AvesTerra™

A Framework for Global-Scale Knowledge Orchestration

Hypergraph Transfer Protocol (HGTP)

Version 3.5

April 2023

GEORGETOWN UNIVERSITY

V3.5 – April 2023

Table of Contents

Preface

This document is the second in a series of companion documents that collectively describe the AvesTerra system for global-scale knowledge orchestration and analytic interoperability. The list of key AvesTerra documents include:

- AvesTerra: FOUR-Color Framework

  The *FOUR-Color Framework* document provides an overview of the original principles and architectural aspects of AvesTerra's approach to knowledge sharing and analytic interoperability, including a description of the end-to-end system structure and each of the main four layers of the design. AvesTerra is the first major effort to make use of the full spectrum of FOUR-Color architectural principles.

- AvesTerra: Hypergraph Transaction Protocol (HGTP)

  The *HGTP* document (presented here) provides a specification of the communication protocol used for all AvesTerra interactions including between AvesTerra clients and AvesTerra servers, and between AvesTerra servers with their peers.

- AvesTerra: Application Programming Interface (API)

  The *API* document describes the application programming interface constructs of the AvesTerra system. This interface is intended for systems level developers needing to interact with AvesTerra servers at the core level.

- AvesTerra: Integration and Application Library (Avial)

  The *Avial* document provides the definition and technical details of the layer of standardized integration and application services built directly atop the AvesTerra API. This document is intended for developers integrating new data sources, adapters, analytics, tools, and application components into the AvesTerra ecosystem.

Acknowledgement

This paper is a result of lengthy technical conversations with an extraordinary community of researchers, analysts, and computational science talent spanning the Nation over the past two decades. Special thanks to the original contributors including Lawrence Livermore National Laboratory, Oak Ridge National Laboratory, Pacific Northwest National Laboratory, and the Raytheon Company for their intellectual contributions to this effort, and to the Defense Advanced Research Projects Agency (DARPA) for their sponsorship of this work under the SIMPLEX program via contract #N66001-15-C-4044. The views contained within are solely those of the author and do not necessarily reflect the opinions or positions of Georgetown University or any collaborator or sponsor organization involved.

For additional information, contact:

J. C. Smart, Ph.D.
AvesTerra Chief Scientist
Research Professor
Department of Computer Science
Office of the Senior Vice President for Research
Georgetown University
37th and O Street, NW
Washington, D.C. 20057-1241
smart@georgetown.edu
(443) 745-5876

Summary of Changes

Version 1.0 – Initial Release. [June 2016]

Version 2.0 – Conversion to 1280-byte binary message frame

Version 3.0 – Conversion to TLS encrypted 1024-byte message frame

Version 3.1 – Fields renamed (**class**, **category**, **context**, **resultant**)

Version 3.2 – **instance** field reduced to 32 bits; 32-bit **offset** field added.

Version 3.3 – **value** field renamed **bytes**

Version 3.4 – Added SHUTDOWN_ERROR code in support of remote server/adapter termination

Version 3.5 – Added TUNNEL_COMMAND code in support of HGTP tunneling. BYE_COMMAND code change (no longer defaults to 0). NULL_COMMAND code added (i.e. 0). ADDRESS_REPORT code added (i.e. 40)

Introduction

This document provides a specification for the AvesTerra Hypergraph Transaction Protocol (HGTP). The AvesTerra Application Programming Interface (API) services (described in a companion document) are layered upon this protocol. In AvesTerra (beginning with Version 2), HGTP is used for all communication between AvesTerra clients and AvesTerra servers, and between AvesTerra servers and other peer AvesTerra servers. Client applications typically need not interact directly with AvesTerra servers at the HGTP level. Instead, clients communicate using an AvesTerra API programming language binding which performs the appropriate HGTP transactions on their behalf.

AvesTerra uses a peer-to-peer server style infrastructure to create a distributed knowledge representation space. Client applications need only communicate with one of the servers in this infrastructure (e.g. a server running on the local computer or perhaps elsewhere on the local area network). When a client makes an entity request (e.g. via an API call), the server that receives the request determines if it is the server that manages the specified entity. This is performed by examining the entity's unique identifier (EUID) which internally encodes the network and host addresses of the entity's server of record. If the entity is managed by the receiving server, then this server fulfills the request, sending back an appropriate response to the client. However, if the entity is managed elsewhere, then the server forwards the request to a peer AvesTerra server, as determined by its routing algorithm, to help the request reach the necessary final destination server. When the server forwards this request to the peer server, it again uses HGTP to communicate. However, the server then behaves as a client to the peer server.

