITAN_STRING method_name, TITAN_UINT simulation_index,
TITAN_HANDLE64 user_data, AtlasSimulationMethod method,
TITAN_BITMAP64 method_flags = 0)
```

Methods **C3, C4, C5** are called when a concept interpretation needs to advertise.

Three methods are available, one for each corresponding aspect of Do, Be, and Have.

During KB_Attach, it is expected of the interpretation to register the simulation methods for the aspects advertised in the definition of the concept. An interpretation is not necessarily required to handle all the advertised methods.

Where:

- **perspective**: The perspective of the simulation (See *Directive*). The method simulates:
  - **TITAN_PERSPECTIVE_SUBJECT**: The subject of the directive.
  - **TITAN_PERSPECTIVE_OBJECT**: The object of the directive.
  - **TITAN_PERSPECTIVE_OBSERVER**: An observer of the directive.
- **method_name:** The matching aspect name registered in the definition file (*See Definition*).
- **simulation_index:** Simulation level of detail. This value is reserved for future use, and must be 0 for now.
- **user_data**: Discretionary handle which typically corresponds to the class pointer of the axiom managing this simulation.
- **method:** Actual C++ API method handling the simulation. This is the true interpretation axiom.

- ***method_flags*:** Bitmap of method flags. This value is reserved for future use, and must be 0 for now.

RETURN:

- **TITAN_RESULT_OK**: The method was successfully assigned.
- **TITAN_RESULT_***: See Atlas Errors for more information.

RETRIEVE A CONCEPT (C6):

```
TITAN_HANDLE64 GetConcept(TITAN_HANDLE64
private_reference, const TITAN_STRING concept_name,
TITAN_HANDLE64 handshake = 0)
```

Method **C6** is called, after *a KB_Attach*, when one concept seeks another concept in the same domain.

This is necessary if concepts need to collaborate.

Where:

- **private_reference**: Private handle of the reference requesting the concept (See *Reference*).
- **concept_name:** The official or soft-synonym concept name (See *Definition*, *Manifest*).
- ***handshake***: Discretionary validation token sent to the domain manager hosting the concept (See *Domain*).

RETURN:

- Handle to the concept retrieved upon successful operation.
- 0 upon failure to retrieve the requested concept.

RETRIEVE A SUB-CONCEPT (C7):

```
TITAN_HANDLE64 GetSubconcept(const TITAN_STRING
sub_concept_name)
```

Under normal circumstances, an interpretation would need to have access to its subordinate concepts in order to delegate or configure its concept hierarchy. Method **C7** is called when a concept needs to retrieve the handles of its registered sub-concepts.

Where:

- **sub_concept_name:** The official subordinate concept name (See *Manifest*).

RETURN:

- Handle to the concept retrieved upon successful operation.
- 0 upon failure to retrieve the requested concept.

CHECK CONCEPT COMPATIBILITY (C8):

```
TITAN_BOOL IsCompatible(TITAN_HANDLE64 reference)
```

Method **C8** is called when, upon receiving a directive, an axiom tries to determine if its existing reference is compatible with the request.

Where:

- **reference**: Public handle of the reference probed (See *Reference*).

RETURN:

- TRUE if the reference is compatible with the calling concept.
- FALSE if the reference is not compatible with the calling concept.

REGISTER AN INSTANCE MANAGER OF A SUBORDINATE CONCEPT (C9):

```
titan_result_t SetSubInstantiator(const TITAN_STRING
sub_concept_name, TITAN_HANDLE64 sub_concept)
```

Interpretations are responsible for the data management of their subordinate concepts. Method **C9** is called to register the instantiator of a subordinate concept.

Where:

- **sub_concept_name:** The official subordinate concept name (See *Manifest*).
- **sub_concept**: Handle to the sub-concept instance manager.

RETURN:

- **TITAN_RESULT_OK**: The subordinate concept instantiator was successfully assigned.
- **TITAN_RESULT_*****: See Atlas Errors for more information.

ADD CONCEPT ATTRIBUTES (CA):

```
TITAN_POINTER AddAttribute(titan_attribute_scope_t
attribute_scope, TITAN_HANDLE64 attribute_id,
TITAN_HANDLE64 handshake = 0)
```

Method **CA** is called when a concept requests one or more attributes.

Atlas has internal knowledge of some concepts, also known as attributes, to handle features common to most concepts such as position, scale, shape, temperature, age, and so on (See Attributes). Though most of these attributes are optional, some are necessary for spatial reasoning.

Where:

- **attribute_scope**: Scope of the feature as it relates to the concept, where:
  - **TITAN_ATTRIB_SCOPE_CONCEPT**: Attribute probe gives the same result for the entire concept, regardless of instance probed.
  - **TITAN_ATTRIB_SCOPE_INSTANCE**: Attribute probe gives the same result for all elements in the probed instance.
  - **TITAN_ATTRIB_SCOPE_ELEMENT**: Attribute probe gives the different results for each element in the probed instance.
- **attribute_id:** The official attribute ID (See *Attributes*).
- ***handshake***: Discretionary validation token sent to the simulation manager hosting the concept (See *Simulation*).

RETURN:

- Pointer to the C API interface managing the added attribute upon success.
- NULL upon failure to add a simulation attribute.

MANAGE ATLAS (CB, CC, CD):

```
titan_result_t Pause(TITAN_HANDLE64 handshake = 0)
```

```
titan_result_t Resume(TITAN_HANDLE64 handshake = 0)
```

```
titan_result_t ShutDown(TITAN_HANDLE64 handshake = 0)
```

Methods **CB, CC, and CD** are used to alert Titan eXchange that a domain is coming Online or Offline.

Where:

- ***handshake***: Discretionary validation token sent to the cognitive exchange manager hosting the domain (See *Titan eXchange*).

II. Domain Interface:

Information:

Concepts can create a domain for Atlas to manage. This is done in one of two ways:

- A reference creating an agent will automatically join the newly created agent.
- A reference creating a domain will automatically join the newly created domain.

Once a domain is created, Developers can manage access to it through a Domain Manager.

Annotation:

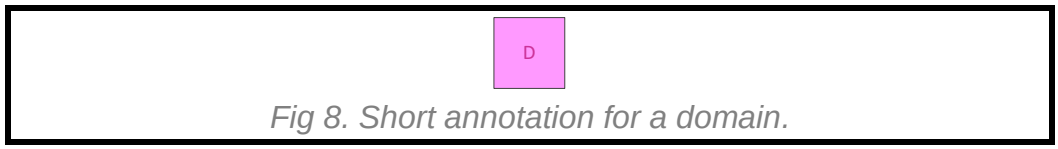


Fig 8. Short annotation for a domain.

Domain Interface:

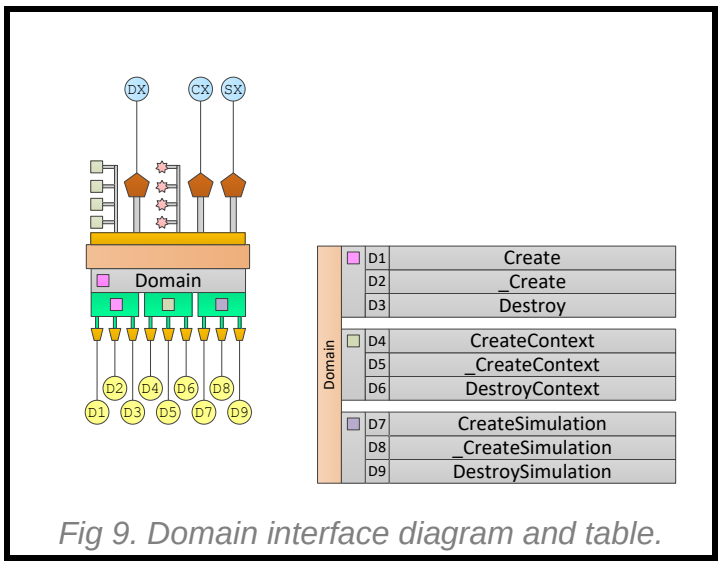


Fig 9. Domain interface diagram and table.

CREATE A DOMAIN (D1):

```
TITAN_HANDLE64 Create(TITAN_HANDLE64 caller_reference,
const TITAN_STRING domain_name, const TITAN_STRING
zone_name, TITAN_UINT num_simulations, const
TITAN_STRING manifest, const titan_manager_t
*domain_manager = NULL, const titan_manager_t
*zone_manager = NULL, TITAN_HANDLE64 domain_handshake =
0)
```

Method **D1** is used to create a domain and a context. Once the domain is created, a manifest is loaded, and a script is executed. The script and the manifest names are also specified in this method.

Where:

- **caller_reference**: Private handle of the reference creating the domain.
- **domain_name**: Name of the domain to be created.
- **zone_name**: Name of the zone to be created upon domain creation.
- **num_simulations**: Reserved (must be 0).
- **manifest**: Path to the manifest to be loaded upon domain creation (See *Manifest*).
- **script**: Path to the script to be executed upon domain creation.
- ***domain_manager***: Optional data structure of the registering manager, where:
  - **Domain**: Pointer to the domain manager method (See D*X: Domain Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***zone_manager***: Optional data structure of the registering manager, where:
  - **Zone**: Pointer to the zone manager method (See *CX: Zone Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the calling reference.

RETURN:

- Handle of the created or existing domain upon successful operation.
- 0 upon failure to create a domain.

CREATE A DOMAIN (D2):

```
TITAN_HANDLE64 _Create(TITAN_HANDLE64 caller_reference,
TITAN_HANDLE64 domain_name_handle, TITAN_HANDLE64
zone_name_handle, TITAN_UINT num_simulations, const
TITAN_STRING manifest, const TITAN_STRING script, const
titan_manager_t *domain_manager = NULL, const
titan_manager_t *zone_manager = NULL, TITAN_HANDLE64
domain_handshake = 0)
```

Method **D2** is used to create a domain and a context. Once the domain is created, a manifest is loaded, and a script is executed. The script and the manifest names are also specified in this method.

Where:

- **caller_reference**: Private handle of the reference creating the domain.
- **domain_name_handle**: Handle to the name of the domain to be created.
- **zone_name_handle**: Handle to the name of the zone to be created upon domain creation.
- **num_simulations**: Reserved (must be 0).
- **manifest**: Path to the manifest to be loaded upon domain creation (See *Manifest*).
- **script**: Path to the script to be executed upon domain creation.
- ***domain_manager***: Optional data structure of the registering manager, where:
  - **Domain**: Pointer to the domain manager method (See D*X: Domain Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***zone_manager***: Optional data structure of the registering manager, where:
  - **Zone**: Pointer to the zone manager method (See *CX: Zone Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the calling reference.

RETURN:

- Handle of the created or existing domain upon successful operation.
- 0 upon failure to create a domain.

DESTROY A DOMAIN (D3):

```
titan_result_t Destroy(TITAN_HANDLE64 caller_reference,
TITAN_HANDLE64 domain_handle, TITAN_HANDLE64
domain_handshake = 0)
```

Method **D3** is called when an existing domain needs to be destroyed.

NOTE: *Upon destruction, everything inside the domain is also destroyed. Atlas will destroy any referenced instances within the domain, but non-referenced instances will need to be destroyed by their hosting concepts prior to detachment (See T2:KB_Detach).*

Where:

- **caller_reference**: Private handle of the reference destroying the domain.
- **domain_handle**: Handle of the domain being destroyed.
- ***domain_handshake***: Optional handshake of the request to be processed by the domain being destroyed.

RETURN:

- **TITAN_RESULT_OK**: The domain has been successfully destroyed.
- **TITAN_RESULT_***: See Atlas Errors for more information.

DOMAIN MANAGER (DX:ATLAS DOMAIN MANAGER METHOD):

```
TITAN_METHOD_RESULT AtlasDomainManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

The domain manager is the axiom-supplied method responsible for validating domain requests. This method is registered during domain creation (*D1:Create*, *D2:_Create*). All component calls that require a domain handshake will be evaluated by this manager method prior to processing those requests.

Where:

- **caller_reference**: Private handle of the reference making the domain request.
- **request_type**: Type of request, where:
  - **TITAN_DOMAIN_DESTROY**: Request domain destruction.
  - **TITAN_DOMAIN_ENTER**: Request for an agent to enter the domain.
  - **TITAN_DOMAIN_EXIT**: Request for an agent to exit the domain.
  - **TITAN_DOMAIN_CREATE_ZONE**: Request zone creation.
  - **TITAN_DOMAIN_DESTROY_ZONE**: Request zone destruction.
  - **TITAN_DOMAIN_CREATE_SIMULATION**: Request simulation creation.
  - **TITAN_DOMAIN_DESTROY_SIMULATION**: Request simulation destruction.
  - **TITAN_DOMAIN_LEARN_CONCEPT**: Request concept to be learned.
  - **TITAN_DOMAIN_FORGET_CONCEPT**: Request concept to be forgotten.
  - **TITAN_DOMAIN_READ_MANIFEST**: Request *Atlas* to read a manifest. (See *Manifest*).
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *D1:Create*, *D2:_Create*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the domain to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the domain not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

CREATE A ZONE (D4):

```
TITAN_HANDLE64 CreateZone(TITAN_HANDLE64
caller_reference, const TITAN_STRING zone_name, const
titan_manager_t *zone_manager = NULL, TITAN_HANDLE64
domain_handshake = 0)
```

Method **D4** is called to create a context zone in the domain hosting the calling reference.

Where:

- **caller_reference**: Private handle of the reference creating the zone.
- **zone_name**: Name of the zone to be created.
- ***zone_manager***: Optional data structure of the registering manager, where:
  - **Zone**: Pointer to the zone manager method (See *CX: Zone Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the new zone.

RETURN:

- Handle of the zone created upon successful operation.
- 0 upon failure to create a zone.

CREATE A ZONE (D5):

```
TITAN_HANDLE64 _CreateZone(TITAN_HANDLE64
caller_reference, TITAN_HANDLE64 zone_name_handle,
const titan_manager_t *zone_manager = NULL,
TITAN_HANDLE64 domain_handshake = 0)
```

Method **D5** is called to create a context zone in the domain hosting the calling reference.

Where:

- **caller_reference**: Private handle of the reference creating the zone.
- **zone_name_handle**: Handle to the name of the zone to be created.
- ***zone_manager***: Optional data structure of the registering manager, where:
  - **Zone**: Pointer to the zone manager method (See *CX: Zone Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the new zone.

RETURN:

- Handle of the zone created upon successful operation.
- 0 upon failure to create a zone.

DESTROY A ZONE (D6):

```
titan_result_t DestroyZone(TITAN_HANDLE64
caller_reference, TITAN_HANDLE64 zone_handle,
TITAN_HANDLE64 zone_handshake = 0)
```

Method **D6** is called to destroy a zone specified by handle on behalf of a calling reference. The zone handle should have been received by method *D4:CreateZone*, or method *D5:_CreateZone*.

Where:

- **private_reference**: Private handle of the reference destroying the zone.
- **zone_handle**: Handle of the zone being destroyed.
- ***zone_handshake***: Optional handshake of the request to be processed by the zone being destroyed.

RETURN:

- **TITAN_RESULT_OK**: The specified context zone has been successfully destroyed.
- **TITAN_RESULT_***: See Atlas Errors for more information.

Usage:

The **D6** method is called when a zone needs to be destroyed.

NOTE: ***A zone does not destroy the data it is hosting upon destruction***.

ZONE MANAGER (CX:ATLAS ZONE MANAGER METHOD):

```
TITAN_METHOD_RESULT AtlasZoneManager(TITAN_HANDLE64
caller_reference, titan_zone_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

The Zone manager is the axiom-supplied method responsible for validating zone requests. This method is registered during zone creation (D4:Create, D5:_Create). All component calls that require a zone handshake will be evaluated by this manager method prior to processing those requests.

Method **CX** is called when a request that requires a handshake is evaluated by a zone manager prior to processing a request.

Info:

- Zone managers are responsible for validating zone requests.
- Zone manager is registered during zone creation (D1:Create, D2:_Create, D4:CreateZone, D5:_CreateZone).

Where:

- **caller_reference**: Private handle of the reference making the zone request (See *Reference*).
- **request_type**: Type of request, where:
  - **TITAN_ZONE_DESTROY**: Request zone destruction.
  - **TITAN_ZONE_ENTER**: Request to enter a zone by an agent.
  - **TITAN_ZONE_EXIT**: Request to exit a zone by an agent.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *D1:Create*, *D2:_Create, D4:CreateZone*, *D5:_CreateZone*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the zone to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the zone not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

CREATE A SIMULATION (D7):

```
TITAN_HANDLE64 CreateSimulation(TITAN_HANDLE64
caller_reference, const TITAN_STRING simulation_name,
TITAN_ULONG complexity, const titan manager_t
*simulation_manager = NULL, TITAN_HANDLE64
domain_handshake = 0)
```

Method **D7** is called to create a simulation within a domain. This is the string-based version of **D8**.

Where:

- **caller_reference**: Private handle of the reference creating the simulation.
- **simulation_name**: Name of the simulation to be created.
- **complexity**: Complexity of the simulation. This value determines size of the trail and frequency of the simulation.
- **simulation_manager**: Optional data structure of the registering manager, where:
- **Simulation**: Pointer to the simulation manager method (See S*X: Simulation Manager*).
- **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the new simulation.

RETURN:

- Handle of the created or existing simulation upon successful operation.
- 0 upon failure to create a simulation.

CREATE A SIMULATION (D8):

```
TITAN_HANDLE64 _CreateSimulation(TITAN_HANDLE64
caller_reference, TITAN_HANDLE64
simulation_name_handle, TITAN_ULONG complexity, const
titan manager_t *simulation_manager = NULL,
TITAN_HANDLE64 domain_handshake = 0)
```

Method **D8** is called to create a simulation within a domain. This is the token-based version of **D7**.

Where:

- **caller_reference:** Private handle of the reference creating the simulation.
- **simulation_name_handle**: Handle to the name of the simulation to be created.
- **complexity**: Complexity of the simulation. This value determines size of the trail and frequency of the simulation.
- ***simulation_manager***: Optional data structure of the registering manager, where:
  - **Simulation**: Pointer to the simulation manager method (See S*X: Simulation Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Optional handshake of the request to be processed by the hosting domain of the new simulation.

RETURN:

- Handle of the created or existing simulation upon successful operation.
- 0 upon failure to create a simulation.

DESTROY A SIMULATION (D9):

```
titan_result_t Destroy(TITAN_HANDLE64 caller_reference,
TITAN_HANDLE64 simulation_handle, TITAN_HANDLE64
simulation_handshake = 0)
```

Method **D9** is called to destroy a simulation by handle. The simulation must exist in the domain hosting the calling reference.

NOTE: A simulation does not destroy the data it is hosting upon destruction.

Where:

- **caller_reference**: Private handle of the reference destroying the simulation.
- **simulation_handle**: Handle of the simulation being destroyed.
- ***simulation_handshake***: Optional handshake of the request to be processed by the simulation being destroyed.

RETURN:

- **TITAN_RESULT_OK**: The simulation has been successfully destroyed.
- **TITAN_RESULT_***: See Atlas Errors for more information.

SIMULATION MANAGER (SX: ATLAS SIMULATION MANAGER METHOD):

```
TITAN_METHOD_RESULT MySimulationManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

Method **SX** is called when a simulation manager needs to validate simulation requests prior to processing. This manager regulates access to the simulation features by reviewing the calling reference, the request type, and the supplied user data.

