JSC XXXXX-DD.mm.20

# Trick High Level Architecture
# TrickHLA
# Verification and Validation

Simulation and Graphics Branch (ER7)
Software, Robotics and Simulation Division
Engineering Directorate

## Package Release TrickHLA v3.0.0 - Beta

## Document Revision 1.0
## June 2020

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC XXXXX-DD.mm.20

# Trick High Level Architecture
# TrickHLA
# Verification and Validation

Document Revision 1.0
June 2020

Edwin Z. Crues
and
Daniel E. Dexter

Simulation and Graphics Branch (ER7)
Software, Robotics and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

## Abstract

TrickHLA is a middleware model package that provides an interface framework for enabling IEEE-1516 High Level Architecture (HLA) capabilities in simulations developed in the Trick Simulation Environment. TrickHLA allows a developer to concentrate on simulation development without needing to be an HLA expert. The TrickHLA model is data driven and provides a simplified API making it relatively easy to take an existing Trick-based simulation and make it HLA capable.

# Contents

# Chapter 1

# Introduction

The objective of TrickHLA is to simplify the process of providing simulations built with the Trick Simulation Environment[8] with the ability to participate in distributed executions using the High Level Architecture (HLA)[12]. This allows a simulation developer to concentrate on the simulation and not have to be an HLA expert. TrickHLA is data driven and provides a simple API making it relatively easy to take an existing Trick simulation and make it HLA capable.

## 1.1 Identification of Document

This document describes the TrickHLA model developed for use in the Trick Simulation Environment. This document adheres to the documentation standards defined in NASA Software Engineering Requirements Standard [7].

## 1.2 Scope of Document

This document provides information on the requirements for TrickHLA.

## 1.3 Purpose and Objectives of Document

The purpose of this document is to define the set of requirements that the TrickHLA must achieve to be compatible with Federate Inferface Specification of the IEEE Standard for Modeling and Smulation (M&S) High Level Architecture (HLA) [11].

## 1.4 Documentation Status and Schedule

The information in this document is current with the TrickHLA v3.0.0 - Beta implementation of the TrickHLA. Updates will be kept current with module changes.

| Author | Date | Description |
|---|---|---|
| Edwin Z. Crues | June 2020 | TrickHLA Version 3 |

| Revised by | Date | Description |
|---|---|---|

## 1.5 Document Organization

This document is organized into the following sections:

**Chapter 1: Introduction** - Identifies this document, defines the scope and purpose, present status, and provides a description of each major section.

**Chapter 2: Related Documentation** - Lists the related documentation that is applicable to this project.

**Chapter 3: Verification** - Presents the results of `TrickHLA` requirements verification.

**Bibliography** - Informational references associated with this document.

## Chapter 2

# Related Documentation

## 2.1 Parent Documents

The following documents are parent to this document:

- *Trick High Level Architecture (*`TrickHLA`*)* [1]

## 2.2 Applicable Documents

The following top level documents are applicable to this document:

- `TrickHLA` *Product Specification* [3]
- `TrickHLA` *User Guide* [4]
- `TrickHLA` *Product Requirements* [2]
- *Distributed Space Exploration Simulation Multiphase Initialization Design* [5]
- *Integrated Mission Simulation Multiphase Initialization Design* [6]

The following specific documents are applicable to this document:

- *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Inferface Specification* [11]
- *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification* [13]

The following additional documents are applicable to this document:

- *Trick Simulation Environment: Installation Guide* [9]

- *Trick Simulation Environment: Tutorial* [10]
- *Trick Simulation Environment: Documentation* [8]
- *NASA Software Engineering Requirements* [7]

# Chapter 3

# Verification

This chapter summarizes the verification activities carried out for each of the `TrickHLA` requirements.

## 3.1 General Requirements

### 3.1.1 TrickHLA_1: Documentation

**Summary.** The model must include requirements specifications, software/interface/version information, a users guide, and documentation of test procedures and results.

**Method.** Inspection.

**Results.** `TrickHLA` requirements are documented in the `TrickHLA` *Product Requirements*. The software, interfaces and current version information is documented in the `TrickHLA` *Product Specification*. An introduction for users is available in the `TrickHLA` *User Guide*. Test procedures and results are documented in this document.