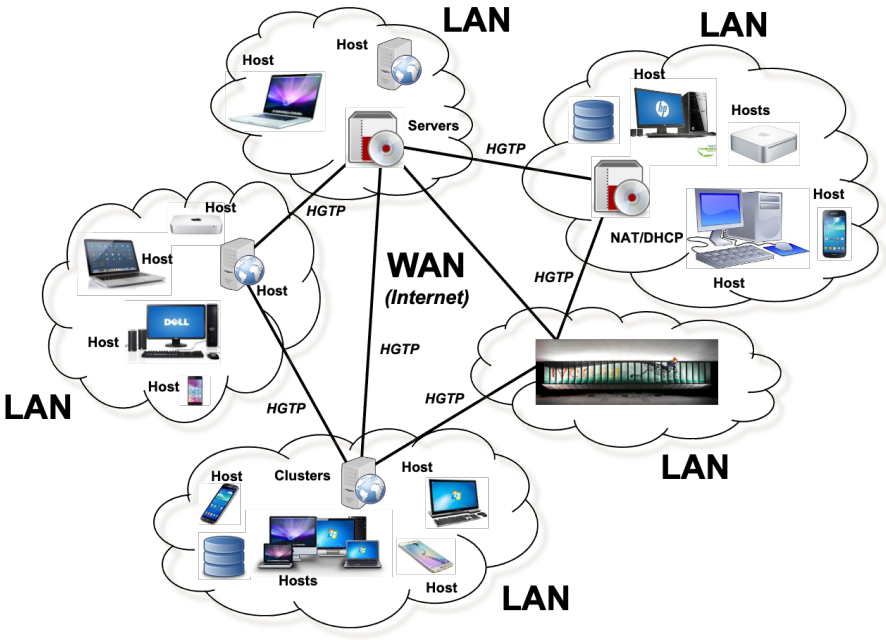
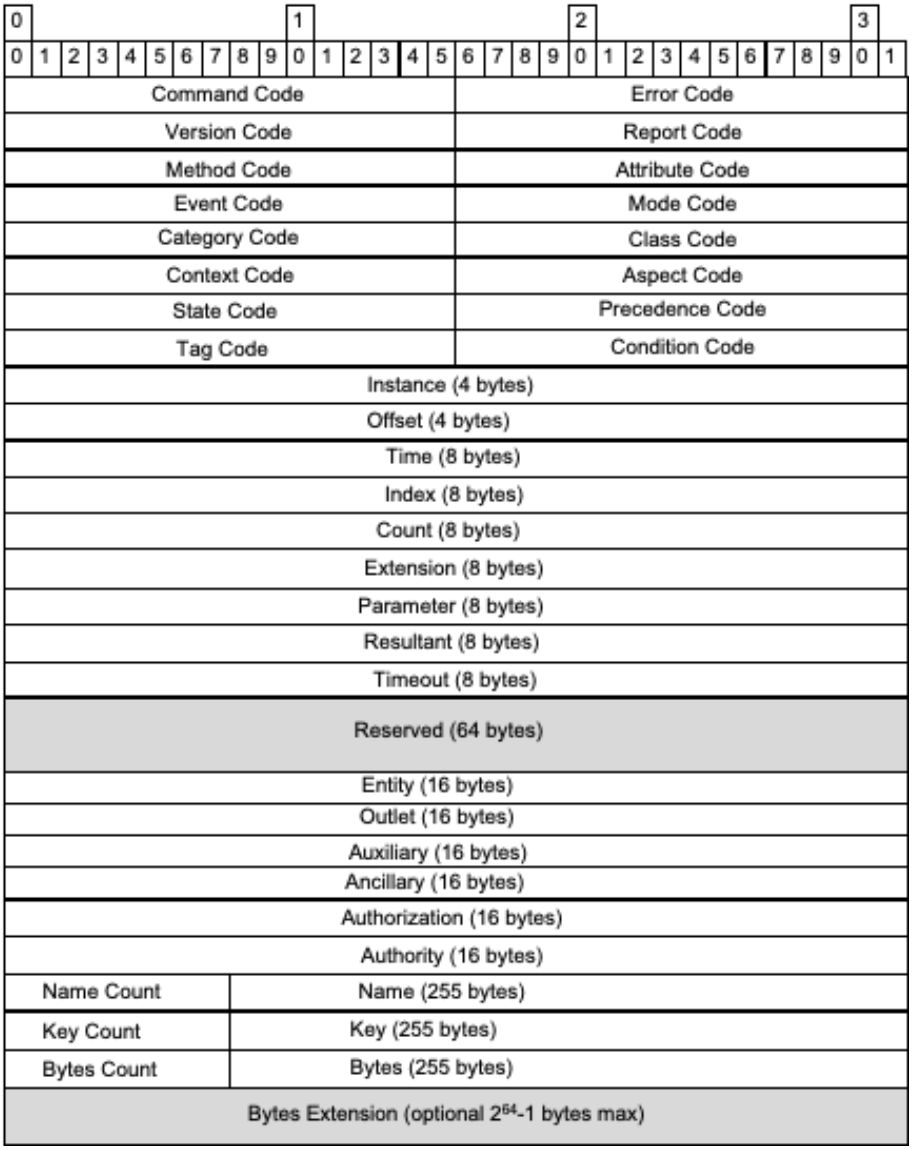