Where:

- **caller_reference**: Private handle of the reference making the domain request (See Reference).
- **request_type:** Type of request where:
  - **TITAN_SIMULATION_DESTROY:** Request simulation destruction.
  - **TITAN_SIMULATION_ENTER:** Request for a context agent to join the simulation.
  - **TITAN_SIMULATION_LEAVE:** Request for a context agent to leave the simulation.
  - **TITAN_SIMULATION_CREATE_AGENT:** Request an Atlas clone to be created.
  - **TITAN_SIMULATION_DESTROY_AGENT:** Request an Atlas clone to be destroyed.
  - **TITAN_SIMULATION_JOIN_AGENT:** Request from a context agent to join an Atlas clone.
  - **TITAN_SIMULATION_LEAVE_AGENT:** Request from a context agent to leave an Atlas clone.
- **user_data:** Discretionary 64-bit value supplied by the user during registration (See D1:Create, D2:_Create).
- **message_data:** Message data is dependent on message type. (See Manager Messages).
- ***handshake:*** Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the domain to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the domain not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

III. Directive Interface:

Information:

Directives are the main data payloads sent to the axioms during simulation.

Directives are created using the *AA:Tell* method.

Annotation:

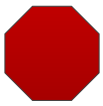
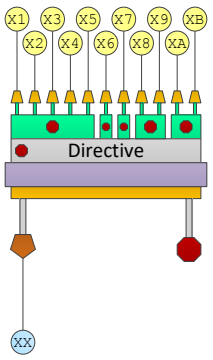


Fig 10. Short annotation of a directive.

Directive Interface:



Directive		
	X1	GetNumComponents
	X2	GetComponent
	X3	GetNextComponent
	X4	GetComponentByName
	X5	RephraseComponent
	X6	GetAction
	X7	GetState
	X8	SetComponentHandshake
	X9	ClearHandshakes
	XA	Remember
	XB	Forget

Fig 11. Directive interface diagram and table.

RETRIEVE A DIRECTIVE COMPONENT BY ID (X1):

```
TITAN_HANDLE64 Get(TITAN_HANDLE64 directive,
titan_directive_component_t component_id,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **X1** is used to retrieve the first element of a directive component specified by ID.

Where:

- **directive:** Handle of the directive whose component is being retrieved.
- **component_id:** Component identification, where:
  - **Author**: The component is an author of the directive.
  - **Subject**: The component is a subject of the directive.
  - **Object**: The component is an object of a directive (transitive).
  - **Tool**: The component is a tool object (using, with).
  - **Time**: The component is a time manifold.
  - **Space**: The component is space manifold.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **value**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the value.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.

        - **short_value**: Relation amount data is a short signed 16-bit integer.
    - **amount**: A titan_amount_t structure describing the amount of the reference such that:
        - **format_handle**: Handle of the amount format (type or units).
        - ***_value**: Union of the amount where:
            - **handle_value**: Relation amount data is a 64-bit handle.
            - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
            - **long_value**: Relation amount data is a long signed 64-bit integer.
            - **double_value**: Relation amount data is a double-precision floating point.
            - **float_value**: Relation amount data is a single-precision floating point.
            - **string_value**: Relation amount data is a null-terminate character string.
            - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
            - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
            - **uint_value**: Relation amount data is an unsigned 32-bit integer.
            - **int_value**: Relation amount data is a signed 32-bit integer.
            - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
            - **short_value**: Relation amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference.
    - **element**: Returned element of the reference.
- ***query_flags***: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.

RETURN:

- Handle of the directive component upon successful operation.
- 0 upon failure to retrieve the component.

RETRIEVE NEXT DIRECTIVE COMPONENT (X2):

```
TITAN_HANDLE64 GetNext(TITAN_HANDLE64 directive,
TITAN_HANDLE64 component_handle, titan_reference_info_t
&reference_info, TITAN_BITMAP64 query_flags =
TITAN_GET_DEFAULT)
```

The **X2** method is called to retrieve the next component under the same ID based on supplied component handle.

Where:

- **directive**: Handle of the directive whose component is being retrieved.
- **component_handle**: Handle of the component whose successor will be retrieved (previous handle).
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **value**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the value.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).

    - ***_value**: Union of the amount where:
        - **handle_value**: Relation amount data is a 64-bit handle.
        - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
        - **long_value**: Relation amount data is a long signed 64-bit integer.
        - **double_value**: Relation amount data is a double-precision floating point.
        - **float_value**: Relation amount data is a single-precision floating point.
        - **string_value**: Relation amount data is a null-terminate character string.
        - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
        - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
        - **uint_value**: Relation amount data is an unsigned 32-bit integer.
        - **int_value**: Relation amount data is a signed 32-bit integer.
        - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
        - **short_value**: Relation amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference
    - **element**: Returned element of the reference
- ***query_flags***: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.

RETURN:

- Handle of the next directive component upon successful operation.
- 0 upon failure to retrieve the component.

RETRIEVE A DIRECTIVE COMPONENT BY RELATION (X3):

```
titan_result_t Get(TITAN_HANDLE64 directive,
TITAN_HANDLE64 relation, titan_reference_info_t
&reference_info, TITAN_BITMAP64 query_flags =
TITAN_GET_DEFAULT)
```

Method **X3** is used to retrieve a relation by name handle from supplied directive.

Where:

- **directive**: Handle of the directive whose relation is being retrieved.
- **relation**: Handle of the relation name.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **value**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the value.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.

        - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
        - **long_value**: Relation amount data is a long signed 64-bit integer.
        - **double_value**: Relation amount data is a double-precision floating point.
        - **float_value**: Relation amount data is a single-precision floating point.
        - **string_value**: Relation amount data is a null-terminate character string.
        - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
        - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
        - **uint_value**: Relation amount data is an unsigned 32-bit integer.
        - **int_value**: Relation amount data is a signed 32-bit integer.
        - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
        - **short_value**: Relation amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference.
    - **element**: Returned element of the reference.
- ***query_flags***: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.

RETURN:

- **TITAN_RESULT_OK:** The component requested has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REPHRASE A DIRECTIVE COMPONENT (X4):

```
titan_result_t Rephrase(TITAN_HANDLE64 directive,
TITAN_HANDLE64 component_handle, TITAN_HANDLE64
reference = 0, TITAN_HANDLE64 reference_handshake = 0)
```

Method **X4** is used to rephrase a directive component needs. Directive components can be rephrased when the original directive references are swapped with rephrased references.

Where:

- **directive**: Handle of the directive whose component is being retrieved.
- **component_handle**: Handle of the component whose reference will be updated.
- **reference**: Public handle to the reference that will be rephrasing the component. 0 is also a valid handle.
- ***directive_handshake***: Handshake to be processed by the manager hosting the directive.

RETURN:

- **TITAN_RESULT_OK:** The directive component has been successfully rephrased.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET THE NUMBER OF COMPONENTS IN A DIRECTIVE (X5):

```
TITAN_ULONG GetNumComponents(TITAN_HANDLE64 directive,
directive_handshake = 0)
```

Directives have a varying number of components that depends on the statement of the directive. Method **X5** is used to retrieve the total count of components in supplied directive.

Where:

- **directive**: Public handle of the directive queried.
- **directive_handshake**: Handshake to be processed by the manager hosting the directive.

RETURN:

- Number of components in the directive upon successful operation.
- 0 if the directive has no components or upon failure.

GET DIRECTIVE ACTION (X6):

```
titan_result_t GetAction(TITAN_HANDLE64 directive,
titan_action_info_t &action_info, TITAN_HANDLE64
directive_handshake = 0)
```

Method **X6** is used to retrieve action information of the directive.

Where:

- **directive**: Public handle of the directive queried.
- **action_info**: A user-supplied structure where directive action information will be stored, where:
  - **name**: Text string where the name of the directive action will be copied. Set to NULL if no name is requested.
  - **label_handle**: Directive action label handle.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **amount**: A titan_amount_t structure describing the amount of the action such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:
      - **handle_value**: Amount data is a 64-bit handle.
      - **ulong_value**: Amount data is a long unsigned 64-bit integer.
      - **long_value**: Amount data is a long signed 64-bit integer.
      - **double_value**: Amount data is a double-precision floating point.
      - **float_value**: Amount data is a single-precision floating point.
      - **string_value**: Amount data is a null-terminate character string.
      - **raw_value**: Amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Amount data is an unsigned 32-bit integer.
      - **int_value**: Amount data is a signed 32-bit integer.
      - **ushort_value**: Amount data is a short unsigned 16-bit integer.
      - **short_value**: Amount data is a short signed 16-bit integer.
- **query_flags**: Retrieved data depends on this bitmap, where active bits are interpreted as:
  - **TITAN_GET_NONE**: No query, but action name can still be retrieved.
  - **TITAN_GET_LABEL**: Retrieve directive action name label handle.
  - **TITAN_GET_AMOUNT**: Retrieve directive action amount.
- **directive_handshake**: Handshake to be processed by the manager hosting the directive.

RETURN:

- **TITAN_RESULT_OK:** Directive action information has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET DIRECTIVE STATE (X7):

```
TITAN_HANDLE64 GetState(TITAN_HANDLE64 directive,
TITAN_HANDLE64 previous_state, titan_value_t
&state_info, TITAN_HANDLE64 directive_handshake = 0)
```

Method **X7** is called to retrieve the state of a directive or the next state of a directive (if a previous state is specified).

Where:

- **directive**: Public handle of the directive queried.
- **previous_state**: Handle of the previous directive state. Set to 0 to receive the first state.
- **state_info**: A user-supplied structure where directive state information will be stored, where:
  - **name_handle**: Directive state label handle.
  - **amount.format_handle**: Handle of the amount format (type or units).
  - **amount.*_value**: Union of the amount where:
    - **handle_value**: Amount data is a 64-bit handle.
    - **ulong_value**: Amount data is a long unsigned 64-bit integer.
    - **long_value**: Amount data is a long signed 64-bit integer.
    - **double_value**: Amount data is a double-precision floating point.
    - **float_value**: Amount data is a single-precision floating point.
    - **string_value**: Amount data is a null-terminate character string.
    - **raw_value**: Amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Amount data is an unsigned 32-bit integer.
    - **int_value**: Amount data is a signed 32-bit integer.
    - **ushort_value**: Amount data is a short unsigned 16-bit integer.
    - **short_value**: Amount data is a short signed 16-bit integer.
- **directive_handshake**: Handshake to be processed by the manager hosting the directive.

RETURN:

- Handle of the first directive state upon successful operation.
- 0 upon failure to retrieve the first directive state.

SET A DIRECTIVE COMPONENT HANDSHAKE(X8):

```
titan_result_t SetHandshake(TITAN_HANDLE64 directive,
TITAN_HANDLE64 component_handle, TITAN_HANDLE64
component_handshake, TITAN_HANDLE64 directive_handshake
= 0)
```

Method **X8** is called to set the handshake of a specific component.

Where:

- **directive**: Public handle of the directive queried.
- **component_handle**: Handle of the component whose handshake will be set.
- ***directive_handshake***: Handshake to be processed by the manager hosting the directive.

RETURN:

- **TITAN_RESULT_OK:** Component handshake has been set successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

CLEAR DIRECTIVE HANDSHAKES (X9):

```
titan_result_t ClearHandshakes(TITAN_HANDLE64
directive, TITAN_HANDLE64 directive_handshake = 0)
```

Method **X9** is called to clear all the handshakes of a directive.

Where:

- **directive**: Public handle of the directive queried.
- **directive_handshake**: Handshake to be processed by the manager hosting the directive.

RETURN:

- **TITAN_RESULT_OK:** All directive handshakes have been cleared.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REMEMBER A DIRECTIVE (XA):

```
TITAN_HANDLE64 Remember(TITAN_HANDLE64 directive,
titan_manager_t *directive_manager, TITAN_HANDLE64
directive_handshake = 0)
```

Method **XA** is used to temporarily remember a directive.

Where:

II. **directive**: Public handle of the directive queried.
III. **directive_manager**: Optional data structure of the registering manager, where:
- **DirectiveMethod**: Pointer to the directive manager method (See *XX: Directive Manager*).
- **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.

IV. **directive_handshake**: Handshake to be processed by the manager hosting the directive.
V. **RETURN**:
- Handle of the directive remembered upon successful operation.
- 0 upon failure to remember the directive.

FORGET A DIRECTIVE DATA (XB):

```
titan_result_t Forget(TITAN_HANDLE64 directive,
TITAN_HANDLE64 directive_handshake = 0)
```

Method **XB** is called to instruct Atlas to forget a previously remembered directive specified by handle. This handle should have been created using *XA:Remember* earlier in the session.

Where:

- **directive**: Public handle of the directive queried.
- ***directive_handshake***: Handshake to be processed by the manager hosting the directive.

RETURN:

- **TITAN_RESULT_OK:** The directive has been forgotten successfully. No future calls should reference the directive.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

DIRECTIVE MANAGER (XX: ATLAS DIRECTIVE MANAGER METHOD):

```
TITAN_METHOD_RESULT MyDirectiveManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake)
```

Method **XX** is called when a directive requests that require handshake need to be validated prior to processing.

Where:

- **caller_reference**: Private handle of the reference making the request.
- **request_type**: Type of request, where:
  - **TITAN_DIRECTIVE_GET_COMPONENT**: Request component or state information from a directive.
  - **TITAN_DIRECTIVE_REPHRASE**: Request a directive component to be rephrased.
  - **TITAN_DIRECTIVE_GET_ACTION**: Request action of the directive.
  - **TITAN_DIRECTIVE_REMEMBER**: Request for directive to be remembered.
  - **TITAN_DIRECTIVE_FORGET**: Request for directive to be forgotten.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *XA:Remember*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the directive to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the directive not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information

IV. Label Interface:

Information:

A label is an arbitrary text string associated with a 64-bit value.

Labels are domain dependent.

This rule applies to all label creation, retrieval, and destruction methods.

Annotation:

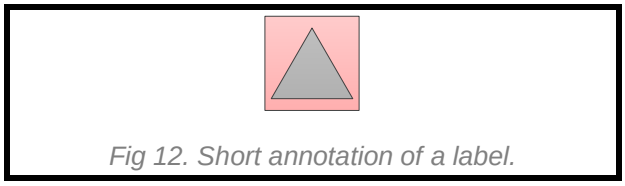


Fig 12. Short annotation of a label.

Label Interface:

Labels can be created and accessed through the interface below:

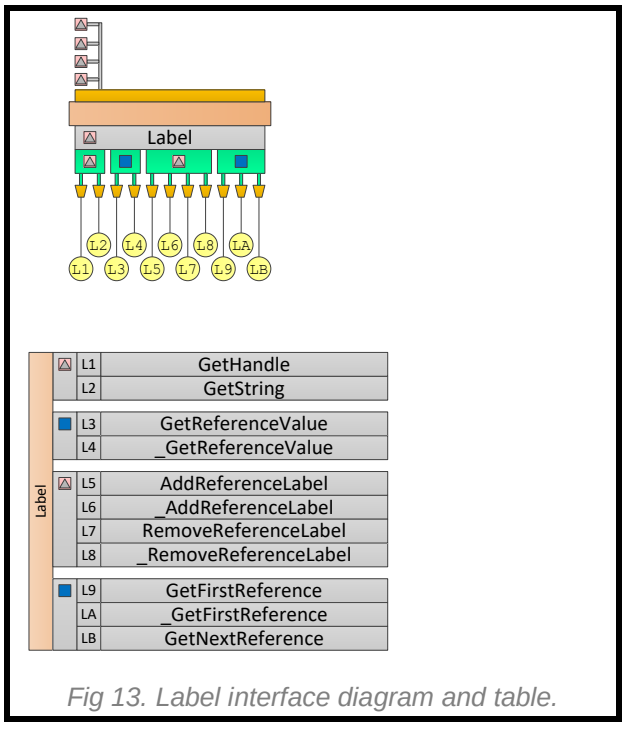


Fig 13. Label interface diagram and table.

GET A LABEL HANDLE (L1):

```
TITAN_HANDLE64 GetHandle(const TITAN_STRING label_text,
TITAN_BOOL create = true)
```

Method **L1** is called to get or create a text string label

Where:

- **label_text**: Label text string to be cataloged or retrieved.
- ***create***: Catalog/create label if *create* is true; otherwise, attempt to retrieve label handle based on the cataloged label text string.

- **RETURN**: The expected return values of the manager are:
  - Handle of the retrieved label upon successful operation.
  - 0 upon failure to create or retrieve a cataloged label.

GET A LABEL STRING (L2):

```
TITAN_UINT GetString(TITAN_HANDLE64 label_handle,
TITAN_STRING output_text, TITAN_UINT
output_text_max_size = 0)
```

Method **L2** is used to get the text string of a cataloged label.

Where:

- **label_handle**: Handle of a cataloged label.
- **output_text:** User-supplied string where the label text string will be copied.
- ***output_text_max_size*:** Maximum number of bytes to be copied in the user-supplied string.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK**: The label text string was successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET A LABEL VALUE FROM A REFERENCE (L3):

```
titan_result_t GetReferenceValue(TITAN_HANDLE64
calling_reference, TITAN_HANDLE64 public_reference,
const TITAN_STRING label_text_string, titan_value_t
&value, TITAN_HANDLE64 instance_handshake = 0)
```

Method **L3** is used to retrieve the label value of a reference through the instantiator method *I9:AtlasSim_GetValue* which requests the reference's instance to return values registered under labeled formats.

Where:

- **calling_reference**: Public handle of the reference making the request.
- **public_reference**: Public handle of the reference whose instance value will be retrieved.
- **label_handle**: Handle of the format (or name) of the value retrieved.
- **value**: The output configuration value structure, where:
  - **format_handle**: *ignored*.
  - ***_value**: The output atomic value retrieved from the instance, such that:
    - **handle_value**: The value is a 64-bit handle.
    - **ulong_value**: The value is a long unsigned 64-bit integer.
    - **long_value**: The value is a long signed 64-bit integer.
    - **double_value**: The value is a double-precision floating point.
    - **float_value**: The value is a single-precision floating point.
    - **string_value**: The value is a null-terminate character string.
    - **raw_value**: The value is an opaque 64-bit memory pointer.
    - **buffer_value**: The value is a 64-bit unsigned byte pointer.
    - **uint_value**: The value is an unsigned 32-bit integer.
    - **int_value**: The value is a signed 32-bit integer.
    - **ushort_value**: The value is a short unsigned 16-bit integer.
    - **short_value**: The value is a short signed 16-bit integer.
- **instance_handshake**: Handshake to be processed by the instantiator hosting the referenced instance.

RETURN:

- **TITAN_RESULT_OK**: The value was successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET A LABEL VALUE FROM A REFERENCE (L4):