### 3.1.2 TrickHLA_2: Header File Trick Header

**Summary.** `TrickHLA` header files must include a Trick header that specifies the purpose of the file, references, assumptions/limitations and the author.

**Method.** Inspection

**Results.** The `TrickHLA` header files (`.hh` files) are in the `include/TrickHLA/` directory. All of them include a Trick file header with the necessary information.

### 3.1.3 TrickHLA_3: Trick Comments for Enumerated Types

**Summary.** The enumerated values of `enum` types must be accompanied by a comment explaining each one.

**Method.** Inspection.

**Results.** The `TrickHLA` enumerated types are defined in `include/TrickHLA/Types.hh`. They are all commented with explanation of what each one means.

### 3.1.4 TrickHLA_4: Trick Comments for Data Structures

**Summary.** Each data structure must have a Trick-compliant explaining its purpose.

**Method.** Inspection.

**Results.** The `TrickHLA` data structures are C++ classes. The fields of these classes are defined in the header files (`.hh` files) in the `include/TrickHLA/` directory. They all have Trick comments.

### 3.1.5 TrickHLA_5: Source File Trick Headers

**Summary.** `TrickHLA` source code files (`.c and .cpp` files) must include a Trick header that specifies the purpose of the file, references, assumptions/limitations, Trick job class, library dependencies and the author.

**Method.** Inspection.

**Results.** The `TrickHLA` source code is C++ in `.cpp` files in the `source/TrickHLA/` directory. All of the files inlcude a Trick file header with the necessary information.

### 3.1.6 TrickHLA_6: Trick Comments for Function Definitions

**Summary.** Each function must be documented with Trick-compliant comments that explain the arguments and return value.

**Method.** Inspection

**Results.** All the functions defined in the C++ source files (`.cpp` files) in the `source/TrickHLA/` directory have Trick-compliant comments describing the arguments and return values.

### 3.1.7 TrickHLA_7: HLA Federate Interface

**Summary.** `TrickHLA` must be based on the IEEE 1516.1-2010 service definitions.

**Method.** Inspection

**Results.** `TrickHLA` is built on top of the Pitch HLA system, which is compliant with IEEE 1516.

## 3.2 Data Requirements

### 3.2.1 TrickHLA_8: Primitive Data Types

**Summary.** `TrickHLA` must support a wide variety of C/C++ primitive data types (e.g, int, long int, float, double, etc...).

**Method.** Inspection.

**Results.** HLA saves data in so-called *object attributes* and *interaction parameters*. Inspection of the `TrickHLA` source files, `source/TrickHLA/Attribute.cpp` and `source/TrickHLA/Parameter.cpp` reveals support for all the required primitive types.

### 3.2.2 TrickHLA_9: Static Arrays of Primitive Data Types

**Summary.** `TrickHLA` must support arrays of the supported primitive types.

**Method.** Inspection

**Results.** The `TrickHLA::Attribute` and `TrickHLA::Parameter` classes both have a method, `get_number_of_items()` which returns the number of items in an attribute and/or parameters array. The only type no supported by the code is an HLA logical time. All the other supported primitive types may occur in attribute or parameter arrays.

## 3.3 Functional Requirements

### 3.3.1 TrickHLA_10: Data Driven

**Summary.** `TrickHLA` must be data driven (i.e., parameterized by values specified in input files).

**Method.** Inspection

**Results.** Like most Trick models, using TrickHLA requires a balance of jobs defined in S_define files and initialization data specified in input files. TrickHLA is aggressively parameterized, allowing many parameters to be specified in the input files. The TrickHLA *User Guide* includes a detailed description of these input files for each of the various TrickHLA capabilities (e.g., ownership transfer, interactions, publish/subscribe, etc...).

### 3.3.2 TrickHLA_11: HLA Big and Little Endian

**Summary.** TrickHLA must support big- and little-endian byte ordering.

**Method.** Inspection.

**Results.** TrickHLA attribute and parameter primitive type *encoding* may be specified by the developer as ENCODING_BIG_ENDIAN or ENCODING_LITTLE_ENDIAN. This specification is done by setting a .rti_encoding input parameter to one of these two values for primitive types. This can be seen in the source code for the functions, TrickHLA::Attribute.initialize() and TrickHLA::Parameter.initialize().

### 3.3.3 TrickHLA_12: HLA Encoding

**Summary.** TrickHLA must allow strings and/or byte arrays to be encoded as unicode strings, ASCII strings or opaque data (as defined in the HLA standard).