Figure 1. AvesTerra Peer-to-Peer Architecture

Protocol Overview

HGTP (Version 3 series) uses a fixed-length standard binary message frame with an optional variable-length extension block for all Internet communications (Figure 2). All messages are sent using the Transmission Control Protocol (TCP) with Transport Layer Security (TLS) encryption. All socket communication is through TCP port 20057 (IANA approved). Character encodings within message frames conform to the American Standard Code for Information Interchange (US-ASCII), or the more general Unicode Transformation Format with 8-bit variable width encoding (UTF-8). Numeric values are transmitted in network byte order (i.e. "Big-Endian").



0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	
Command Code	Error Code
Version Code	Report Code
Method Code	Attribute Code
Event Code	Mode Code
Category Code	Class Code
Context Code	Aspect Code
State Code	Precedence Code
Tag Code	Condition Code
Instance (4 bytes)	
Offset (4 bytes)	
Time (8 bytes)	
Index (8 bytes)	
Count (8 bytes)	
Extension (8 bytes)	
Parameter (8 bytes)	
Resultant (8 bytes)	
Timeout (8 bytes)	
Reserved (64 bytes)	
Entity (16 bytes)	
Outlet (16 bytes)	
Auxiliary (16 bytes)	
Ancillary (16 bytes)	
Authorization (16 bytes)	
Authority (16 bytes)	
Name Count	Name (255 bytes)
Key Count	Key (255 bytes)
Bytes Count	Bytes (255 bytes)
Bytes Extension (optional 2^{64}-1 bytes max)	

Figure 2. Standard HGTP Message Frame

As described in the API, the primary objects manipulated via this protocol are AvesTerra "entities", the fundamental construct of an AvesTerra knowledge space. These entities are representations of knowledge items governed either implicitly or explicitly an ontology. Ontology related notions are outside of the scope of this protocol. That is, all ontology references occur at the Avial or higher application level in order to preserve HGTP's ontological neutrality.

HGTP typically follows a simple command/response message model. A client application makes a request by sending a message to a server. The server performs the operation, if possible, sending a return message with an acknowledgement and any result. If the server is unable to perform the operation, an error message is returned instead. A single request made to a server, however, may initiate other messages to potentially other servers and clients. The **INVOKE** and **INQUIRE** request are two key operations where this frequently occurs. Invoking an entity method, or inquiring an entity attribute results in the server forwarding a **CALL** request to the appropriate adapter that is connected or attached, respectively. Similarly, publication of an entity event results in a **NOTIFY** request being forwarded to the appropriate subscriber(s).

The first field of every message transmitted by a client is the HGTP **command** code. This code indicates the specific type of transaction being requested and determines how fields present in the remainder of the message are to be interpreted. The 8-byte **extension** field is used to specify any additional data bytes transmitted after the fixed 1024-byte block. The extension bytes are typically used when transmitting values that exceed the limits of the 255-byte **value** field.

In total, HGTP contains 59 different message types, in close alignment with the API specification. A typical message exchange involves a client opening a socket connection to an AvesTerra server listening on port 20057. The client sends the properly formatted message frame, any optional extension bytes, and then awaits a return response. Success is indicated by a null **error** field (i.e. 0) in the returned message. When the error code is not null, an optional error message may be passed via the **value** field.

Implementation Notes

To minimize the burden on the host operating system's networking subsystem, an AvesTerra API implementation may choose to leave the socket open for subsequent reuse in the event of another client request to the same server. A timer (e.g. 1-2 minutes) is generally affixed to this socket to ensure eventual closing of any dormant connections. Upon expiration of this timer, the **BYE** message is sent by a client to notify the server of an orderly socket shutdown. A client and/or a server may each also choose to unilaterally shutdown a connection in the event of application termination or an unrecoverable error condition (e.g. host shutdown, network disruption, etc). The definition of each HGTP message field is described in the following section.