```
titan_result_t _GetReferenceValue(TITAN_HANDLE64
calling_reference, TITAN_HANDLE64 public_reference,
titan_value_t &value, TITAN_HANDLE64 handshake = 0)
```

Method **L4** is used to retrieve the label value of a reference through the instantiator method *I9:AtlasSim_GetValue* which requests the reference's instance to return values registered under labeled formats.

Where:

- **calling_reference**: Public handle of the reference making the request.
- **public_reference**: Public handle of the reference whose instance value will be retrieved.
- **value**: A titan_value_t structure, where:
  - **name_handle**: Handle of the name of the value.
  - **amount.format_handle**: Handle of the amount format (type or units).
  - **amount.*_value**: Union of the amount where:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the referenced instance.

RETURN:

- **TITAN_RESULT_OK**: The value was successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

ADD A LABEL TO AN EXISTING REFERENCE (L5):

```
titan_result_t AddReferenceLabel(TITAN_HANDLE64
calling_reference, const TITAN_STRING label,
TITAN_HANDLE64 public_reference, TITAN_HANDLE64
agent_handshake = 0)
```

Method **L5** is called to add a label to an existing reference. This label will be another name by which the reference is invoked.

Where:

- **calling_reference**: Private handle of the reference requesting a label to be added.
- **label**: Name of the reference.
- **public_reference:** Public handle of the reference being labeled.
- **agent_handshake**: Handshake to be processed by the context hosting the calling reference.

RETURN:

- **TITAN_RESULT_OK:** The reference has been successfully labeled.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

ADD A LABEL TO AN EXISTING REFERENCE (L6):

```
titan_result_t _AddReferenceLabel(TITAN_HANDLE64
public_context, TITAN_HANDLE64 label_handle,
TITAN_HANDLE64 public_reference, TITAN_HANDLE64
agent_handshake = 0)
```

Method **L6** is called to add a label to an existing reference. This label will be another name by which the reference is invoked.

Where:

- **calling_reference**: Private handle of the reference requesting a label to be added.
- **label_handle**: Handle of the name of the reference.
- **public_reference:** Public handle of the reference being labeled.
- ***agent_handshake***: Handshake to be processed by the context hosting the calling reference.

RETURN:

- **TITAN_RESULT_OK:** The reference has been successfully labeled.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REMOVE A LABEL FROM AN EXISTING REFERENCE (L7):

```
titan_result_t RemoveReferenceLabel(TITAN_HANDLE64
calling_reference, const TITAN_STRING label,
TITAN_HANDLE64 public_reference, TITAN_HANDLE64
agent_handshake = 0)
```

Method **L7** is called to remove a label from an existing reference.

Where:

- **calling_reference**: Private handle of the reference requesting a label to be removed.
- **label**: Name of the reference.
- **public_reference**: Reference whose label will be removed.
- ***agent_handshake***: Handshake to be processed by the context hosting the calling reference.

RETURN:

- **TITAN_RESULT_OK:** The reference label has been successfully removed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REMOVE A LABEL FROM AN EXISTING REFERENCE (L8):

```
titan_result_t _RemoveLabel(TITAN_HANDLE64
label_handle, TITAN_HANDLE64 public_reference,
TITAN_HANDLE64 agent_handshake = 0)
```

Method **L8** is called to remove a label from an existing reference.

Where:

- **calling_reference**: Private handle of the reference requesting a label to be removed.
- **label_handle**: Handle of the name of the reference.
- **public_reference**: Reference whose label will be removed.
- ***agent_handshake***: Handshake to be processed by the context hosting the calling reference.

RETURN:

- **TITAN_RESULT_OK:** The reference label has been successfully removed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE THE FIRST LABELED REFERENCE (L9):

```
TITAN_HANDLE64 GetFirstReference(TITAN_HANDLE64
calling_reference, const TITAN_STRING label,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **L9** is called to register a reference as an expert.

Where:

- **calling_reference**: Private handle of the reference making the request.
- **label**: Name of the references queried.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **relation**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the relation or preposition.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:

        - **handle_value**: Amount data is a 64-bit handle.
            - **ulong_value**: Amount data is a long unsigned 64-bit integer.
            - **long_value**: Amount data is a long signed 64-bit integer.
            - **double_value**: Amount data is a double-precision floating point.
            - **float_value**: Amount data is a single-precision floating point.
            - **string_value**: Amount data is a null-terminate character string.
            - **raw_value**: Amount data is an opaque 64-bit memory pointer.
            - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
            - **uint_value**: Amount data is an unsigned 32-bit integer.
            - **int_value**: Amount data is a signed 32-bit integer.
            - **ushort_value**: Amount data is a short unsigned 16-bit integer.
            - **short_value**: Amount data is a short signed 16-bit integer.
        - **instance**: Returned instance of the reference.
        - **element**: Returned element of the reference.
- **query_flags**: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.
- **TITAN_GET_RELATION**: Retrieve reference relation or preposition.

RETURN:

- Handle of the first reference occurrence upon successful operation.
- 0 upon failure to retrieve the first occurrence.

RETRIEVE THE FIRST LABELED REFERENCE (LA):

```
TITAN_HANDLE64 _GetFirst(TITAN_HANDLE64
private_reference, TITAN_HANDLE64 label_handle,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **LA** is called to retrieve the first reference matching supplied name in the zones entered by the context agent.

Where:

- **calling_reference**: Private handle of the reference making the request.
- **label_handle**: Handle of the name of the references queried.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **relation**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the relation or preposition.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).

- ***_value**: Union of the amount where:
  - **handle_value**: Amount data is a 64-bit handle.
  - **ulong_value**: Amount data is a long unsigned 64-bit integer.
  - **long_value**: Amount data is a long signed 64-bit integer.
  - **double_value**: Amount data is a double-precision floating point.
  - **float_value**: Amount data is a single-precision floating point.
  - **string_value**: Amount data is a null-terminate character string.
  - **raw_value**: Amount data is an opaque 64-bit memory pointer.
  - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
  - **uint_value**: Amount data is an unsigned 32-bit integer.
  - **int_value**: Amount data is a signed 32-bit integer.
  - **ushort_value**: Amount data is a short unsigned 16-bit integer.
  - **short_value**: Amount data is a short signed 16-bit integer.
- **instance**: Returned instance of the reference.
- **element**: Returned element of the reference.
- **query_flags**: Retrieved data depends on this bitmap, where active bits are interpreted as:
  - **TITAN_GET_NONE**: No query, but names can still be retrieved.
  - **TITAN_GET_REFERENCE**: Retrieve reference ID.
  - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
  - **TITAN_GET_AMOUNT**: Retrieve reference amount.
  - **TITAN_GET_RELATION**: Retrieve reference relation or preposition.

RETURN:

- Handle of the first reference occurrence upon successful operation.
- 0 upon failure to retrieve the first occurrence.

RETRIEVE THE NEXT LABEL REFERENCE (LB):

```
TITAN_HANDLE64 GetNextReference(TITAN_HANDLE64
calling_reference, TITAN_HANDLE64 label_element,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **LB** is called to retrieve the next reference matching the supplied name in the zones entered by the context agent.

Where:

- **calling_reference**: Private handle of the reference making the request.
- **label_element**: Handle to the label reference whose successor will be retrieved.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
- **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
- **name_max_size**: Maximum number of characters to copy to the *name* string.
- **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
- **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
- **relation**: A titan_value_t structure, where:
  - **name_handle**: Handle of the name of the relation or preposition.
  - **amount.format_handle**: Handle of the amount format (type or units).
  - **amount.*_value**: Union of the amount where:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- **amount**: A titan_amount_t structure describing the amount of the reference such that:
  - **format_handle**: Handle of the amount format (type or units).
  - ***_value**: Union of the amount where:

    - **handle_value**: Amount data is a 64-bit handle.
    - **ulong_value**: Amount data is a long unsigned 64-bit integer.
    - **long_value**: Amount data is a long signed 64-bit integer.
    - **double_value**: Amount data is a double-precision floating point.
    - **float_value**: Amount data is a single-precision floating point.
    - **string_value**: Amount data is a null-terminate character string.
    - **raw_value**: Amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Amount data is an unsigned 32-bit integer.
    - **int_value**: Amount data is a signed 32-bit integer.
    - **ushort_value**: Amount data is a short unsigned 16-bit integer.
    - **short_value**: Amount data is a short signed 16-bit integer.
- **instance**: Returned instance of the reference.
- **element**: Returned element of the reference.
- **query_flags**: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.
    - **TITAN_GET_RELATION**: Retrieve reference relation or preposition.

RETURN:

- Handle of the next reference occurrence upon successful operation.
- 0 upon failure to retrieve the next occurrence.

V. Ledger Interface:

Information:

The ledger is defined as a circular buffer where the oldest entries are overwritten by the new. Multiple ledgers can exist in the same domain.

Each entry in a ledger is comprised of an expression (text string), the author of that expression, and an attachment of an arbitrary size that relates to the expression.

The arbitrary data, for example, could be a list of corresponding interpreted directives, a system state, or anything the user seems valuable to be stored with the expression.

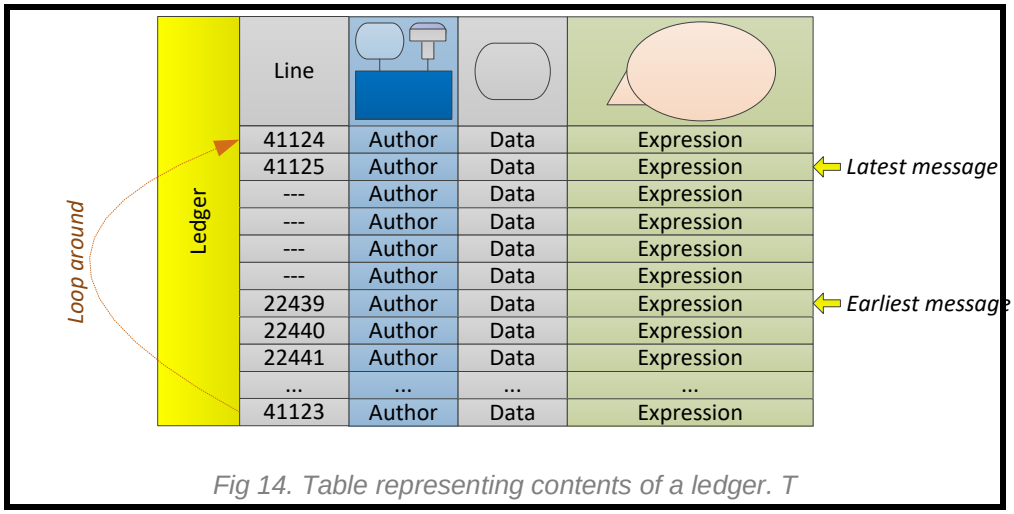


Fig 14. Table representing contents of a ledger. T

The ledger API allows references to remember and recall expressions they have performed over time.

Annotation:

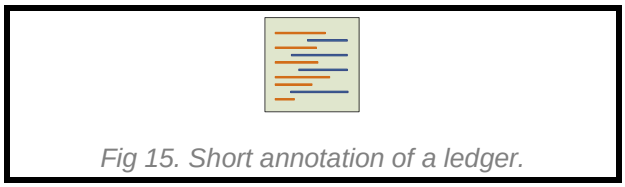
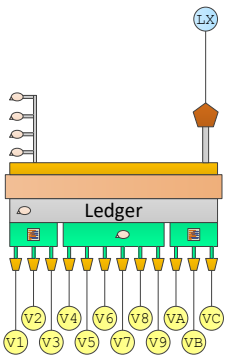


Fig 15. Short annotation of a ledger.

Ledger Interface:



Ledger	V1	Create
	V2	Destroy
	V3	Get
	V4	ClearMessages
	V5	InsertMessage
	V6	GetMessageData
	V7	GetEarliestMessage
	V8	GetLatestMessage
	V9	GetPreviousMessage
	VA	GetNextMessage
	VB	RegisterKeyword
	VC	UnregisterKeyword
	VD	ProcessMessageKeywords

Fig 16. Ledger interface diagram and table.

CREATE A LEDGER (V1):

```
TITAN_HANDLE64 Create(TITAN_HANDLE64 caller_reference,
const TITAN_STRING ledger_name, TITAN_SIZE64
ledger_buffer_size, titan_manager_t *ledger_manager =
NULL, TITAN_HANDLE64 domain_handshake = 0)
```

Method **V1** is called when a ledger needs to be created.

Where:

- **caller_reference**: Private handle of the reference creating the ledger.
- **ledger_name**: Name of the ledger to be created.
- **ledger_buffer_size:** Size of the ledger in bytes.
- ***ledger_manager***: Optional data structure of the registering manager, where:
  - **Ledger**: Pointer to the ledger manager method (See *VX: Ledger Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Handshake to be processed by the domain hosting the reference.

RETURN:

- Handle to the created or retrieved by the ledger upon successful operation.
- 0 upon failing to create the ledger.

DESTROY A LEDGER (V2):

```
void Destroy(TITAN_HANDLE64 ledger_handle,
TITAN_HANDLE64 ledger_handshake = 0)
```

Method **V2** is called when an existing ledger needs to be destroyed. Destroying a ledger will remove all entries and data associated with the entries of the ledger.

Where:

- **ledger_handle**: Handle of an existing ledger to be destroyed.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- **TITAN_RESULT_OK:** The ledger has been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE A LEDGER (V3):

```
TITAN_HANDLE64 Get(TITAN_HANDLE64 private_reference,
const TITAN_STRING ledger_name, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **V3** is called when a ledger needs to be retrieved by name. An existing ledger can be retrieved if the calling reference is hosted in the same domain as the ledger.

Where:

- **private_reference**: Private handle of the reference retrieving the ledger.
- **ledger_name**: Name of the ledger to be retrieved.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the created or retrieved ledger upon successful operation.
- 0 upon failing to create the ledger.

CLEAR THE LEDGER (V4):

```
TITAN_HANDLE64 ClearMessages(TITAN_HANDLE64
ledger_handle, TITAN_HANDLE64 ledger_handshake = 0)
```

Method **V**4 is called to clear existing ledger entries. When a ledger is cleared, all entries and data associated with the entries are cleared.

Where:

- **ledger_handle**: Handle of an existing ledger to be cleared.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- **TITAN_RESULT_OK:** All ledger entries have been cleared.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET LEDGER ENTRY DATA (V5):

```
TITAN_POINTER GetMessageData(TITAN_HANDLE64
message_handle, TITAN_USHORT &data_size, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **V5** is called to retrieve data associated with an existing ledger entry.

Where:

- **message_handle**: Handle of the message entry whose data will be retrieved.
- **data_size:** Returned size of the message data.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Pointer to message data of size *data_size* upon successful operation.
- NULL upon failure to retrieve message data.

ADD A LEDGER ENTRY (V6):

```
TITAN_HANDLE64 InsertMessage(TITAN_HANDLE64
ledger_handle, TITAN_USHORT requested_size,
TITAN_HANDLE64 ledger_handshake, const TITAN_STRING
format, ...)
```

Method **V6** is called to add a new entry into the ledger.

Where:

- **ledger_handle**: Handle of an existing ledger to be cleared.
- **requested_size:** Number of bytes to be allocated alongside the message.
- **ledger_handshake**: Handshake to be processed by the registered ledger manager.
- **format**: Text string that is formatted using the C *printf* specification.
- **…:** Additional arguments to be replaced by the specifiers in *format***.**

RETURN:

- Handle to the inserted message upon successful operation.
- 0 upon failure to insert the message.

RETRIEVE EARLIEST LEDGER ENTRY (V7):

```
TITAN_HANDLE64 GetEarliestMessage(TITAN_HANDLE64
ledger_handle, TITAN_UINT &message_index, TITAN_UINT
&message_size, TITAN_STRING output_text = NULL,
TITAN_SIZE64 output_text_size = 0, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **V7** is called to get the earliest message entry in the ledger.

Where:

- **ledger_handle**: Handle of the ledger whose earliest message will be retrieved.
- **message_index**: Returned absolute index of the first message entry upon success.
- **message_size**: Returned number of bytes in the message retrieved upon success.
- ***output_text*:** User-supplied string where the message string will be copied.
- ***output_text_size*:** Maximum number of bytes to be copied in the user-supplied string.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the retrieved message upon successful operation.
- 0 upon failure to retrieve the message.

RETRIEVE LATEST LEDGER ENTRY (V8):

```
TITAN_HANDLE64 GetLatestMessage(TITAN_HANDLE64
ledger_handle, TITAN_UINT &line_index, TITAN_UINT
&line_size, TITAN_STRING output_text = NULL,
TITAN_SIZE64 output_text_size = 0, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **V8** is called to get the latest message entry in the ledger.

Where:

- **ledger_handle**: Handle of the ledger whose latest message will be retrieved.
- **message_index**: Returned absolute index of the last message entry upon success.
- **message_size**: Returned number of bytes in the message retrieved upon success.
- ***output_text*:** User-supplied string where the message string will be copied.
- ***output_text_size*:** Maximum number of bytes to be copied in the user-supplied string.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the retrieved message upon successful operation.
- 0 upon failure to retrieve the message.

RETRIEVE PREVIOUS LEDGER ENTRY (V9):

```
TITAN_HANDLE64 PreviousMessage(TITAN_HANDLE64
message_handle, TITAN_UINT &line_index, TITAN_UINT
&line_size, TITAN_STRING output_text = NULL,
TITAN_SIZE64 output_text_size = 0, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **V9** is called when a previous message is requested from an existing ledger.

Where:

- **message_handle**: Handle of the message entry whose predecessor will be retrieved.
- **message_index**: Returned absolute index of the previous message entry upon success.
- **message_size**: Returned number of bytes in the message retrieved upon success.
- ***output_text*:** User-supplied string where the message string will be copied.
- ***output_text_size*:** Maximum number of bytes to be copied in the user-supplied string.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the retrieved message upon successful operation.
- 0 upon failure to retrieve the message.

RETRIEVE NEXT LEDGER ENTRY (VA):

```
TITAN_HANDLE64 NextMessage(TITAN_HANDLE64
message_handle, TITAN_UINT &line_index, TITAN_UINT
&line_size, TITAN_STRING output_text = NULL,
TITAN_SIZE64 output_text_size = 0, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **VA** is called when a next message is requested from an existing ledger.

Where:

- **message_handle**: Handle of the message entry whose successor will be retrieved.
- **message_index**: Returned absolute index of the next message entry upon success.
- **message_size**: Returned number of bytes in the message retrieved upon success.
- ***output_text*:** User-supplied string where the message string will be copied.
- ***output_text_size*:** Maximum number of bytes to be copied in the user-supplied string.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the retrieved message upon successful operation.
- 0 upon failure to retrieve the message.

REGISTER A LEDGER KEYWORD (VB):

```
TITAN_HANDLE64 RegisterKeyword(TITAN_HANDLE64
ledger_handle, const TITAN_STRING keyword,
TITAN_HANDLE64 axiom_data, AtlasKeywordMethod
KeywordMethod, TITAN_HANDLE64 ledger_handshake = 0)
```

Method **VB** is called when a text string is registering as a ledger managed message keyword. All future encounters of the keyword in the ledger will invoke the manager registered in this method call.