**Method.** Inspection

**Results.** For non-primitive attributes and parameters, the .rti_encoding input parameter may be specified as ENCODING_C_STRING, ENCODING_UNICODE_STRING, ENCODING_ASCII_STRING or ENCODING_OPAQUE_DAT
This can be seen in the source code for the functions, TrickHLA::Attribute.initialize() and TrickHLA::Parameter.initialize().

### 3.3.4 TrickHLA_13: Time Advancement

**Summary.** TrickHLA must support time stamped order HLA services.

**Method.** Inspection.

**Results.** Inspection of the TrickHLA::Federate class reveals that a federate built using TrickHLA may be

- time regulating (as indicated by the value of the boolean input flag, .time_regulating),
- time constrained (as indicated by the value of the boolean input flag, .time_constrained),

- both, or
- neither.

The HLA time advancement services invoked by TrickHLA are based on the values of these two flags.

### 3.3.5 TrickHLA_14: Lag Compensation

**Summary.** TrickHLA must provide optional support for sender- and receiver-side lag compensation.

**Method.** Inspection

**Results.** The class, TrickHLA::LagCompensation, provides this capability. It is not required, but may be used for sender- or receiver-side compensation. The TrickHLA *User Guide* discusses this class in more detail.

### 3.3.6 TrickHLA_15: Interactions

**Summary.** TrickHLA must support sending and receiving of interactions in receive order (RO) or time stamp order (TSO).

**Method.** Inspection

**Results.** The class, TrickHLA::InteractionHandler, provides this capability. It defines two send_interaction() methods, one of which is used to send receive order interactions and the other of which is used to send time stamp order interactions with some specified timetag. The class also defines a virtual method (which may be overridden in subclasses) which is invoked automatically whenever interactions (RO or TSO) arrive. The TrickHLA *User Guide* discusses this class in more detail.

### 3.3.7 TrickHLA_16: Ownership Transfer

**Summary.** TrickHLA must provide support for HLA ownership transfer.

**Method.** Inspection

**Results.** The class, TrickHLA::OwnershipHandler, provides this capability. it provides several push_ownership() methods that result in the federate *divesting* itself of ownership for the relevant attribute (only if the federate is the attribte's owner). The class also provides several pull_ownership() methods that result in the federate *acquiring* ownership of the relevant attribute if it has been divested by its owner. The TrickHLA *User Guide* discusses this class in more detail.

### 3.3.8 TrickHLA_17: Dynamic Initialization

**Summary.** TrickHLA must support dynamic initialization of an HLA federation in which the federates may exchange data before the simulation begins.

**Method.** Inspection

**Results.** NOTE: This section needs to be rewriten to not reference the DSES code.

TrickHLA supports this via the *multiphase initialization process.* This process is defined in *Distributed Space Exploration Simulation Multiphase Initialization Design* [5]. The machinery supporting this capability is exposed in the TrickHLA::ExecutionControlBase class, which has an input parameter consisting of a comma-separated list of synchronization point names, each of which corresponds to a different phase in the initialization process. The TrickHLA *User Guide* discusses how to construct a Trick S_define file that schedules initialization jobs for execution during each phase of this processes.

### 3.3.9 TrickHLA_18: Automatic Simulation Startup

**Summary.** TrickHLA must provide a mechanism for the various federates to synchronize with each other (i.e., for all of them to arrive) before the simulation begins in earnest.

**Method.** Inspection

**Results.** NOTE: This section needs to be rewriten to reference the TrickHLA::ExecutionControlBase class and associated implementations.

The TrickHLA::Federate class has several input parameters that may be used to specify a list of federates which must join the federation before the execution begins:

- .enable_known_feds, which enables/disables this feature,
- .known_feds_count, which specifies how many federations are to be governed by this mechanism,
- .known_feds, which is an array of size .known_feds_count of structures, each of which specifies the name of the federation and whether or not is must be present before the federation execution may begin.

The TrickHLA *User Guide* presents several examples in which this capability is used.

In addition to this capability, the `TrickHLA::ExecutionCoontorlBase` class provides a similar capability. The class has a parameter, `.required_federates`, which is a comma-separated list of the names of the federates that must be present in order for the federation execution to begin.