Field Specifications

Each of the message fields identified above in the HGTP message frame (figure 2) are specified in this section. Message fields include message parameters, message codes, report codes, response codes, errors, and termination code.

Message Parameters

The following message parameters are defined:

(command)	16-bit command code, enumerated below
(error)	16-bit error code, enumerated below
(version)	16-bit version code, for version management/compatibilit
(report)	16-bit report code, enumerated below
(method)	16-bit method code, enumerated by Method Taxonomy
(attribute)	16-bit attribute code, enumerated by Attribute Taxonomy
(event)	16-bit event code, enumerated by Event Taxonomy
(mode)	16-bit mode code, enumerated by Mode Taxonomy
(category)	16-bit category code, enumerated by Category Taxonomy
(class)	16-bit class code, enumerated by Class Taxonomy
(context)	16-bit context code, enumerated by Context Taxonomy
(aspect)	16-bit aspect code, enumerated by Aspect Taxonomy
(state)	16-bit state code, enumerated by State Taxonomy
(precedence)	16 bit unsigned integer determining INVOKE/INQUIRE precedence
(tag)	16-bit tag code, enumerated by Tag Taxonomy
(condition)	16-bit condition, enumerated by Condition Taxonomy
(instance)	32-bit unsigned integer for attribute instance support
(offset)	32-bit unsigned integer for index offsetting
(time)	64-bit time (Unix EPOCH time)
(index)	64-bit unsigned integer for aspect indexing
(count)	64-bit unsigned integer for aspect counting
(extension)	64-bit unsigned extension byte count
(parameter)	64-bit integer for parameter passing
(resultant)	64-bit integer for resulting return
(timeout)	64-bit unsigned integer timeout (typically in seconds)
<entity>	AvesTerra entity identifier of the form “<NID|HID|UID>” where PID is a 32-bit unsigned integer for the public (wide-area) network identifier; HID is a 32-bit unsigned integer for the private (local area) host identifier; UID is a 64-bit unsigned integer for entity’s unique identifier.
<outlet>	Outlet entity identifier
<auxiliary>	Auxiliary entity identifier
<ancillary>	Ancillary entity identifier
{authorization}	128-bit authorization code
{authority}	128-bit authority code
(name count)	8-bit name field byte count
“name”	255 8-bit byte name field
(key count)	8-bit key field byte count

"key"	255 8-bit byte name field
(bytes count)	8-bit bytes field byte count
(bytes)	255 8-bit byte bytes field.
[bytes extension]	Optional extension bytes; used if bytes field is not sufficient for value transmission

Command Codes

Each client HGTP message begins with an unsigned integer command code. The following command codes are currently defined:

Command Code	Code Value
(NULL_COMMAND)	0
(create_COMMAND)	1
(delete_COMMAND)	2
(connect_COMMAND)	3
(disconnect_COMMAND)	4
(attach_COMMAND)	5
(detach_COMMAND)	6
(invoke_COMMAND)	7
(inquire_COMMAND)	8
(call_COMMAND)	9
(adapt_COMMAND)	10
(reset_COMMAND)	11
(connection_COMMAND)	12
(attachment_COMMAND)	13
(reference_COMMAND)	14
(dereference_COMMAND)	15
(activate_COMMAND)	16
(deactivate_COMMAND)	17
(publish_COMMAND)	18
(subscribe_COMMAND)	19
(notify_COMMAND)	20
(cancel_COMMAND)	21
(subscription_COMMAND)	22
(wait_COMMAND)	23
(flush_COMMAND)	24
(attached_COMMAND)	25
(connected_COMMAND)	26
(subscribed_COMMAND)	27
(authorize_COMMAND)	28
(deauthorize_COMMAND)	29
(authorized_COMMAND)	30
(authorization_COMMAND)	31

(report_COMMAND)	32
(lock_COMMAND)	33
(unlock_COMMAND)	34
(change_COMMAND)	35
(insert_COMMAND)	36
(remove_COMMAND)	37
(replace_COMMAND)	38
(erase_COMMAND)	39
(find_COMMAND)	40
(element_COMMAND)	41
(length_COMMAND)	42
(arm_COMMAND)	43
(disarm_COMMAND)	44
(start_COMMAND)	45
(stop_COMMAND)	46
(reset_COMMAND)	47
(fetch_COMMAND)	48
(set_COMMAND)	49
(clear_COMMAND)	50
(test_COMMAND)	51
(execute_COMMAND)	52
(halt_COMMAND)	53
(schedule_COMMAND)	54
(redirect_COMMAND)	55
(condition_COMMAND)	56
(tunnel_COMMAND)	57
(bye_COMMAND)	58