Where:

- **ledger_handle**: Handle of the ledger hosting the new keyword.
- **keyword**: Keyword string to be registered.
- **axiom_data**: User-supplied handle that will be forwarded to *KeywordMethod* with every call.
- **KeywordMethod:** Pointer to the keyword-intercept method (See *KX: Keyword Method*).
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- Handle to the registered keyword upon successful operation.
- 0 upon failure to register the keyword with the specified ledger.

UNREGISTER A LEDGER KEYWORD(VC):

```
titan_result_t UnregisterKeyword(TITAN_HANDLE64
ledger_handle, TITAN_HANDLE64 keyword_handle,
TITAN_HANDLE64 ledger_handshake = 0)
```

Method **VC** is used to unregister an existing keyword in a specific ledger.

Where:

- **ledger_handle**: Handle of the ledger hosting supplied keyword.
- **keyword_handle**: Handle to the registered keyword in the specified ledger.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- **TITAN_RESULT_OK**: The specified keyword was successfully unregistered from the ledger.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

PROCESS A LEDGER KEYWORD (VD):

```
titan_result_t ProcessMessageKeywords(TITAN_HANDLE64
caller_reference, TITAN_HANDLE64 ledger_handle, const
TITAN_STRING input_text, TITAN_HANDLE64
ledger_handshake = 0)
```

Method **VD** is called to process keywords in a supplied message text.

Where:

- **caller_reference**: Private handle to the reference requesting the message to be processed.
- **ledger_handle**: Handle of the ledger hosting the keywords that will be processing the input message string.
- **input_text**: Text string of the message to be processed.
- ***ledger_handshake***: Handshake to be processed by the registered ledger manager.

RETURN:

- **TITAN_RESULT_OK**: The message was successfully processed.
- **TITAN_RESULT_***: See Atlas Errors for more information.

LEDGER MANAGER (VX:ATLAS LEDGER MANAGER METHOD):

```
TITAN_METHOD_RESULT MyLedgerManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

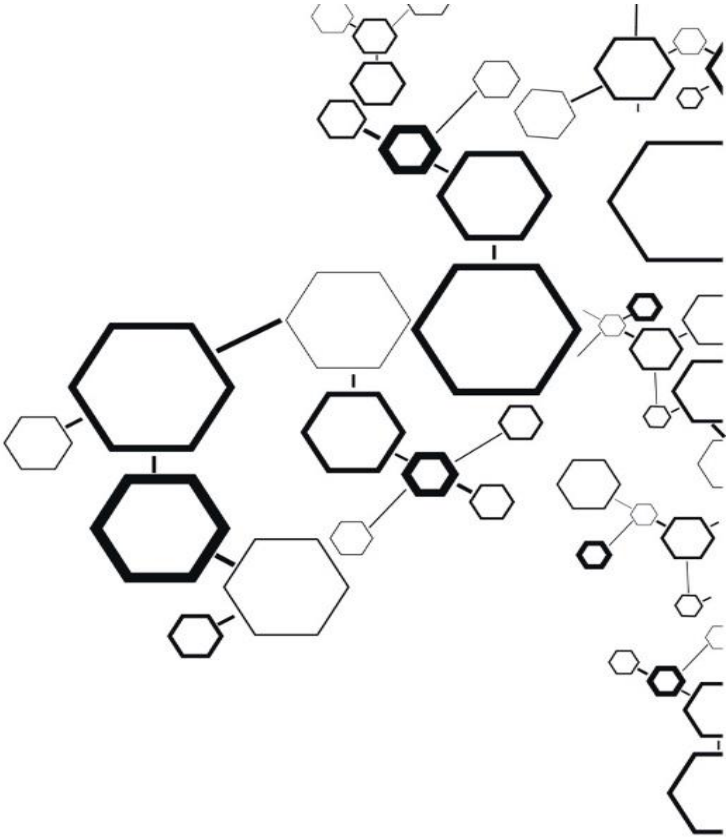
Method **VX** is called to evaluate ledger requests prior to processing.
The ledger manager is the axiom-supplied method responsible for validating ledger requests. This method is registered during ledger creation (*L1:Create*). All ledger requests that require a handshake will be evaluated by this manager method prior to processing the requests.

Where:

- **caller_reference**: Private handle of the reference making the ledger request.
- **request_type**: Type of request, where:
  - **TITAN_LEDGER_DESTROY**: Request ledger destruction.
  - **TITAN_LEDGER_GET_MESSAGE**: Request to retrieve a ledger message.
  - **TITAN_LEDGER_SET_MESSAGE**: Request to set a ledger message.
  - **TITAN_LEDGER_SET_KEYWORD**: Request to register a keyword.
  - **TITAN_LEDGER_PROCESS_KEYWORD**: Request to process a keyword.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See V*1:Create*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the ledger to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the ledger not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.



Atlas

Simulation API Bundle

Simulation API Bundle:

Introduction:

The Simulation API bundles handle functionality exposed to concepts by the cognitive environment.

The cognitive environment can host one or multiple interfaces advertised by Atlas or third-party software.

Interfaces are universal and transcend zones within a domain.

Simulation APIs are grouped into the following:

- **Simulation Interface (Sx):** Manages clones and simulation objects.
- **Reference Interface (Rx)**: Manages simulation references.
- **Instance Interface (Ix)**: Manages concept instances.
- **Group Interface (Gx)**: Manages simulation groups.
- **Attributes Interfaces (F)**: Manage simulation object attributes.

All methods described within are called by the axiom and resolved by Atlas:

- Sx methods handle simulation functionality.
- Rx methods handle reference functionality.
- Mx methods handle instance functionality.
- Gx methods handle group functionality.
- F methods handle attribute functionality.

Annotation:

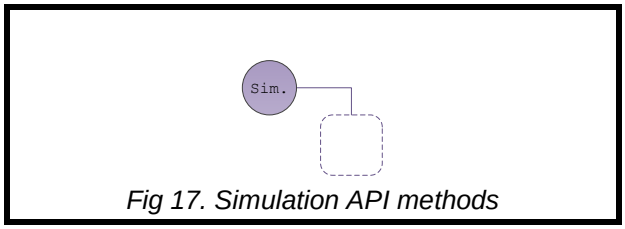


Fig 17. Simulation API methods

Simulation API Interfaces:

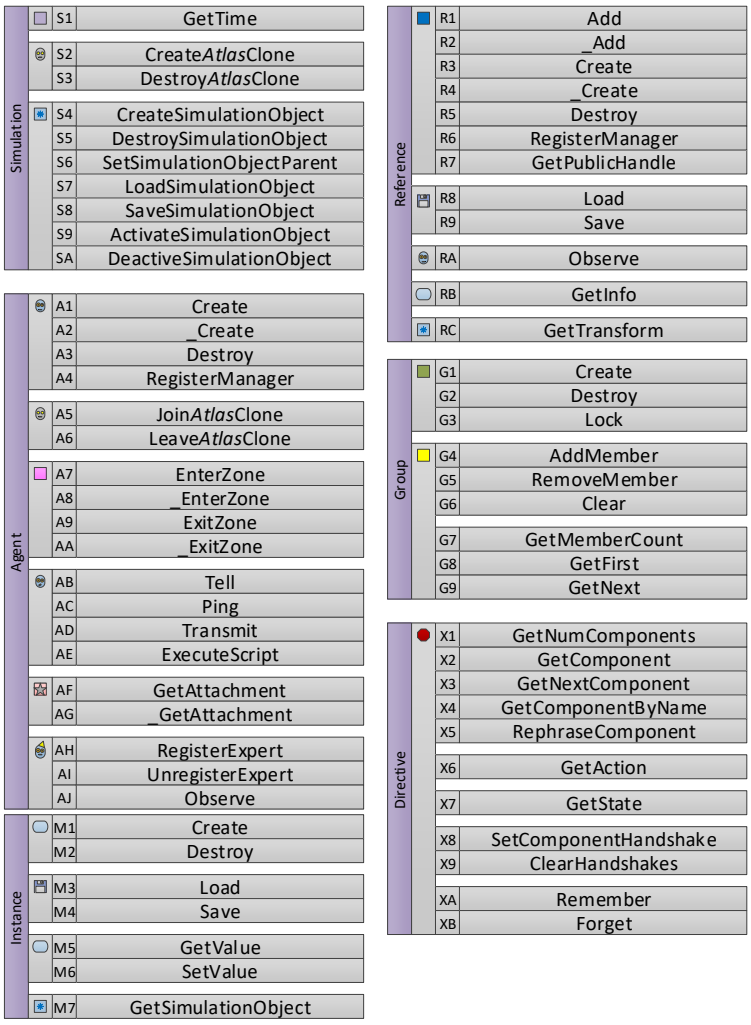


Fig 18. Simulation API interface tables.

I. Simulation Interface:

Annotation:

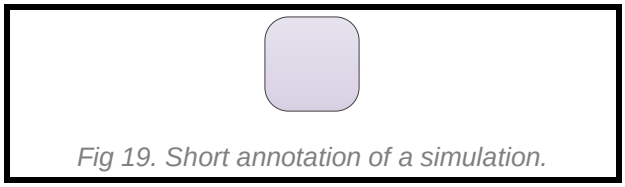


Fig 19. Short annotation of a simulation.

Simulation Interface:

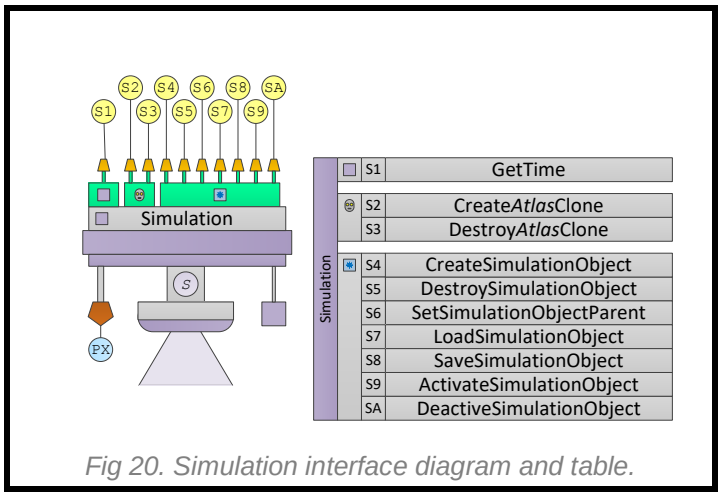


Fig 20. Simulation interface diagram and table.

GET SIMULATION TIME (S1):

```
TITAN_FLOAT GetTime(TITAN_HANDLE64 sim_handle)
```

A simulation has an internal timer that counts the number of seconds since that simulation has been created. The timer has millisecond accuracy. Method **S1** returns the internal timer value for a supplied simulation handle.

Where:

- **sim_level**: Simulation level reserved for future use. Values other than 0 are currently ignored.

RETURN:

- Time in seconds since simulation has started.

CREATE AN ATLAS CLONE (S2):

```
TITAN_HANDLE64 CreateAtlasClone(TITAN_HANDLE64
caller_reference, TITAN_HANDLE64 simulation_handle,
const TITAN_STRING clone_name, TITAN_FLOAT
frame_time_ms = 0.0f, titan_manager_t *clone_manager =
NULL, TITAN_HANDLE64 simulation_handshake = 0)
```

Method **S2** is called to create an Atlas clone. Atlas clones can have names and managers.

Where:

- **caller_reference**: Private handle of the reference creating the new simulation process. This reference must be an agent.
- **clone_name**: Name of the new process being created.
- ***frame_time_ms***: Process frame time in milliseconds *(Process frequency = 1 / frame_time_ms * 1000).*
- ***clone_manager***: Optional data structure of the registering manager, where:
  - **SimulationClone**: Pointer to the clone manager method (**See *PX: Clone Manager***).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the process.

RETURN:

- Handle of the new simulation process upon successful operation.
- 0 upon failure to create a new simulation frame.

DESTROY AN ATLAS CLONE (S3):

```
titan_result_t DestroyAtlasClone(TITAN_HANDLE64
clone_handle, TITAN_HANDLE64 process_handshake = 0)
```

Method **S3** is called to destroy an existing Atlas clone. Destroying a clone will detach all the agents that are joined to it.

Where:

- **clone_handle**: Private handle of an existing simulation clone.
- **clone _handshake**: Handshake to be processed by the process manager of the simulation clone being destroyed.

RETURN:

- **TITAN_RESULT_OK**: Process is successfully terminated and destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

CREATE A SIMULATION OBJECT (S4):

```
TITAN_HANDLE64 CreateSimulationObject(TITAN_HANDLE64
concept, TITAN_HANDLE64 simulation_handshake = 0)
```

Method **S4** is called to create a simulation object (attributes object) on behalf of a concept.

Where:

- **concept**: Handle of the concept whose defined simulation object will be instantiated.
- **simulation_handshake**: Handshake to be processed by the simulation hosting the simulation object.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** The instance referenced has been successfully registered as an expert.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

DESTROY A SIMULATION OBJECT (S5):

```
titan_result_t DestroySimulationObject(TITAN_HANDLE64
simulation_object, TITAN_HANDLE64 simulation_handshake
= 0)
```

Method **S5** is called to destroy a simulation object (attributes object). A destroyed simulation object will no longed be simulated. All discretionary simulation object dependencies should be removed by the developer:

Where:

- **simulation_object**: Handle to the simulation object instance that will be destroyed.
- **simulation_handshake**: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The simulation object instance has been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SET SIMULATION OBJECT PARENT (S6):

```
titan_result_t SetSimulationObjectParent(TITAN_HANDLE64
simulation_object, TITAN_HANDLE64 parent_object,
TITAN_HANDLE64 joint_handle = 0, TITAN_HANDLE64
simulation_handshake = 0)
```

Simulation objects can be hierarchically combined into a tree or a skeleton:

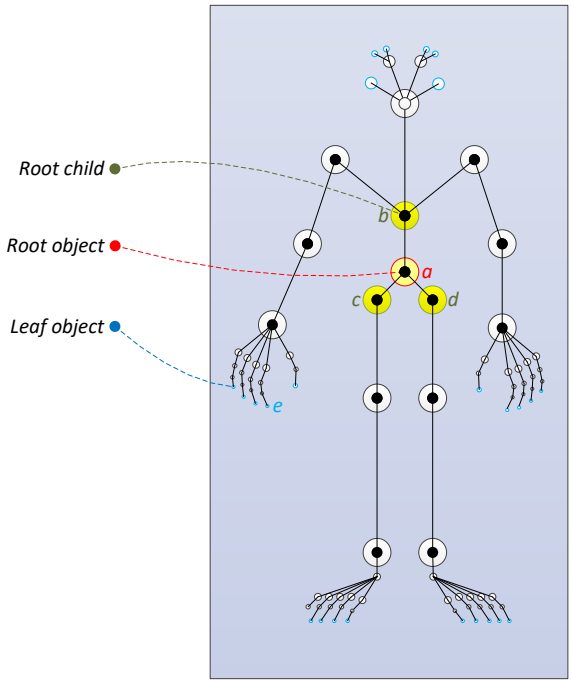


Fig 21. Simulation object tree of a humanoid: The root object, a, has three immediate children b, c, and d. Tree leaves, e, are marked in blue and have no children.

Once a child object is added to a parent object, all coordinates defined by the child object will be relative to the parent object, such that:

- Moving a parent object affects all of its children.
- Moving a child object does not affect the parent object

Method **S6** is used to link a child simulation object to its parent.

Where:

- **simulation_object**: Handle to the child simulation object being added to the tree.
- **parent_object**: Handle to the parent simulation object of supplied *simulation_object*.
- ***joint_handle***: This value is currently ignored. Reserved for future use.

- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The child simulation object was successfully added to the supplied parent.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

LOAD A SIMULATION OBJECT (S7):

```
titan_result_t LoadSimulationObject(titan_file_t *file,
TITAN_HANDLE64 simulation_object)
```

Saved simulation objects can be loaded from a supplied file using method **S7**.

Where:

- **file**: Atlas input data file pointer (See *Files*).
- **simulation_object**: Handle of the simulation object being loaded.

RETURN:

- **TITAN_RESULT_OK**: *Simulation object successfully saved to file*.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SAVE A SIMULATION OBJECT (S8):

```
titan_result_t SaveSimulationObject(titan_file_t *file,
TITAN_HANDLE64 simulation_object)
```

Method **S8** is called to save an existing simulation object to a supplied file.

Where:

- **file**: Atlas output data file pointer. (See *Files*).
- **simulation_object**: Handle of the simulation object being saved.

RETURN:

- **TITAN_RESULT_OK**: Simulation object saved to file successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

ACTIVATE A SIMULATION OBJECT (S9):

```
void ActivateSimulationObject(TITAN_HANDLE64
simulation_object)
```

Method **S9** is called to activate rendering of simulation object. Activation of a simulation object enables the temporal-spatial simulation/animation of the object in the simulation space.

Where:

- **simulation_object**: Activated simulation object in the temporal-spatial simulation.

RETURN:

- No value is returned.

DEACTIVATE A SIMULATION OBJECT (SA):

```
void DeactivateSimulationObject(TITAN_HANDLE64
simulation_object)
```

Method **SA** is called to deactivate rendering of a simulation object. Deactivating a simulation object disables the temporal-spatial simulation/animation of the object in the simulation space.

Where:

- **simulation_object**: Deactivated simulation object in the temporal-spatial simulation.

RETURN:

- No value is returned.

ATLAS CLONE MANAGER (PX: ATLAS CLONE MANAGER METHOD):

```
TITAN_METHOD_RESULT MyCloneManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

The *Atlas* clone manager is the axiom-supplied method responsible for validating *Atlas* clone requests. This method is registered during clone creation (*S2:Create*). All domain requests that require a handshake will be evaluated by this manager method prior to processing the requests.

Where:

- **caller_reference**: Private handle of the reference making the domain request (See *Reference*).
- **request_type**: Type of request, where:
  - **TITAN_CLONE_DESTROY**: Request *Atlas* clone destruction.
  - **TITAN_CLONE_JOIN**: Request for an agent to join the *Atlas* clone.
  - **TITAN_CLONE_LEAVE**: Request for an agent to leave the *Atlas* clone.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *S1:CreateAtlasClone*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the *Atlas* clone to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the *Atlas* clone not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

II. Agent Interface:

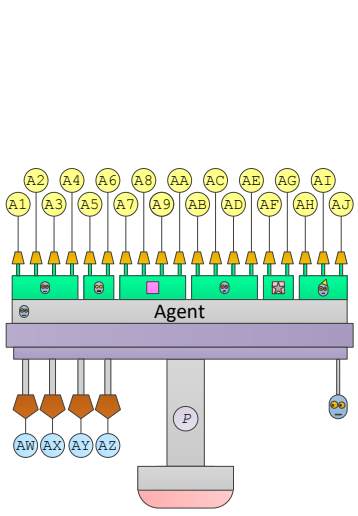
Information

An agent is an entity that interacts with the cognitive system and with other agents.

Annotation:

Fig 22. Short annotation of a context agent.

Agent Interface



Agent	A1	Create
	A2	_Create
	A3	Destroy
	A4	RegisterManager
	A5	Join*Atlas*Clone
	A6	Leave*Atlas*Clone
	A7	EnterZone
	A8	_EnterZone
	A9	ExitZone
	AA	_ExitZone
	AB	Tell
	AC	Ping
	AD	Transmit
	AE	ExecuteScript
	AF	GetAttachment
	AG	_GetAttachment
	AH	RegisterExpert
	AI	UnregisterExpert
	AJ	Observe

Fig 23. Agent interface diagram and table.