### 3.3.10 TrickHLA_19: Pack / Unpack of Simulation Data

**Summary.** `TrickHLA` must provide a mechanism for user specified code to be called to perform processing of data sent to or received from the HLA interface.

**Method.** Inspection

**Results.** The class, `TrickHLA::Packing`, provides the capability. It provides a pack() method that is called before data is sent through the HLA interface. It also provides an unpack() method that is called when data is received through the HLA interface. The `TrickHLA` *User Guide* discusses this class in detail.

### 3.3.11 TrickHLA_20: ObjectDeleted Callback

**Summary.** `TrickHLA` must provide a mechanism for notification of an object being deleted from the federation.

**Method.** Inspection

**Results.** The class, `TrickHLA::ObjectDeleted`, provides the capability. It provides a deleted() method that is called when an object is deleted from the federation. The `TrickHLA` *User Guide* discusses this class in detail.

### 3.3.12 TrickHLA_21: Federation Restore Callback

**Summary.** `TrickHLA` must provide a mechanism for a trick model to request a federation restore from the `RTI`.

**Method.** Inspection

**Results.** The class, `TrickHLA::Federate`, provides the capability. It provides a perform_restore() method that sends a completed federation restore request to the `TrickHLA::Manager` which, in turn, sends the request to the `RTI`. The `TrickHLA` *User Guide* discusses this class in detail.

### 3.3.13 TrickHLA_22: Federation Save Callback

**Summary.** TrickHLA must provide a mechanism for a trick model to request a federation save from the RTI.

**Method.** Inspection

**Results.** The class, TrickHLA::Manager, provides the capability. It provides a start_federation_save(), start_federation_save_at_sim_time() and start_federation_save_at_scenario_time() methods that starts the federation wide save process. The TrickHLA *User Guide* discusses this class in detail.

### 3.3.14 TrickHLA_23: Conditional sending of attributes

**Summary.** TrickHLA must provide a mechanism for a trick model to conditionally send attributes over the wire.

**Method.** Inspection

**Results.** The class, TrickHLA::Conditional, provides the capability. It provides a should_send() method that is called on each send cycle to identify if an attribute should be sent over the wire. The TrickHLA *User Guide* discusses this class in detail.

### 3.3.15 TrickHLA_24: Multiple verbose levels

**Summary.** TrickHLA must provide a mechanism to print multiple levels of information from the TrickHLA software. It also must provide a mechanism to allow the user to specify which TrickHLA module(s) shall print messages.

**Method.** Inspection

**Results.** The class, TrickHLA::DebugHandler, provides this capability. It provides a should_print() method, accepting a debug level and code section, returns true or false after determining if the message should be printed. The TrickHLA *Product Specification* discusses this class in detail.

# Bibliography

[1] Edwin Z. Crues. *Trick High Level Architecture (TrickHLA)*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[2] Edwin Z. Crues. *TrickHLA Product Requirements*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[3] Edwin Z. Crues. *TrickHLA Product Specification*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[4] Edwin Z. Crues. *TrickHLA User Guide*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[5] Dan E. Dexter. *Distributed Space Exploration Simulation Multiphase Initialization Design*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, December 2007.

[6] Dan E. Dexter. *Integrated Mission Simulation Multiphase Initialization Design*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2009.

[7] NASA. *NASA Software Engineering Requirements*. Technical Report NPR-7150.2C, National Aeronautics and Space Administration, NASA Headquarters, Washington, D.C., August 2019.

[8] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Documentation. Trick Wiki Site: `https://nasa.github.io/trick/documentation/Documentation-Home`, May 2020. Accessed 18 May 2020.

[9] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Installation guide. Trick Wiki Site: `https://nasa.github.io/trick/documentation/install_guide/Install-Guide`, May 2020 (accessed May 18, 2020). Accessed 18 May 2020.

[10] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Tutorial. Trick Wiki Site: https://nasa.github.io/trick/tutorial/Tutorial, May 2020 (accessed May 18, 2020). Accessed 18 May 2020.

[11] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Inferface Specification.* Technical Report IEEE-1516.1-2010, The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, August 2010.

[12] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.* Technical Report IEEE-1516-2010, The Institute of Electrical and Electronics Engineers, 2 Park Avenue, New York, NY 10016-5997, August 2010.

[13] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.* Technical Report IEEE-1516.2-2010, The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, August 2010.