Report Codes

Each client HGTP **REPORT** message includes a report code that identifies the specific report being requested. The following report codes are currently defined:

Report Code	Code Value
(NULL_REPORT)	0
(AVAILABLE_REPORT)	1
(NAME_REPORT)	2
(CATEGORY_REPORT)	3
(CLASS_REPORT)	4
(ACTIVATED_REPORT)	5
(CONNETIONS_REPORT)	6
(ATTACHMENTS_REPORT)	7
(SUBSCRIPTIONS_REPORT)	8
(AUTHORITIES_REPORT_)	9

(REDIRECTION_REPORT)	10
(REFERENCES_REPORT)	11
(TIMESTAMP_REPORT)	12
(SERVER_REPORT)	13
(LOCAL_REPORT)	14
(GATEWAY_REPORT)	15
(HOSTNAME_REPORT)	16
(VERSION_REPORT)	17
(EXTINCT_REPORT)	18
(PENDING_REPORT)	19
(LOCKED_REPORT)	20
(RENDEZVOUS_REPORT)	21
(STATUS_REPORT)	22
(CONTEXT_REPORT)	23
(CLOCK_REPORT)	24
(TIMER_REPORT)	25
(ELAPSED_REPORT)	26
(ARMED_REPORT)	27
(ACTIVE_REPORT)	28
(STATE_REPORT)	29
(CONDITIONS_REPORT)	30
(ELEMENTS_REPORT)	31
(KEY_REPORT)	32
(BUSY_REPORT)	33
(BLOCKING_REPORT)	34
(WAITING_REPORT)	35
(ADAPTING_REPORT)	36
(INVOKING_REPORT)	37
(INTERNET_REPORT)	38
(AUTHORITY)_REPORT)	39
(ADDRESS_REPORT)	40

Error Codes

A response message is returned for every HGTP message request. Each response message contains an error code. The following error codes are currently defined:

Response Code	Code Value
(NULL_ERROR)	0
(AVESTERRA_ERROR)	1
(ENTITY_ERROR)	2
(OUTLET_ERROR)	3
(NETWORK_ERROR)	4
(TIMEOUT_ERROR)	5

(AUTHORIZATION_ERROR)	6
(ADPATER_ERROR)	7
(SUBSCRIBER_ERROR)	8
(APPLICATION_ERROR)	9
(BYPASS_ERROR)	10
(FORWARD_ERROR)	11
(VALUE_ERROR)	12
(MESSAGE_ERROR)	13
(EXECUTION_ERROR)	14
(SHUTDOWN_ERROR)	15

With each non-null error code, an optional error message string can be passed via the **bytes** and **byte count** fields.

HGTP Development

The development of HGTP has been an ongoing research and development activity at Georgetown University, designed and implemented with support from DARPA under contract for the SIMPLEX and AIDA programs. The HGTP protocol is currently in widespread operation throughout all AvesTerra system deployments including its ATra configurations. The protocol was approved for Internet use and is listed in the IANA Service Name and Transport Protocol Port Number Registry as:

Service Name:	avesterra
Port Number:	20057
Transport protocol	tcp
Description:	AvesTerra Hypergraph Transfer Protocol (HGTP)
Assignee:	Georgetown University
Contact:	J. C. Smart
Registration date:	2016-06-06

HGTP V1.0 marked the completion of the protocol's first major version, providing support for all of AvesTerra's original API transactions. The HGTP V1 series used a simple text-based message exchange and was extended numerous times to accommodate enhancements to the AvesTerra API. HGTP V2.0 replaced the cumbersome and problematic text-based format with a fixed binary message frame. This enabled a dramatic simplification to the API with improved performance and reliability. In addition, it enabled transactions to potentially be examined at line rate to filter across select isolation boundaries based on a mask of the **command**, **category**, **class**, **context**, **method**, **attribute**, **condition**, **state**, **aspect**, and **error** fields. The ability to design and verify such isolators using formal method techniques has been demonstrated via a Georgetown research collaborator. HGTP V3.0 incorporated full TLS encryption, with subsequent revisions made to incorporate numerous server operations and upper-level Avial functions such as remote systems management and tunneling.