CREATE A CONTEXT AGENT (A1):

```
TITAN_HANDLE64 Create(const TITAN_STRING domain_name,
const TITAN_STRING zone_name, TITAN_ULONG memory_units,
TITAN_USHORT simulation_level = 0, TITAN_HANDLE64
domain_handshake = 0, TITAN_HANDLE64 zone_handshake =
0)
```

Context agents can be created inside of domains to represent goal-oriented entities. Method **A1** is called to create a context agent within a domain.

Where:

- **domain_name**: Name of the domain that will be hosting the context.
- **zone_name**: Name of the zone that will be hosting the context.
- **memory_units**: Amount of memory units dedicated to the context.
- ***simulation_level*:** Reserved for future use. This value is currently ignored.
- ***domain_handshake***: Handshake to be processed by the domain hosting the new context.
- ***zone_handshake***: Handshake to be processed by the zone hosting the new context.

RETURN:

- Handle to the context upon successful creation.
- 0 upon failure to create the context.

CREATE A CONTEXT AGENT (A2):

```
TITAN_HANDLE64 _Create(TITAN_HANDLE64
domain_name_handle, TITAN_HANDLE64 zone_name_handle,
TITAN_ULONG memory, TITAN_USHORT simulation_level = 0,
TITAN_HANDLE64 domain_handshake = 0, TITAN_HANDLE64
zone_handshake = 0)
```

Context agents can be created inside of domains to represent goal-oriented entities. Method **A2** is called to create a context agent within a domain.

Where:

- **domain_name_handle**: Handle of the name of the domain that will be hosting the context.
- **zone_name_handle**: Handle of the name of the zone that will be hosting the context.
- **memory_units**: Amount of memory units dedicated to the context.
- ***simulation_level*:** Reserved for future use. This value is currently ignored.
- ***domain_handshake***: Handshake to be processed by the domain hosting the new context.
- ***zone_handshake***: Handshake to be processed by the zone hosting the new context.

RETURN:

- Handle to the context upon successful creation.
- 0 upon failure to create the context.

DESTROY A CONTEXT AGENT (A3):

```
titan_result_t Destroy(TITAN_HANDLE64 private_context,
const TITAN_HANDLE64 agent_handshake = 0)
```

Method **A3** is called to destroy an existing context agent. Upon destruction of a context agent, all hosted references by the agent will be automatically destroyed.

Where:

- **private_context**: Private handle of the context to be destroyed.
- ***agent_handshake***: Handshake to be processed by the manager of the context being destroyed.

RETURN:

- **TITAN_RESULT_OK**: The context has been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REGISTER A CONTEXT AGENT MANAGER (A4):

```
titan_result_t RegisterManager(TITAN_HANDLE64
private_context, const atlas _manager_t *agent_manager,
TITAN_HANDLE64 agent_handshake = 0)
```

Method **A4** is called to register a context manager.

Where:

- **private_context**: Private handle of the context whose manager is being registered.
- **agent_manager**: Optional data structure of the registering manager, where:
  - **Agent**: Pointer to the context manager method (See *PX:AtlasContextManagerMethod*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***agent_handshake***: Handshake to be processed by current manager of the context.

RETURN:

- **TITAN_RESULT_OK**: Manager has been successfully registered.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

JOIN AN ATLAS CLONE (A5):

```
titan_result_t JoinClone(TITAN_HANDLE64
private_context, const TITAN_STRING clone_name,
TITAN_HANDLE64 agent_handshake = 0, TITAN_HANDLE64
simulation_handshake = 0)
```

Method **A5** is called by a context agent to join an Atlas clone.

Where:

- **private_context**: Private handle of the agent context joining the simulation clone.
- **clone_name**: Name of the simulation process being joined. This process must be hosted by the simulation in the domain of the joining reference.
- ***agent_handshake***: Handshake to be processed by the manager of the agent joining the clone on behalf of the reference.
- ***simulation_handshake***: Handshake to be processed by the simulation manager of the clone being joined.

RETURN:

- **TITAN_RESULT_OK**: The context has successfully joined the clone.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

LEAVE AN ATLAS CLONE (A6):

```
titan_result_t LeaveClone(TITAN_HANDLE64
private_context, TITAN_HANDLE64 agent_handshake = 0,
TITAN_HANDLE64 simulation_handshake = 0)
```

Method **A6** is called by a context agent to leave an Atlas clone.

Where:

- **private_context**: Private handle of the agent leaving the simulation clone.
- ***agent_handshake***: Handshake to be processed by the manager of the agent leaving the clone on behalf of the reference.
- ***simulation_handshake***: Handshake to be processed by the simulation manager of the clone hosting the context agent.

RETURN:

- **TITAN_RESULT_OK:** The context has successfully left its hosting clone.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

ENTER A ZONE (A7):

```
titan_result_t EnterZone(TITAN_HANDLE64
private_reference, const TITAN_STRING domain_name,
const TITAN_STRING zone_name, TITAN_HANDLE64
domain_handshake = 0, TITAN_HANDLE64 zone_handshake =
0, TITAN_HANDLE64 agent_handshake = 0)
```

Method **A7** is called by a context agent to enter a context zone.

Where:

- **private_reference**: Private handle of the reference whose context agent will be entering a new zone.
- **domain_name**: Name of the domain the context agent will be joining.
- **zone_name**: Name of the zone within the specified domain that the context agent will be joining.
- ***domain_handshake***: Handshake to be processed by the manager of the domain being joined.
- ***zone_handshake***: Handshake to be processed by the manager of the zone being joined.
- ***agent_handshake***: Handshake to be processed by the manager of the entering context agent.

RETURN:

- **TITAN_RESULT_OK**: The context agent has joined the zone successfully.
- **TITAN_RESULT_***: See Atlas Errors for more information.

ENTER A ZONE (A8):

```
titan_result_t _EnterZone(TITAN_HANDLE64
private_context, TITAN_HANDLE64 domain_name_handle,
TITAN_HANDLE64 zone_name_handle, TITAN_HANDLE64
domain_handshake = 0, TITAN_HANDLE64 zone_handshake =
0, TITAN_HANDLE64 agent_handshake = 0)
```

Method **A8** is called by a context agent to enter a context zone.

Where:

- **private_reference**: Private handle of the reference whose context agent will be joining a zone.
- **domain_name_handle**: Handle of the name of the domain the context agent will be joining.
- **zone_name_handle**: Handle of the name of the zone within the specified domain that the context agent will be joining.
- ***domain_handshake***: Handshake to be processed by the manager of the domain being joined.
- ***zone_handshake***: Handshake to be processed by the manager of the zone being joined.
- ***agent_handshake***: Handshake to be processed by the manager of the entering context agent.

RETURN:

- **TITAN_RESULT_OK**: The context agent has joined the zone successfully.
- **TITAN_RESULT_*****: See Atlas Errors for more information.

EXIT A ZONE (A9):

```
titan_result_t ExitZone(TITAN_HANDLE64 private_context,
const TITAN_STRING zone_name, TITAN_HANDLE64
agent_handshake = 0)
```

Method **A9** is used by context agents to exit a context zone.

Where:

- **private_reference**: Private handle of the reference whose context agent will be leaving a zone.
- **domain_name**: Name of the domain the context agent will be leaving.
- **zone_name**: Name of the zone within the specified domain that the context agent will be leaving.
- ***agent_handshake***: Handshake to be processed by the manager of the leaving context.

RETURN:

- **TITAN_RESULT_OK**: The context agent has successfully left the zone.
- **TITAN_RESULT_***: See Atlas Errors for more information.

EXIT A ZONE (AA):

```
titan_result_t _ExitZone(TITAN_HANDLE64
private_context, TITAN_HANDLE64 zone_name_handle,
TITAN_HANDLE64 agent_handshake = 0)
```

Method **AA** is used by context agents to exit a context zone.

Where:

- **private_context**: Private handle of the context agent leaving a zone.
- **domain_name_handle**: Handle of the name of the domain the context agent will be leaving.
- **zone_name_handle**: Handle of the name of the zone within the specified domain that the context agent will be leaving.
- ***agent_handshake***: Handshake to be processed by the manager of the exiting context agent.

RETURN:

- **TITAN_RESULT_OK**: The context agent has left the zone successfully.
- **TITAN_RESULT_*****: See Atlas Errors for more information.

COMMUNICATE IN NATURAL LANGUAGE (AB):

```
titan_result_t Tell(TITAN_HANDLE64 private_reference,
TITAN_HANDLE64 expert_handle, TITAN_HANDLE64 say_type,
const TITAN_STRING expression, TITAN_HANDLE64
user_data, TITAN_HANDLE64 expert_handshake = 0)
```

A context agent can communicate a natural language expression to another agent using method **AB**.

Where:

- **private_reference**: Private handle of the source reference of the message.
- **expert_handle**: Public handle of the destination reference of the message.
- **say_type**: Type of message being transmitted, where:
  - **TITAN_A0_VALIDATE**: Agent should validate the expression.
  - **TITAN_A0_TRANSLATE**: Agent should translate the expression.
  - **TITAN_A0_REMEMBER**: Agent should remember the expression.
  - **TITAN_A0_FORGET**: Agent should forget the expression.
  - **TITAN_A0_SIMULATE**: Agent should simulate the expression.
- **expression**: Text containing a sentence written in natural language.
- **user_data**: Handle of the user data that will be sent back to the agent upon resolving the statement (See *AZ: Agent Hear Method).*
- ***expert_handshake***: Handshake to be processed by the manager of the expert receiving the statement.

RETURN:

- **TITAN_RESULT_OK**: The message has been sent successfully.
- **TITAN_RESULT_*****: See Atlas Errors for more information.

SEND A COMMUNICATION DIRECTIVE (AC):

```
titan_result_t Ping(TITAN_HANDLE64 private_reference,
TITAN_HANDLE64 expert_handle, TITAN_HANDLE64 tell_type,
TITAN_HANDLE64 directive, TITAN_HANDLE64 user_data,
TITAN_HANDLE64 expert_handshake = 0)
```

Method **AC** is used to ping/emit a directive in the zones entered by the calling reference.

Where:

- **private_reference**: Private handle of the source reference of the message.
- **expert_handle**: Public handle of the destination reference of the message.
- **tell_type**: Type of message being transmitted, where:
  - **TITAN_A0_VALIDATE:** Agent should validate the directive.
  - **TITAN_A0_TRANSLATE:** Agent should translate the directive.
  - **TITAN_A0_REMEMBER:** Agent should remember the directive.
  - **TITAN_A0_FORGET:** Agent should forget the directive.
  - **TITAN_A0_SIMULATE:** Agent should simulate the directive.
- **directive**: Handle of the directive being transmitted.
- **user_data**: Handle of the user data that will be sent back to the agent upon resolving the statement (See *AW: Agent Observe Method).*
- ***expert_handshake***: Handshake to be processed by the manager of the expert receiving the statement.

RETURN:

- **TITAN_RESULT_OK**: The message has been sent successfully.
- **TITAN_RESULT_***: See Atlas Errors for more information.

SEND A BINARY COMMUNICATION MESSAGE (AD):

```
titan_result_t Transmit(TITAN_HANDLE64
private_reference, TITAN_HANDLE64 expert_handle,
TITAN_HANDLE64 ping_type, TITAN_HANDLE64 ping_data,
TITAN_HANDLE64 user_data, TITAN_HANDLE64
expert_handshake = 0)
```

A context agent can send a datagram message to a receiving agent using method **AD**.

Where:

- **private_reference**: Private handle of the source reference of the message.
- **expert_handle**: Public handle of the destination reference of the message.
- **ping_type**: Type of message being transmitted, where:
  - **TITAN_A0_VALIDATE**: Agent should validate ping data.
  - **TITAN_A0_TRANSLATE**: Agent should translate ping data.
  - **TITAN_A0_REMEMBER**: Agent should remember ping data.
  - **TITAN_A0_FORGET**: Agent should forget ping data.
  - **TITAN_A0_SIMULATE**: Agent should simulate ping data.
- **ping_data**: Handle of the data transmitted.
- **user_data**: Handle of the user data that will be sent back to the agent upon resolving the statement (See *AY: Agent Receive Method).*
- ***expert_handshake***: Handshake to be processed by the manager of the expert receiving the statement.

RETURN:

- **TITAN_RESULT_OK**: The message has been sent successfully.
- **TITAN_RESULT_*****: See Atlas Errors for more information.

EXECUTE A SCRIPT (AE):

```
titan_result_t ExecuteScript(TITAN_HANDLE64
private_reference, const TITAN_STRING script_name,
TITAN_HANDLE64 domain_handshake = 0)
```

Method **AE** is called to execute a script loaded from a file on behalf of a calling context agent**.**

Where:

- **private_reference**: Private handle of the reference requesting the script to be executed.
- **script_name**: Path to the plain-text script file on disk.
- ***domain_handshake***: Handshake to be processed by the domain hosting the reference requesting the script to be executed.

RETURN:

- **TITAN_RESULT_OK**: The script was successfully executed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE AN ATTACHMENT (AF):

```
TITAN_HANDLE64 GetAttachment(TITAN_HANDLE64
private_reference, const TITAN_STRING attachment_name)
```

Method **AF** is called to retrieve attachments associated with the expression told. After a directive has been processed, an attachment might be available**.**

Where:

- **private_reference**: Private handle of the reference retrieving the attachment.
- **attachment_name**: Name of the attachment being retrieved.

RETURN:

- Handle of the attachment reference retrieved upon successful operation.
- 0 upon failure to retrieve attachment reference or if attachment is invalid.

RETRIEVE AN ATTACHMENT (AG):

```
TITAN_HANDLE64 _GetAttachment(TITAN_HANDLE64
private_reference, TITAN_HANDLE64
attachment_name_handle)
```

Method **AG** is called to retrieve attachments associated with the expression told. After a directive has been processed, an attachment might be available**.**

Where:

- **private_reference**: Private handle of the reference retrieving the attachment.
- **attachment_name_handle**: Handle of the name of the attachment being retrieved.

RETURN:

- Handle of the attachment reference retrieved upon successful operation.
- 0 upon failure to retrieve attachment reference or if attachment is invalid.

REGISTER AN EXPERT (AH):

```
titan_result_t RegisterExpert(TITAN_HANDLE64
private_reference, titan_manager_t *expert_manager,
TITAN_HANDLE64 user_data, TITAN_HANDLE64
domain_handshake = 0)
```

A reference can register its hosting context agent as an expert. Method **AH** is called to register a context agent as an expert.

Where:

- **private_reference**: Private handle of the reference registering as an expert.
- **expert_manager**: Optional data structure of the registering manager, where:
  - **Expert**: Pointer to the expert manager method (See *EX: Expert Manager*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- ***domain_handshake***: Handshake to be processed by the domain hosting the reference.

RETURN:

- **TITAN_RESULT_OK:** The reference has been successfully registered as an expert.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

UNREGISTER AN EXPERT (AI):

```
titan_result_t UnregisterExpert(TITAN_HANDLE64
private_reference)
```

When an existing expert reference needs to terminate its services as an expert, it calls method **AI**.

Where:

- **private_reference**: Private handle of the reference unregistering as an expert.

RETURN:

- **TITAN_RESULT_OK:** The reference has been successfully unregistered as an expert.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

OBSERVE ANOTHER AGENT (AJ):

```
titan_result_t Observe(TITAN_HANDLE64 caller_reference,
TITAN_HANDLE64 observed_reference, TITAN_BOOL observe =
TRUE, TITAN_HANDLE64 domain_handshake = 0,
TITAN_HANDLE64 observed_handshake = 0, TITAN_HANDLE64
observer_handshake = 0)
```

A reference can observe another agent if they belong in the same zone. Method **AJ** is called by a reference concept to observe another reference context agent.

Where:

- **caller_reference**: Private handle to the observer reference.
- **observed_reference**: Public handle to the reference whose context agent will be observed.
- ***observe*:** Set to TRUE when requesting to observe; FALSE when no longer observing.
- ***domain_handshake***: Handshake to be processed by the domain hosting the observed reference.
- ***observed_handshake***: Handshake to be processed by the context agent hosting the observed reference.
- ***observer_handshake***: Handshake to be processed by the context agent hosting the observer reference.

RETURN:

- **TITAN_RESULT_OK**:
  - **observe == true**: The calling reference has started observing the requested reference.
  - **observe == false**: The calling reference has stopped observing the requested reference.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

MANAGE CONTEXT (AX:ATLAS AGENT MANAGER METHOD):

```
TITAN_METHOD_RESULT MyAgentManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

The reference manager is the axiom-supplied method responsible for validating reference requests. This method is registered through a call to *A4:RegisterManager*. All reference requests that require a handshake will be evaluated by this manager method prior to processing the requests.

Where:

- **caller_reference**: Private handle of the reference making the domain request (See *Reference*).
- **request_type**: Type of request, where:
  - **TITAN_AGENT_DESTROY**: Request domain destruction.
  - **TITAN_AGENT_JOIN**: Request for the context agent to join a clone.
  - **TITAN_AGENT_LEAVE**: Request for the context agent to leave a clone.
  - **TITAN_AGENT_ENTER_DOMAIN**: Request for the context agent to enter a domain.
  - **TITAN_AGENT_EXIT_DOMAIN**: Request for the context agent to exit a domain.
  - **TITAN_AGENT_ENTER_ZONE**: Request for the context agent to enter a zone.
  - **TITAN_AGENT_EXIT_ZONE**: Request for the context agent to exit a zone.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *A4:RegisterManager*).
- **message_data**: Message data is dependent on message type (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the domain to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the domain not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

III. Reference Interface:

Introduction:

Instance data is introduced to Atlas in the form of references.

A reference is a tag which links an instance handle (an element), its hosting concept, and a label together.

Annotation:

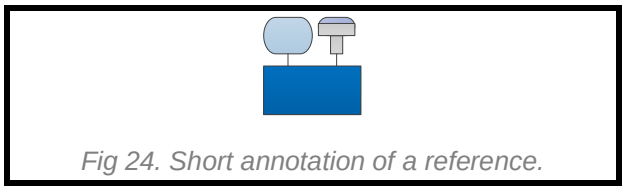


Fig 24. Short annotation of a reference.

Reference Interface:

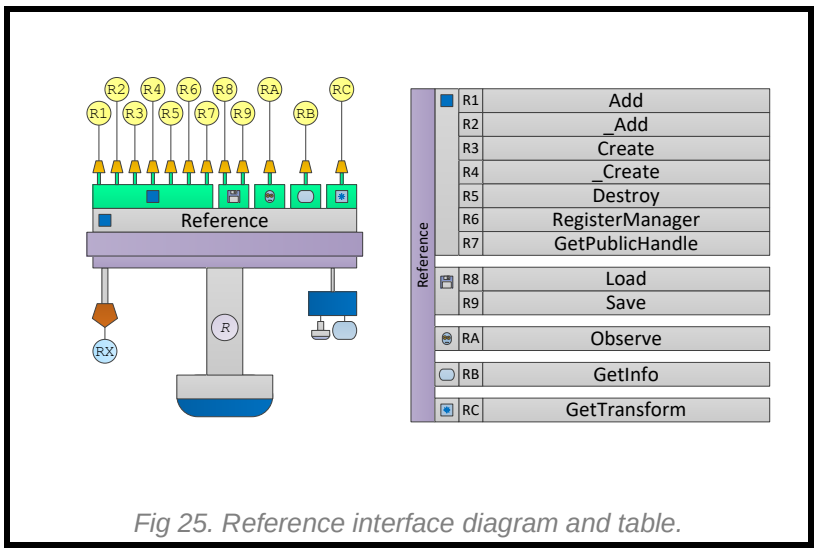


Fig 25. Reference interface diagram and table.

ADD A REFERENCE (R1):

```
TITAN_HANDLE64 Add(TITAN_HANDLE64 calling_reference,
const TITAN_STRING label, TITAN_HANDLE64 concept,
TITAN_HANDLE64 instance, TITAN_HANDLE64 element = 0,
titan_agency_t agency = TITAN_AGENCY_NONE,
TITAN_HANDLE64 agent_handshake = 0)
```

Method **R1** is called to add a reference of an existing instance to the domain. The added reference will be supplied with an existing instance and an element.

Where:

- **calling_reference**: Private handle of the reference requesting the new reference to be added.
- **reference_name**: Name of the new reference.
- **concept:** Supplied concept responsible for new reference's instance (and element).
- **instance**: Supplied instance of the new reference.
- ***element***: Supplied handle of the new reference.
- ***agency***: Agency of the reference, where:
  - **TITAN_AGENCY_NONE**: Calling reference has no access to cognitive services.
  - **TITAN_AGENCY_HIDDEN**: Calling reference is not visible to agents and other references.
  - **TITAN_AGENCY_ZONE**: Calling reference can modify agent zone.
  - **TITAN_AGENCY_DOMAIN**: Calling reference can modify agent domain.
  - **TITAN_AGENCY_SAY**: Calling reference can say to another agent on behalf of agent.
  - **TITAN_AGENCY_TELL**: Calling reference can tell another agent on behalf of agent.
  - **TITAN_AGENCY_PING**: Calling reference can ping another agent on behalf of agent.
  - **TITAN_AGENCY_CREATE**: Calling reference can create another reference under agent.
  - **TITAN_AGENCY_LABEL**: Calling reference can access labels of agent.
  - **TITAN_AGENCY_CONCRETE**: Calling reference can control the concrete features agent.
- ***agent_handshake***: Handshake to be processed by the hosting context agent prior to creating the reference (See *AX: ContextAgentManager*).

RETURN:

- Private handle to the reference created upon successful operation.

- 0 upon failing to create the reference.

ADD A REFERENCE (R2):

```
TITAN_HANDLE64 _Add(TITAN_HANDLE64 private_reference,
TITAN_HANDLE64 label_handle, TITAN_HANDLE64
concept_handle, TITAN_HANDLE64 instance, TITAN_HANDLE64
element = 0, titan_agency_t agency = TITAN_AGENCY_NONE,
TITAN_HANDLE64 agent_handshake = 0)
```

Method **R2** is called to add a reference of an existing instance to the domain. The added reference will be supplied with an existing instance and an element.

Where:

- **private_reference**: Private handle of the reference requesting the new reference to be added.
- **reference_name_handle**: Name handle of the reference.
- **concept:** Supplied concept responsible for this reference's instance (and element).
- **instance**: Supplied instance of the reference.
- ***element***: Supplied handle of the reference.
- ***agency***: Agency of the reference, where:
  - **TITAN_AGENCY_NONE**: Calling reference has no access to cognitive services.
  - **TITAN_AGENCY_HIDDEN**: Calling reference is not visible to agents and other references.
  - **TITAN_AGENCY_ZONE**: Calling reference can modify agent zone.
  - **TITAN_AGENCY_DOMAIN**: Calling reference can modify agent domain.
  - **TITAN_AGENCY_SAY**: Calling reference can say to another agent on behalf of agent.
  - **TITAN_AGENCY_TELL**: Calling reference can tell another agent on behalf of agent.
  - **TITAN_AGENCY_PING**: Calling reference can ping another agent on behalf of agent.
  - **TITAN_AGENCY_CREATE**: Calling reference can create another reference under agent.
  - **TITAN_AGENCY_LABEL**: Calling reference can access labels of agent.
  - **TITAN_AGENCY_CONCRETE**: Calling reference can control the concrete features agent.
- ***agent_handshake***: Handshake to be processed by the context agent prior to creating the reference (See *AX:AtlasAgentManager*).

RETURN:

- Private handle to the reference created upon successful operation.
- 0 upon failing to create the reference.

CREATE AN INSTANCED REFERENCE (R3):

```
TITAN_HANDLE64 Create(TITAN_HANDLE64 private_reference,
const TITAN_STRING name, TITAN_HANDLE64 concept,
TITAN_ULONG num_elements = 1, titan_agency_t agency =
TITAN_AGENCY_NONE, TITAN_HANDLE64 instance_params = 0,
TITAN_HANDLE64 agent_handshake = 0, TITAN_HANDLE64
instance_handshake = 0)
```

Method **R3** is called to create a reference of a new instance to the domain. The created reference will generate a new instance of specified element(s).

Where:

- **private_reference**: Private handle of the reference creating the new instance.
- **label:** Name of the new reference.
- **concept**: Handle of the concept whose instance is being created.
- **num_elements**: Number of elements to be allocated in the instance. Default is 1.
- **agency**: Agency of the reference, where:
  - **TITAN_AGENCY_NONE**: Calling reference has no access to cognitive services.
  - **TITAN_AGENCY_HIDDEN**: Calling reference is not visible to agents and other references.
  - **TITAN_AGENCY_ZONE**: Calling reference can modify agent zone.
  - **TITAN_AGENCY_DOMAIN**: Calling reference can modify agent domain.
  - **TITAN_AGENCY_SAY**: Calling reference can say to another agent on behalf of agent.
  - **TITAN_AGENCY_TELL**: Calling reference can tell another agent on behalf of agent.
  - **TITAN_AGENCY_PING**: Calling reference can ping another agent on behalf of agent.
  - **TITAN_AGENCY_CREATE**: Calling reference can create another reference under agent.
  - **TITAN_AGENCY_LABEL**: Calling reference can access labels of agent.
  - **TITAN_AGENCY_CONCRETE**: Calling reference can control the concrete features agent.
- **instance_params**: Instance creation parameters (See *I1:Create*).
- **agent_handshake**: Handshake to be processed by the context agent prior to creating the reference (See *CX:AtlasContextManagerMethod*).
- **instance_handshake**: Handshake to be processed by the instantiator hosting the instance being created.

RETURN:

- Private handle to the reference created upon successful operation.
- 0 upon failing to create the reference.

CREATE AN INSTANCED REFERENCE (R4):

```
TITAN_HANDLE64 _Create(TITAN_HANDLE64
private_reference, TITAN_HANDLE64 label_handle,
TITAN_HANDLE64 concept, TITAN_ULONG num_elements = 1,
titan_agency_t agency = TITAN_AGENCY_NONE,
TITAN_HANDLE64 instance_params = 0, TITAN_HANDLE64
agent_handshake = 0, TITAN_HANDLE64 instance_handshake
= 0)
```

Method **R4** is called to create a reference of a new instance to the domain. The created reference will generate a new instance of specified element(s)**.**

Where:

- **private_reference**: Private handle of the reference creating the new instance.
- **label_handle:** Handle of the name of the new reference.
- **concept**: Handle of the concept whose instance is being created.
- **num_elements**: Number of elements to be allocated in the instance. Default is 1.
- **agency**: Agency of the reference, where:
  - **TITAN_AGENCY_NONE**: Calling reference has no access to cognitive services.
  - **TITAN_AGENCY_HIDDEN**: Calling reference is not visible to agents and other references.
  - **TITAN_AGENCY_ZONE**: Calling reference can modify agent zone.
  - **TITAN_AGENCY_DOMAIN**: Calling reference can modify agent domain.
  - **TITAN_AGENCY_SAY**: Calling reference can say to another agent on behalf of agent.
  - **TITAN_AGENCY_TELL**: Calling reference can tell another agent on behalf of
  - agent.
  - **TITAN_AGENCY_PING**: Calling reference can ping another agent on behalf of agent.
  - **TITAN_AGENCY_CREATE**: Calling reference can create another reference under agent.
  - **TITAN_AGENCY_LABEL**: Calling reference can access labels of agent.
  - **TITAN_AGENCY_CONCRETE**: Calling reference can control the concrete features agent.
- ***agent_handshake***: Handshake to be processed by the context agent prior to creating the reference (See *AX:AtlasAgentManagerMethod*).

- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being created.

RETURN:

- Private handle to the reference created upon successful operation.
- 0 upon failing to create the reference.

DESTROY A REFERENCE (R5):

titan_result_t Destroy(TITAN_HANDLE64 private_reference, TITAN_HANDLE64 agent_handshake = 0, TITAN_HANDLE64 instance_handshake = 0)

Method **R5** is called to destroy existing references. Destroying references will destroy instance data of created references (with *R3: Create* or *R4:_Create*). Added references (with *R1:Add* or *R2:_Add*) will not have their instances destroyed.

Where:

- **private_reference**: The private handle of the reference being destroyed.
- **agent_handshake**: Handshake to be processed by the context agent hosting the reference.
- **instance_handshake**: Handshake to be processed by the instance manager hosting the reference.

RETURN:

- **TITAN_RESULT_OK:** The reference and its instance have been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REGISTER A REFERENCE MANAGER (R6):

```
titan_result_t RegisterManager(TITAN_HANDLE64
private_reference, titan_manager_t *reference_manager,
TITAN_HANDLE64 reference_handshake = 0)
```

Each reference can have its own manager. Method **R6** is called to register a manager for a specified reference.

Where:

- **private_reference**: Private handle of the reference whose manager is being registered.
- **reference_manager**: Optional data structure of the registering manager, where:
  - **Reference**: Pointer to the reference manager method (See *RX:AtlasReferenceManagerMethod*).
  - **manager_data**: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.
- **reference_handshake**: Handshake to be processed by current manager of the reference.

RETURN:

- **TITAN_RESULT_OK**: Manager has been successfully registered.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

REQUEST A PUBLIC HANDLE OF A REFERENCE (R7):

```
TITAN_HANDLE64 GetPublicHandle(TITAN_HANDLE64
private_reference)
```

Communication methods require the public handle of a reference rather the private handle. Method **R7** is called to retrieve the public handle of a reference.

Where:

- **private_reference**: The private handle of the reference whose public handle will be retrieved.

RETURN:

- Public handle of supplied private reference handle.
- 0 upon failure

LOAD A REFERENCE (R8):

```
titan_result_t Load(titan_file_t *file, TITAN_HANDLE64
calling_reference, const TITAN_STRING path_name,
TITAN_HANDLE64 concept_handle, const TITAN_STRING name,
TITAN_HANDLE64 &out_private_reference, TITAN_HANDLE64
agent_handshake = 0, TITAN_HANDLE64 instance_handshake
= 0)
```

Method **R8** is called to load a reference from a supplied file. A reference can be read from an open file <*file*> or loaded from a closed file <*path_name*>.

Where:

- **file**: Atlas data file pointer. If the *file* parameter is not defined, the *path_name* parameter should contain the file path (See *Files*).
- **calling_reference:** Private handle to the calling reference requesting the load.
- **path_name**: If file is undefined (NULL), then the reference should be loaded from file at *path_name*. It is up to the interpretation to decide how to open the file at *path_name*.
- **concept_handle**: Handle of the concept responsible for loading the reference.
- **name**: Name of the loaded reference.
- **out_private_reference**: User-supplied handle variable where the newly loaded private reference ID will be stored.
- ***agent_handshake*:** Handshake to be processed by the manager of the context agent hosting the reference being loaded.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the reference being loaded.

RETURN:

- **TITAN_RESULT_OK**: The reference has been loaded successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SAVE A REFERENCE (R9):

```
titan_result_t Save(titan_file_t *file, TITAN_HANDLE64
calling_reference, const TITAN_STRING path_name,
TITAN_HANDLE64 public_reference, TITAN_HANDLE64
agent_handshake = 0, TITAN_HANDLE64 instance_handshake
= 0)
```

Method **R9** is called to save an existing reference to a supplied file. A reference can be written to an open file <*file*> or saved to a closed file <*path_name*>.

Where:

- **file**: Atlas data file pointer. If the *file* parameter is not defined, the *path_name* parameter should contain the file path (See *Files*).
- **calling_reference:** Private handle to the calling reference requesting the save.
- **path_name**: If file is undefined (NULL), then the reference should be saved to file at *path_name*. It is up to the interpretation to decide how to open the file at *path_name*.
- ***public_reference***: Public handle of the reference being saved.
- ***agent_handshake*:** Handshake to be processed by the manager of the context agent hosting the reference being saved.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the reference being saved.

RETURN:

- **TITAN_RESULT_OK**: The reference has been saved successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

OBSERVE A REFERENCE (RA):

```
titan_result_t Observe(TITAN_HANDLE64
private_reference, TITAN_HANDLE64 observed_reference,
TITAN_BOOL observe = TRUE, TITAN_HANDLE64
domain_handshake = 0, TITAN_HANDLE64 observed_handshake
= 0, TITAN_HANDLE64 observer_handshake = 0)
```

A reference can observe another instance if they belong in the same zone. Method **RA** is called by a reference concept to observe another reference.

Where:

- **private_reference**: Private handle to the observer reference.
- **observed_reference**: Public handle to the observed reference.
- ***observe*:** Set to TRUE when requesting to observe; FALSE when no longer observing.
- ***domain_handshake***: Handshake to be processed by the domain hosting the observed reference.
- ***observed_handshake***: Handshake to be processed by the observed reference.
- ***observer_handshake***: Handshake to be processed by the context agent hosting the observer reference.

RETURN:

- **TITAN_RESULT_OK**:
  - **observe == true**: The calling reference has started observing the requested reference.
  - **observe == false**: The calling reference has stopped observing the requested reference.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET REFERENCE DATA (RB):

```
titan_result_t GetInfo(TITAN_HANDLE64 public_reference,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Reference information can be retrieved from a given reference public handle. Method **RB** is called to retrieve reference information based on supplied query flags.

Where:

- **public_reference**: Public handle of the reference whose information is being retrieved.
- **reference_info**: A user-supplied structure where reference information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **relation**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the relation or preposition.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:

        - **handle_value**: Amount data is a 64-bit handle.
        - **ulong_value**: Amount data is a long unsigned 64-bit integer.
        - **long_value**: Amount data is a long signed 64-bit integer.
        - **double_value**: Amount data is a double-precision floating point.
        - **float_value**: Amount data is a single-precision floating point.
        - **string_value**: Amount data is a null-terminate character string.
        - **raw_value**: Amount data is an opaque 64-bit memory pointer.
        - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
        - **uint_value**: Amount data is an unsigned 32-bit integer.
        - **int_value**: Amount data is a signed 32-bit integer.
        - **ushort_value**: Amount data is a short unsigned 16-bit integer.
        - **short_value**: Amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference.
    - **element**: Returned element of the reference.
- **query_flags**: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.
    - **TITAN_GET_RELATION**: Retrieve reference relation or preposition.

RETURN:

- **TITAN_RESULT_OK:** Reference information has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET THE SPATIAL TRANSFORM OF A REFERENCE (RC):

```
titan_result_t GetTransform(TITAN_HANDLE64
public_reference, titan_mat4_t &transform,
TITAN_HANDLE64 simulation_handshake = 0, TITAN_HANDLE64
agent_handshake = 0)
```

References that have location attributes can have their transform matrix inspected. Method **RC** is called when a reference transform is requested by a concept. If it exists, the reference transform is copied into the memory buffer supplied to the method.

Where:

- **public_reference**: Public handle of the reference whose transform is being retrieved.
- **transform**: Atlas *mat4* structure where the reference's transform matrix will be copied.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the reference.
- ***agent_handshake***: Handshake to be processed by the context hosting the reference.

RETURN:

- **TITAN_RESULT_OK:** The reference transform has been retrieved successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

MANAGE A REFERENCE (RX:ATLAS REFERENCE MANAGER METHOD):

```
TITAN_METHOD_RESULT MyReferenceManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake)
```

The reference manager is the axiom-supplied method responsible for validating reference requests. This method is registered through a call to *R6:RegisterManager*. All reference requests that require a handshake will be evaluated by this manager method prior to processing the requests. The method is of the form:

Where:

- **caller_reference**: Private handle of the reference making the request.
- **request_type**: Type of request, where:
  - **TITAN_REFERENCE_DESTROY**: Request reference destruction.
  - **TITAN_REFERENCE_GET_TRANSFORM**: Request for a reference transformation matrix.
  - **TITAN_REFERENCE_REGISTER_MANAGER**: Request for a reference manager to be registered.
  - **TITAN_REFERENCE_LOAD**: Request for a reference to be loaded.
  - **TITAN_REFERENCE_SAVE**: Request for a reference to be saved.
  - **TITAN_REFERENCE_OBSERVE**: Request to observe reference.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *R6:RegisterManager*).
- **message_data**: Message data is dependent on message type (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the reference to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the reference not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

IV. Instance Interface:

Information:

Annotation:

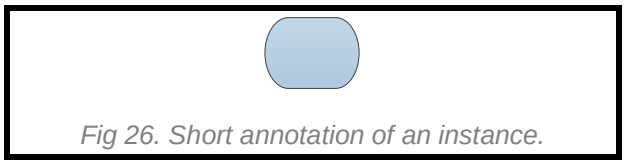


Fig 26. Short annotation of an instance.

Instance Interface

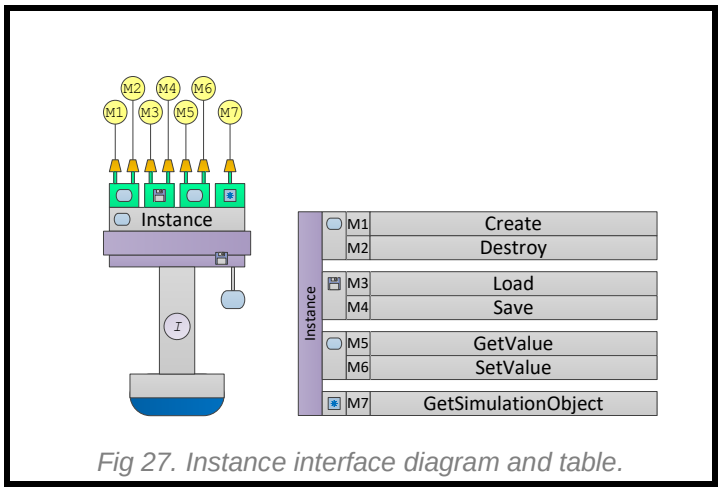


Fig 27. Instance interface diagram and table.

CREATE AN INSTANCE (M1):

```
titan_result_t Create(TITAN_HANDLE64 calling_reference,
TITAN_HANDLE64 concept, TITAN_HANDLE64 &out_instance,
TITAN_ULONG num_elements = 1, TITAN_HANDLE64
instance_params = 0, TITAN_HANDLE64 instance_handshake
= 0)
```

Although references are used to label instances in a cognitive space, some instances are not labeled. Such instances reside deeper as instances of sub-concepts of a reference. A top concept can request its subordinate concepts to create instances. Method **M1** is called to create an instance of the concept specified by handle.

Where:

- **calling_reference**: Private handle to the reference creating the instance.
- **concept**: Handle of the concept whose instance will be created.
- **out_instance:** Created instance handle will be stored here upon success.
- ***num_elements***: Number of elements to be allocated in the instance. Default is 1.
- ***instance_params***: Instance creation parameters (See *I1:Create*).
- ***instance_handshake***: Handshake to be processed by the instantiator creating the instance.

RETURN:

- **TITAN_RESULT_OK:** The instance has been created successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

DESTROY AN INSTANCE (M2):

```
titan_result_t Destroy(TITAN_HANDLE64 concept,
TITAN_HANDLE64 instance, TITAN_HANDLE64
instance_handshake = 0)
```

Method **M2** is used to destroy a known instance of a concept specified by handle. The destroyed instance is responsible for clearing any sub-instances or simulation objects it might have allocated.

Where:

- **concept**: Handle of the concept whose instance will be destroyed.
- **instance:** Handle of the instance being destroyed.
- ***instance_handshake***: Handshake to be processed by the instantiator destroying the instance.

RETURN:

- **TITAN_RESULT_OK:** The instance has been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

LOAD AN INSTANCE (M3):

```
titan_result_t Load(TITAN_HANDLE64 calling_reference,
titan_file_t *file, const TITAN_STRING path_name, const
TITAN_STRING concept_name, TITAN_HANDLE64
&out_instance, TITAN_HANDLE64 &out_element,
TITAN_HANDLE64 instance_handshake = 0)
```

Method **M3** is called to load an instance of a concept specified by handle. This method is typically called by the top instance to its load subordinate concept instances.

Where:

- **calling_reference:** Private handle to the calling reference requesting the load.
- **file**: Atlas data file pointer. If the *file* parameter is not defined, the *path_name* parameter should contain the file path (See *Files*).
- **path_name**: If file is undefined (NULL), then the instance should be loaded from file at *path_name*. It is up to the interpretation to decide how to open the file at *path_name*.
- **concept_name**: Name of the concept responsible for loading the instance.
- **out_instance**: Loaded instance handle will be stored here upon success.
- **out_element**: Loaded element handle will be stored here upon success.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being loaded.

RETURN:

- **TITAN_RESULT_OK**: The instance has been loaded successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SAVE AN INSTANCE (M4):

```
titan_result_t Save(TITAN_HANDLE64 calling_reference,
titan_file_t *file, const TITAN_STRING path_name,
TITAN_HANDLE64 concept, TITAN_HANDLE64 instance,
TITAN_HANDLE64 element = 0, TITAN_HANDLE64
instance_handshake = 0)
```

Method **M4** is called to save an existing instance of a concept specified by handle. This method is typically called by the top instance to its save subordinate concept instances.

Where:

- **calling_reference:** Private handle to the calling reference requesting the save.
- **concept**: Handle of the concept responsible for saving the instance.
- **file**: Atlas data file pointer. If the *file* parameter is not defined, the *path_name* parameter should contain the file path (See *Files*).
- **path_name**: If file is undefined (NULL), then the instance should be saved from file at *path_name*. It is up to the interpretation to decide how to open the file at *path_name*.
- **instance**: Handle of the instance being saved.
- ***element***: Handle of the element being saved.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being saved.

RETURN:

- **TITAN_RESULT_OK**: The instance has been saved successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

GET THE VALUE OF AN INSTANCE (M5):

```
titan_result_t GetValue(TITAN_HANDLE64 concept,
TITAN_HANDLE64 instance, TITAN_HANDLE64 element,
titan_value_t &value, TITAN_HANDLE64 instance_handshake
= 0)
```

Instances can have multiple values embedded into them. Method **M5** is used to get the value of an instance of a concept specified by handle. This method should be handled by *I9:Sim_GetValue* in the instantiator of queried instance.

Where:

- **concept**: Handle of the concept responsible for retrieving the instance value.
- **instance**: Handle of the instance whose value is being retrieved.
- **element:** Handle of the element whose value is being retrieved.
- **value**: A titan_value_t structure, where:
  - **name_handle**: Handle of the name of the value.
  - **amount.format_handle**: Handle of the amount format (type or units).
  - **amount.*_value**: Union of the amount where:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the referenced instance.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being queried.

RETURN:

- **TITAN_RESULT_OK:** The instance value has been retrieved successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

SET THE VALUE OF AN INSTANCE (M6):

```
titan_result_t SetValue(TITAN_HANDLE64 concept,
TITAN_HANDLE64 instance, TITAN_HANDLE64 element,
titan_value_t &value, TITAN_HANDLE64 instance_handshake
= 0)
```

Instances can have multiple values embedded into them. Method **M6** is used to set the value of an instance of a concept specified by handle. This method should be handled by *I8:Sim_SetValue* in the instantiator of updated instance.

Where:

- **concept**: Handle of the concept responsible for updating the instance value.
- **instance**: Handle of the instance whose value is being updated.
- **element:** Handle of the element whose value is being updated.
- **value**: A titan_value_t structure, where:
  - **name_handle**: Handle of the name of the value.
  - **amount.format_handle**: Handle of the amount format (type or units).
  - **amount.*_value**: Union of the amount where:
    - **handle_value**: Relation amount data is a 64-bit handle.
    - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
    - **long_value**: Relation amount data is a long signed 64-bit integer.
    - **double_value**: Relation amount data is a double-precision floating point.
    - **float_value**: Relation amount data is a single-precision floating point.
    - **string_value**: Relation amount data is a null-terminate character string.
    - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
    - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
    - **uint_value**: Relation amount data is an unsigned 32-bit integer.
    - **int_value**: Relation amount data is a signed 32-bit integer.
    - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
    - **short_value**: Relation amount data is a short signed 16-bit integer.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being set.

RETURN:

- **TITAN_RESULT_OK:** The instance value has been set successfully.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE THE SIMULATION OBJECT OF AN INSTANCE (M7):

```
titan_result_t GetObject(TITAN_HANDLE64 concept,
TITAN_HANDLE64 instance, TITAN_HANDLE64 element,
TITAN_HANDLE64 &out_object_handle, TITAN_HANDLE64
instance_handshake = 0)
```

Simulation objects of an instance can be retrieved and queried using the attribute interfaces. Method **M7** is used to retrieve the simulation object of an instance.

Where:

- **concept**: Handle of the concept whose instance object is being retrieved.
- **instance**: Handle of the instance whose object is being retrieved.
- **element:** Handle of the element whose object is being retrieved.
- **out_object_handle**: Instance simulation object handle will be stored here upon success.
- ***instance_handshake***: Handshake to be processed by the instantiator hosting the instance being probed.

RETURN:

- **TITAN_RESULT_OK:** The instance simulation object has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

V. Group Interface:

Introduction:

Atlas offers a way to group multiple references together into groups.

A group is a list of references, where each reference is perceived as a group member.

Each group can have no members or an arbitrary number of members.

Like the reference, a group is represented by two 64-bit handles – a control handle for modifying the group, and an access handle for inspecting the group.

Groups are the main components of the directive subjects and objects.

Annotation:

In the overview diagram, groups are represented as the following:

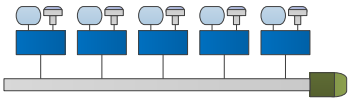


Fig 28. Short annotation of a group.

Group Interface:

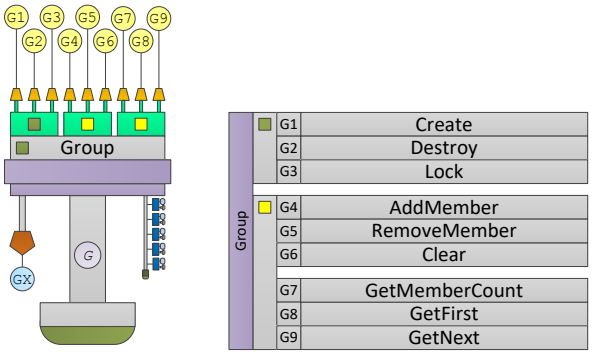


Fig 29. Group interface diagram and table.

Group Interface Call Outs: (Gx methods)

CREATE A GROUP (G1):

```
TITAN_HANDLE64 Create(titan_manager_t *group_manager = NULL)
```

Method **G1** is used to create a group.

Where:

- **group_manager**: Optional data structure of the registering manager, where:
- **Group**: Pointer to the group manager method (See G*X:AtlasGroupManagerMethod*).
- ***manager_data***: Discretionary 64-bit value to be sent to the manager method when the manager is invoked.

RETURN:

- Handle of the group created upon successful operation.
- 0 upon failure to create a group.

DESTROY A GROUP (G2):

```
titan_result_t Destroy(TITAN_HANDLE64 group,
TITAN_HANDLE64 group_handshake = 0)
```

Method **G2** is used to destroy an existing group.

Where:

- **group**: Handle of the group being destroyed.
- ***group_handshake***: Handshake to be processed by the manager hosting the group.

RETURN:

- **TITAN_RESULT_OK:** The group has been successfully destroyed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

LOCK A GROUP (G3):

```
titan_result_t Lock(TITAN_HANDLE64 group,
TITAN_HANDLE64 group_handshake = 0)
```

The **G3** method used to lock an existing group. Locking a group will prevent its content from been modified. A locked group has better speed performance than an unlocked group.

Where:

- **group**: Handle of the group being locked.
- ***group_handshake***: Handshake to be processed by the domain hosting the group.

RETURN:

- **TITAN_RESULT_OK**: The group has been successfully locked.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

ADD A GROUP MEMBER (G4):

```
TITAN_HANDLE64 AddMember(TITAN_HANDLE64 group,
TITAN_HANDLE64 public_reference, TITAN_HANDLE64
group_handshake = 0)
```

Method **G4** is used to add a specified reference as a member of a specified group.

Where:

- **group**: Handle of the group being modified.
- **public_reference**: Public handle of the reference being added to the group.
- ***group_handshake***: Handshake to be processed by the manager hosting the group.

RETURN:

- Handle to the group member added upon successful operation.
- 0 upon failure to add the group member.

REMOVE A GROUP MEMBER (G5):

```
titan_result_t RemoveMember(TITAN_HANDLE64 group,
TITAN_HANDLE64 group_member, TITAN_HANDLE64
group_handshake = 0
```

Method **G5** is used to remove a member from a group if its handle is known.

Where:

- **group**: Handle of the group being modified.
- **group_member**: Handle of the group member to be removed from the group.
- **group_handshake**: Handshake to be processed by the manager hosting the group.

RETURN:

- **TITAN_RESULT_OK:** The group member has been successfully removed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

CLEAR A GROUP (G6):

```
titan_result_t Clear(TITAN_HANDLE64 group,
TITAN_HANDLE64 group_handshake = 0)
```

The **G6** method is used to remove all members of a group.

When a group is destroyed, the references within that group are left intact.

Where:

- **group**: Handle of the group being cleared.
- ***group_handshake***: Handshake to be processed by the manager hosting the group.

RETURN:

- **TITAN_RESULT_OK:** The group has been successfully cleared.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE A GROUP MEMBER COUNT (G7):

```
TITAN_ULONG GetMemberCount(TITAN_HANDLE64 public_group,
TITAN_HANDLE64 group_handshake = 0)
```

Method **G7** is used to retrieve the number of members or references in a group.

Where:

- **public_group**: Public handle of the group being queried.
- **group_handshake**: Handshake to be processed by the manager hosting the group.

RETURN:

- Number of members registered in the specified group.
- 0 upon failure to retrieve the members or if the group is empty.

RETRIEVE THE FIRST GROUP MEMBER (G8):

```
TITAN_HANDLE64 GetFirst(TITAN_HANDLE64 group,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **G8** is used to retrieve the first member in the group.

Where:

- **group**: Handle of the group being queried.
- **reference_info**: A user-supplied structure where member information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **relation**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the relation or preposition.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:
      - **handle_value**: Amount data is a 64-bit handle.
      - **ulong_value**: Amount data is a long unsigned 64-bit integer.

        - **long_value**: Amount data is a long signed 64-bit integer.
        - **double_value**: Amount data is a double-precision floating point.
        - **float_value**: Amount data is a single-precision floating point.
        - **string_value**: Amount data is a null-terminate character string.
        - **raw_value**: Amount data is an opaque 64-bit memory pointer.
        - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
        - **uint_value**: Amount data is an unsigned 32-bit integer.
        - **int_value**: Amount data is a signed 32-bit integer.
        - **ushort_value**: Amount data is a short unsigned 16-bit integer.
        - **short_value**: Amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference.
    - **element**: Returned element of the reference.
- ***query_flags***: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.
    - **TITAN_GET_RELATION**: Retrieve reference relation or preposition.
- ***group_handshake***: Handshake to be processed by the manager hosting the group**.**

RETURN:

- Handle of the first member in the group upon successful operation.
- 0 upon failure to retrieve the first member of the group.

RETRIEVE THE NEXT GROUP MEMBER (G9):

```
TITAN_HANDLE64 GetNext(TITAN_HANDLE64 member,
titan_reference_info_t &reference_info, TITAN_BITMAP64
query_flags = TITAN_GET_DEFAULT)
```

Method **G9** is used to retrieve the successor of a member of a group.

Where:

- **member**: Handle of the group member whose successor is requested.
- **reference_info**: A user-supplied structure where member information will be stored, where:
  - **name**: Text string where the name of the reference will be copied. Set to NULL if no name is requested.
  - **name_max_size**: Maximum number of characters to copy to the *name* string.
  - **concept_name**: Text string where the name of the reference's concept will be copied. Set to NULL if no concept name is requested.
  - **concept_name_max_size**: Maximum number of characters to copy to the *concept_name* string.
  - **value**: A titan_value_t structure, where:
    - **name_handle**: Handle of the name of the value.
    - **amount.format_handle**: Handle of the amount format (type or units).
    - **amount.*_value**: Union of the amount where:
      - **handle_value**: Relation amount data is a 64-bit handle.
      - **ulong_value**: Relation amount data is a long unsigned 64-bit integer.
      - **long_value**: Relation amount data is a long signed 64-bit integer.
      - **double_value**: Relation amount data is a double-precision floating point.
      - **float_value**: Relation amount data is a single-precision floating point.
      - **string_value**: Relation amount data is a null-terminate character string.
      - **raw_value**: Relation amount data is an opaque 64-bit memory pointer.
      - **buffer_value**: Relation amount data is a 64-bit unsigned byte pointer.
      - **uint_value**: Relation amount data is an unsigned 32-bit integer.
      - **int_value**: Relation amount data is a signed 32-bit integer.
      - **ushort_value**: Relation amount data is a short unsigned 16-bit integer.
      - **short_value**: Relation amount data is a short signed 16-bit integer.
  - **amount**: A titan_amount_t structure describing the amount of the reference such that:
    - **format_handle**: Handle of the amount format (type or units).
    - ***_value**: Union of the amount where:
      - **handle_value**: Amount data is a 64-bit handle.
      - **ulong_value**: Amount data is a long unsigned 64-bit integer.

        - **long_value**: Amount data is a long signed 64-bit integer.
        - **double_value**: Amount data is a double-precision floating point.
        - **float_value**: Amount data is a single-precision floating point.
        - **string_value**: Amount data is a null-terminate character string.
        - **raw_value**: Amount data is an opaque 64-bit memory pointer.
        - **buffer_value**: Amount data is a 64-bit unsigned byte pointer.
        - **uint_value**: Amount data is an unsigned 32-bit integer.
        - **int_value**: Amount data is a signed 32-bit integer.
        - **ushort_value**: Amount data is a short unsigned 16-bit integer.
        - **short_value**: Amount data is a short signed 16-bit integer.
    - **instance**: Returned instance of the reference.
    - **element**: Returned element of the reference.
- ***query_flags***: Retrieved data depends on this bitmap, where active bits are interpreted as:
    - **TITAN_GET_NONE**: No query, but names can still be retrieved.
    - **TITAN_GET_REFERENCE**: Retrieve reference ID.
    - **TITAN_GET_INSTANCE**: Retrieve reference instance and element handles.
    - **TITAN_GET_AMOUNT**: Retrieve reference amount.
    - **TITAN_GET_RELATION**: Retrieve reference relation or preposition.
- ***group_handshake***: Handshake to be processed by the manager hosting the group**.**

RETURN:

- Handle of the next member in the group upon successful operation.
- 0 upon failure to retrieve the first member of the group.

MANAGE A GROUP (GX: ATLAS GROUP MANAGER METHOD):

```
TITAN_METHOD_RESULT MyGroupManager(TITAN_HANDLE64
caller_reference, titan_domain_request_t request_type,
TITAN_HANDLE64 user_data, TITAN_HANDLE64 message_data,
TITAN_HANDLE64 handshake = 0)
```

The group manager is responsible for validating group requests.

The group manager (the axiom-supplied method) is registered during group creation (*G1:Create*).

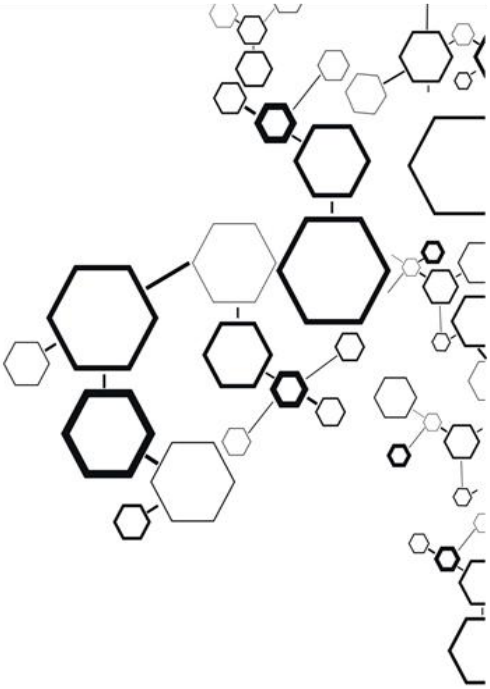
All group requests that require a handshake will be evaluated by this manager method prior to processing the requests.

Where:

- **caller_reference**: Private handle of the reference making the domain request (See *Reference*).
- **request_type**: Type of request, where:
  - **TITAN_GROUP_DESTROY**: Request group destruction.
  - **TITAN_GROUP_LOCK**: Request to lock the group.
  - **TITAN_GROUP_ADD_MEMBER**: Request to add a group member.
  - **TITAN_GROUP_REMOVE_MEMBER**: Request to remove a group member.
  - **TITAN_GROUP_CLEAR**: Request to clear the group.
  - **TITAN_GROUP_GET_MEMBER_COUNT**: Request number of members in the group.
  - **TITAN_GROUP_GET_FIRST_MEMBER**: Request first member in the group.
  - **TITAN_GROUP_GET_NEXT_MEMBER**: Request next member in the group.
- **user_data**: Discretionary 64-bit value supplied by the user during registration (See *G1:Create*).
- **message_data**: Message data is dependent on message type. (See *Manager Messages*).
- ***handshake***: Handshake to be processed by the manager.

RETURN: The expected return values of the manager are:

- **TITAN_RESULT_OK:** Inform the group to proceed with the request.
- **TITAN_RESULT_STOP:** Inform the group not to proceed with the request.
- **TITAN_RESULT_*:** See Atlas Errors for more information.



ATLAS

Attributes API

VI. Attributes Interface:

Introduction:

Attributes are internal features (or sub-concepts) that are typically common to most cognitive concepts in a domain.

Features such as location, size, shape, texture, temperature, pressure, color, mood, material, etc. can be represented internally by *Atlas* for quick deployment but these features can also be augmented or overridden by any concept.

Atlas's attributes can be represented within a concept or across concepts (e.g. changing the shape attribute might affect the size attribute).

These attributes are the equivalent of a standard library in C; they are available for ease of implementation and use, but they can be replaced by user-defined concepts to achieve different results.

A collection of attributes aggregate to a simulation object through one or multiple calls to method *CA:AddAttribute*.

Location Attribute Interface:

Information

The location attribute represents the position, orientation, and scale of the simulation object in a time frame. The location coordinate structure, titan_coordinate_t, is defined as:

- **udpate_flags**: Bitmap of one or multiple update flags, where:
  - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
  - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
  - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
- **trigger_flags**:
  - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
  - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
  - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
- **position**: Destination position of the simulation object relative to its parent, where:
  - **X** is the destination x-coordinate of the simulation object.
  - **Y** is the destination y-coordinate of the simulation object.
  - **Z** is the destination z-coordinate of the simulation object.
  - **T** is the future time (in seconds) when the simulation object will reach the destination position.
- **angle**: Destination angle of the simulation object relative to its parent, where:
  - **pitch** is the destination pitch of the simulation object in radians.
  - **yaw** is the destination yaw of the simulation object in radians.
  - **roll** is the destination roll of the simulation object in radians.
  - **t** is the future time (in seconds) when the simulation object will reach the destination angle.
- **scale**: Destination absolute scale of the simulation object, where:
  - **X** is the destination width of the simulation object.
  - **Y** is the destination height of the simulation object.
  - **Z** is the destination depth of the simulation object.
  - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
- **prev**: Previous coordinate index in an array of coordinates (animation).
- **next**: Next coordinate index in an array of coordinates (animation).
- **trigger**: A trigger data structure, where:
  - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.

- **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
- **Trigger**: Registering trigger method that will be invoked when the destination is reached.

Location trails: Over time, the location of a simulation object might be updated to different spatial positions.

Atlas can remember the location updates if requested. The list of location updates over time is called a trail.

The size of the trail depends on the *complexity* parameter of the creation method of the simulation (See *D7:CreateSimulation*, *D8:_CreateSimulation*).

Annotation:

In the overview diagram, the attribute is depicted as follows:

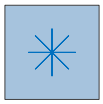


Fig 30. Short annotation of an attribute.

Location Attribute Interface:

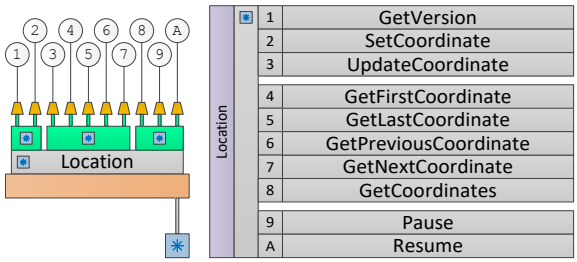


Fig 31. Location attribute interface diagram and table.

GET LOCATION ATTRIBUTE VERSION (1):

```
TITAN_FLOAT GetVersion()
```

Method **Location-1** is used to retrieve the location interface version of current domain simulation.

Where:

RETURN: Version of the location attribute interface.

SET LOCATION ATTRIBUTE COORDINATE (2):

To initialize or set the location attribute of a simulation object, use method:

```
titan_result_t SetCoordinate(TITAN_HANDLE64
object_handle, titan_coordinate_t *coordinate,
titan_mat4_t *transform = NULL, TITAN_BOOL update_trail
= false, TITAN_HANDLE64 simulation_handshake = 0)
```

During instantiation (I1:Sim_Create), a simulation object coordinate attribute can be initialized or set. Method Location-2 is called to set or initialize the coordinate structure of an existing simulation object.

Where:

- **object_handle**: Handle of the simulation object whose location attribute will be set.
- **coordinate:** A pointer to a data structure of type *titan_coordinate_t*, where:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **t** is the future time (in seconds) when the simulation object will reach the destination angle.
  - **scale**: Destination absolute scale of the simulation object, where:

    - **X** is the destination width of the simulation object.
    - **Y** is the destination height of the simulation object.
    - **Z** is the destination depth of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
  - **prev**: Previous coordinate index in an array of coordinates (animation).
  - **next**: Next coordinate index in an array of coordinates (animation).
  - **trigger**: A trigger data structure, where:
    - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
    - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
    - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
- ***update_trail***: Update the coordinate trail when true; do not update the trail otherwise. Coordinate trail coordinates are accessed by methods 4, 5, 6, 7, and 8 of the location interface.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully set.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

UPDATE LOCATION ATTRIBUTE COORDINATE (3):

```
titan_result_t UpdateCoordinate(TITAN_HANDLE64
object_handle, TITAN_HANDLE64 on_flags =
TITAN_COORDINATE_TRS, TITAN_HANDLE64 off_flags = 0,
TITAN_BOOL update_trail = false, TITAN_HANDLE64
simulation_handshake = 0)
```

If a simulation object coordinate has been set, then updating the coordinate is performed using method **Location-3.** This is most commonly called for animating objects or setting virtual scenes.

Where:

- **object_handle**: Handle of the simulation object whose location attribute will be updated.
- **on_flags**: A bitmap of one or multiple of the following options:
  - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
  - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
  - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
- **off_flags**: A bitmap of one or multiple of the following options:
  - **TITAN_COORDINATE_TRANSLATE:** Coordinate will no longer be translated.
  - **TITAN_COORDINATE_ROTATE:** Coordinate will no longer be rotated.
  - **TITAN_COORDINATE_SCALE:** Coordinate will no longer be scaled.
- **coordinate:** A pointer to a data structure of type *titan_coordinate_t*, where:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.

- **angle**: Destination angle of the simulation object relative to its parent, where:
  - **pitch** is the destination pitch of the simulation object in radians.
  - **yaw** is the destination yaw of the simulation object in radians.
  - **roll** is the destination roll of the simulation object in radians.
  - **T** is the future time (in seconds) when the simulation object will reach the destination angle.
- **scale**: Destination absolute scale of the simulation object, where:
  - **X** is the destination width of the simulation object.
  - **Y** is the destination height of the simulation object.
  - **Z** is the destination depth of the simulation object.
  - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
- **prev**: Previous coordinate index in an array of coordinates (animation).
- **next**: Next coordinate index in an array of coordinates (animation).
- **trigger**: A trigger data structure, where:
  - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
  - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
  - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
- **update_trail**: Update the coordinate trail when true; do not update the trail otherwise. Coordinate trail coordinates are accessed by methods 4, 5, 6, 7, and 8 of the location interface.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully updated.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE EARLIEST ATTRIBUTE COORDINATE (4):

```
titan_result_t GetEarliestCoordinate(TITAN_HANDLE64
object_handle, TITAN_ULONG &current_index,
titan_coordinate_t *out_coordinat, TITAN_HANDLE64
simulation_handshake = 0)
```

The location attribute trail holds all previous coordinates of an instance. These past coordinates vanish over time as new coordinates are added to trail. Method **Location-4** is used to retrieve the earliest coordinate in a location trail of a particular simulation object.

Where:

- **object_handle**: Handle of the simulation object whose location attribute earliest trail entry will be retrieved.
- **current_index**: Returned index of the earliest entry in the trail.
- **out_coordinate:** A pointer to a data structure where return coordinate will be copied, such that:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **T** is the future time (in seconds) when the simulation object will reach the destination angle.
  - **scale**: Destination absolute scale of the simulation object, where:

- **X** is the destination width of the simulation object.
- **Y** is the destination height of the simulation object.
- **Z** is the destination depth of the simulation object.
- **T** is the future time (in seconds) when the simulation object will reach the destination scale.

- **prev**: Previous coordinate index in an array of coordinates (animation).
- **next**: Next coordinate index in an array of coordinates (animation).
- **trigger**: A trigger data structure, where:
  - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
  - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
  - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
  - **simulation_handshake**: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE LATEST LOCATION ATTRIBUTE COORDINATE (5):

```
titan_result_t GetLatestCoordinate(TITAN_HANDLE64
object_handle, TITAN_ULONG &current_index,
titan_coordinate_t *out_coordinate, TITAN_HANDLE64
simulation_handshake = 0)
```

The location attribute trail holds all previous coordinates of an instance. These past coordinates vanish over time as new coordinates are added to the trail. Method **Location-5** is used to retrieve the latest or most recent coordinate in a location trail of a particular simulation object.

Where:

- **object_handle**: Handle of the simulation object whose location attribute latest trail entry will be retrieved.
- **current_index**: Returned index of the latest entry in the trail.
- **out_coordinate:** A pointer to a data structure where return coordinate will be copied, such that:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **T** is the future time (in seconds) when the simulation object will reach the destination angle.

- **scale**: Destination absolute scale of the simulation object, where:
  - **X** is the destination width of the simulation object.
  - **Y** is the destination height of the simulation object.
  - **Z** is the destination depth of the simulation object.
  - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
- **prev**: Previous coordinate index in an array of coordinates (animation).
- **next**: Next coordinate index in an array of coordinates (animation).
- **trigger**: A trigger data structure, where:
  - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
  - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
  - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE PREVIOUS LOCATION ATTRIBUTE COORDINATE (6):

```
titan_result_t GetPreviousCoordinate(TITAN_HANDLE64
object_handle, TITAN_ULONG &current_index,
titan_coordinate_t *out_coordinate, TITAN_HANDLE64
simulation_handshake = 0)
```

Method **Location-6** is used to retrieve the previous coordinate in a location trail based on a specified trail entry index.

Where:

- **object_handle**: Handle of the simulation object whose location attribute trail entry will be retrieved.
- **current_index**: Index of the trail coordinate whose predecessor entry will be retrieved.
- **out_coordinate:** A pointer to a data structure where return coordinate will be copied, such that:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **T** is the future time (in seconds) when the simulation object will reach the destination angle.
  - **scale**: Destination absolute scale of the simulation object, where:

    - **X** is the destination width of the simulation object.
    - **Y** is the destination height of the simulation object.
    - **Z** is the destination depth of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
  - **prev**: Previous coordinate index in an array of coordinates (animation).
  - **next**: Next coordinate index in an array of coordinates (animation).
  - **trigger**: A trigger data structure, where:
    - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
    - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
    - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE LOCATION NEXT ATTRIBUTE COORDINATE (7):

```
titan_result_t GetNextCoordinate(TITAN_HANDLE64
object_handle, TITAN_ULONG &current_index,
titan_coordinate_t *out_coordinate, TITAN_HANDLE64
simulation_handshake = 0)
```

Method **Location-7** is used to retrieve the next coordinate in a location trail based on a specified trail entry index.

Where:

- **object_handle**: Handle of the simulation object whose location attribute trail entry will be retrieved.
- **current_index**: Index of the trail coordinate whose successor entry will be retrieved.
- **out_coordinate:** A pointer to a data structure where return coordinate will be copied, such that:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **T** is the future time (in seconds) when the simulation object will reach the destination angle.
  - **scale**: Destination absolute scale of the simulation object, where:

    - **X** is the destination width of the simulation object.
    - **Y** is the destination height of the simulation object.
    - **Z** is the destination depth of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
  - **prev**: Previous coordinate index in an array of coordinates (animation).
  - **next**: Next coordinate index in an array of coordinates (animation).
  - **trigger**: A trigger data structure, where:
    - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
    - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
    - **Trigger**: Registering trigger method that will be invoked when the destination is reached.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RETRIEVE A LIST OF LOCATION ATTRIBUTE COORDINATES (8):

```
titan_result_t GetCoordinates(TITAN_HANDLE64
object_handle, TITAN_ULONG &current_index,
titan_coordinate_t *out_coordinates, TITAN_UINT
&num_out_coordinates, TITAN_HANDLE64
simulation_handshake = 0)
```

Method Location-8 is used to retrieve a list of coordinates in a location trail based on a specified trail entry index.

Where:

- **object_handle**: Handle of the simulation object whose location attribute trail entries will be retrieved.
- **current_index**: Index of the trail coordinate entry of the last coordinate of the list to be retrieved. Set to 0 to retrieve the latest coordinates,
- **out_coordinates:** A pointer to an array where the coordinates will be copied, such that:
  - **udpate_flags**: Bitmap of one or multiple update flags, where:
    - **TITAN_COORDINATE_TRANSLATE:** Coordinate will be translated.
    - **TITAN_COORDINATE_ROTATE:** Coordinate will be rotated.
    - **TITAN_COORDINATE_SCALE:** Coordinate will be scaled.
  - **trigger_flags**:
    - **TITAN_COORDINATE_TRANSLATE:** Trigger method when translation destination is reached.
    - **TITAN_COORDINATE_ROTATE:** Trigger method when rotation destination is reached.
    - **TITAN_COORDINATE_SCALE:** Trigger method when scale destination is reached.
  - **position**: Destination Position of the simulation object relative to its parent, where:
    - **X** is the destination x-coordinate of the simulation object.
    - **Y** is the destination y-coordinate of the simulation object.
    - **Z** is the destination z-coordinate of the simulation object.
    - **T** is the future time (in seconds) when the simulation object will reach the destination position.
  - **angle**: Destination angle of the simulation object relative to its parent, where:
    - **pitch** is the destination pitch of the simulation object in radians.
    - **yaw** is the destination yaw of the simulation object in radians.
    - **roll** is the destination roll of the simulation object in radians.
    - **T** is the future time (in seconds) when the simulation object will reach the destination angle.

- **scale**: Destination absolute scale of the simulation object, where:
  - **X** is the destination width of the simulation object.
  - **Y** is the destination height of the simulation object.
  - **Z** is the destination depth of the simulation object.
  - **T** is the future time (in seconds) when the simulation object will reach the destination scale.
- **prev**: Previous coordinate index in an array of coordinates (animation).
- **next**: Next coordinate index in an array of coordinates (animation).
- **trigger**: A trigger data structure, where:
  - **code_data**: Primary supplied handle to be sent to trigger method upon reaching destination.
  - **axiom_data**: Secondary supplied handle to be sent to trigger method upon reaching destination.
  - **Trigger**: Registering trigger method that will be invoked when the destination is reached.

- **num_out_coordinates**: Number of coordinates in the trail preceding *current_index* that will be retrieved. If *current_index* is 0, then this is the number of the latest coordinates in the trail.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully retrieved.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

PAUSE SIMULATION OF A LOCATION ATTRIBUTE COORDINATE (9):

```
titan_result_t Pause(TITAN_HANDLE64 object_handle,
TITAN_HANDLE64 simulation_handshake = 0)
```

Method **Location-9** is used to pause simulation of the location attribute of the supplied simulation object:

Where:

- **object_handle**: Handle of the simulation object being paused.
- **simulation_handshake**: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully paused.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

RESUME SIMULATION OF A LOCATION ATTRIBUTE COORDINATE (A):

```
titan_result_t Resume(TITAN_HANDLE64 object_handle,
TITAN_HANDLE64 simulation_handshake = 0)
```

Method **Location-A** is used to resume simulation of the location attribute of the supplied simulation object.

Where:

- **object_handle**: Handle of the simulation object being resumed.
- ***simulation_handshake***: Handshake to be processed by the simulation hosting the simulation object.

RETURN:

- **TITAN_RESULT_OK:** The coordinate has been successfully resumed.
- **TITAN_RESULT_*:** See Atlas Errors for more information.

Size Attribute Interface:

NOTE: This interface is currently disabled.

Information:

The size attribute represents information derived from, or specified in, a simulation object.

This attribute is not to be confused with the scale component of the location attribute (See *Location Attribute*). The latter is used for absolute measurements (i.e. scalar values), whereas the size attribute is used for comparative measurements (e.g. big, huge, tiny, etc.).

Annotation:

Soon to be available

Interface:

Soon to be available

GET SIZE ATTRIBUTE VERSION (1):

```
TITAN_FLOAT GetVersion()
```

Method **Size-1** is used to retrieve the size interface version of current domain simulation.

Where:

RETURN:

- Version of the size attribute interface.

Shape Attribute Interface:

NOTE: This interface is currently disabled.

Information:

The shape attribute manages aspects of a geometric representation of a simulation object.

This representation comes in the form of a triangle mesh centered on the location attribute origin.

Annotation:

Soon to be available

Interface:

Soon to be available

GET SHAPE ATTRIBUTE VERSION (1):

```
TITAN_FLOAT GetVersion()
```

Method **Shape-1** is used to retrieve the shape interface version of current domain simulation.

Where:

RETURN:

- Version of the shape attribute interface.

Texture Attribute Interface:

NOTE: This interface is currently disabled.

Information:

The texture attribute manages aspects of a texture representation of a simulation object.

This representation comes in the form of a vector field centered on the location attribute origin.

Annotation:

Available soon

Interface:

Available soon

GET TEXTURE ATTRIBUTE VERSION (1):

```
TITAN_FLOAT GetVersion()
```

Method **Texture-1** is used to retrieve the texture interface version of current domain simulation.

Where:

RETURN:

- Version of the texture attribute interface.

Color Attribute Interface:

Information:

The color attribute manages aspects of color representation of a simulation object.

This representation comes in the form of a *TITAN_FLOAT* RGBA quad.

Interface:

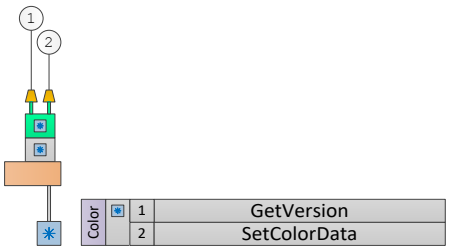


Fig 32. Color attribute interface diagram and table.

GET COLOR ATTRIBUTE VERSION (1):

```
TITAN_FLOAT GetVersion()
```

Method **Color-1** is used to retrieve the color interface version of current domain simulation.

Where:

RETURN:

- Version of the color attribute interface.

SET ATTRIBUTE COLOR DATA (2):

```
titan_result_t SetColorData(TITAN_HANDLE64
object_handle, TITAN_FLOAT *color)
```

Method **Color-2** is used to set the memory pointer of a color attribute of a simulation object, give a simulation object handle.

Where:

- **object_handle**: Handle of the simulation object whose color attribute data will be set.
- **color:** User-supplied pointer to the TITAN_FLOAT RGBA quad in memory that will be associated with the specified object handle.

RETURN:

- **TITAN_RESULT_OK**: Object color data was successfully set.
- **TITAN_RESULT_*:** See Atlas Errors for more information

APPENDIX A

[Insert Detailed Overview]

Fig A1. Domain overview.